Hibernate

Primary key auto Generators

Automatically Hibernate will generate 1 primary key as we are required to pass some id as it is not recommended to take primary key values from users as it may cause duplicate key error which violets the primary key constraints. So it is better that primary key is generated by system rather than taking its value from user.

Hibernate have given following primary key auto Generators

1. Assigned(Default): If we use this type of ID generators then user is responsible to provide the id in form which would store in table as primary key
2. Increment: If we user increment ID Generators then for next user entering values in the form for registration the primary key id will be generated by incrementing the previous record’s id by one.
3. Sequence:
4. Identity
5. Native
6. Hilo
7. Foreign
8. Custom Generators

Suppose we have an application where user come to book movie tickets. Consider following class for ticket booking.

Class BookTicket{

Private int tid;

Private Sring movie;

Private String show;

Private int seatno;

//getters

//Setters

}

Bookticket.hbm.xml

<hibernate-mapping>

<class name=”BookTicket” table=”TicketBooking” schema=”MAS”>

<id name=”tid” column=”id”>

<generate class=”assigned”>

“increment”

“Sequence”

“Identity”

“Native”

“Hilo”

</id>

<property name=” movie” column=” movie”/>

<property name=” show” column=” show”/>

<property name=” seatno” column=” seatno”/>

</class>

</hibernate-mapping>

In case of increment our application will take the max id from the database by running select max(id) from tablename and will increment it with ++ operator and will store the same id in id column of new record along with other data. **Here Application Layer is responsible for incrementing the id primary key column.**

If we are using increment property and our application is not thread safe then we multiple user can book the ticket at the same time then ticket id allocated to them can be same then there is chance of getting errors like inconsistent data.

Identity: **If we use this increment property type then Database layer only is responsible for incrementing the id by one.** If a new record is being inserted into the table and we need to pass the non key attribute values so that database can insert the data also along with id. In oracle database we don’t have this technique as it is not supported by oracle. In database like MySQL, DB2, SQL Server we have this technique by declaring our primary key column as identity column. So for this property be worked our database must support it.

Hibernate:

drop table if exists Mas.Student1

Hibernate:

create table Mas.Student1 (

id integer not null auto\_increment, //This column will be made identity //column

name varchar(255),

email varchar(255),

address varchar(255),

marks integer,

primary key (id)

)

Hibernate:

insert

into

Mas.Student1

(name, email, address, marks)

values

(?, ?, ?, ?)

**Sequence**: If we are using this increment property then both application and database layer is responsible for incrementing the id which is a primary key attribute. Because in this application will command **the database to create a sequence with name hibernate\_seq in the sche**ma that will be incremented and **its incremented value will be fetched by the hibernate application for each time a new row is inserted and will insert the same into the primary key column if the new record is inserted**.

It will create the following sequence

Create sequence hibernate\_sequence

Start with 1

Increment by 1

Nocycle

Nocache;

Configure as follows if we want our own sequences

<id name=*"id"* column=*"id"*>

<generator class=*"sequence"*>

<param name=*"sequence"*>stureg\_sequence</param>

</generator>

</id>

Sequences are not supported by sql database the following snippet is from application ran on db2 database

Hibernate:

drop table Mas.Student1

Hibernate:

drop sequence hibernate\_sequence restrict

Hibernate:

create table Mas.Student1 (

id integer not null,

name varchar(255),

email varchar(255),

address varchar(255),

marks integer,

primary key (id)

)

Hibernate:

create sequence hibernate\_sequence

Hibernate:

values

nextval for hibernate\_sequence

Hibernate:

insert

into

Mas.Student1

(name, email, address, marks, id)

values

(?, ?, ?, ?, ?)

If we change our hbm2ddl property to **update**

Hibernate:

values

nextval for hibernate\_sequence

Hibernate:

insert

into

Mas.Student1

(name, email, address, marks, id)

values

(?, ?, ?, ?, ?)

**Hilo: Application layer is responsible to store next high value and hibernate will create a hibernate\_hilo table.** At the time of first deployment it will have initial value as 0. Suppose we are deploying it again after one year then it will increment its value to 1 and so on. Consider application is deployed for the first time then hilo\_table value will be 0 and hibernate will check the hibernate\_hilo table value using following statement

If(next\_hi==0)

Deployment 1

Next\_hi==0 Nubmers in id column will start from= 1,2,3,4,…………

Deployment 2

Next\_hi==1 Numbers in id column will start from= 32767,…

Deployement 3

Next\_hi==2 Number in id column will start from=65536,….

So every deployment it will increment the Id primary key value with 32767

Hibernate:

drop table if exists Mas.Student1

Hibernate:

drop table if exists hibernate\_unique\_key

Hibernate:

create table Mas.Student1 (

id integer not null,

name varchar(255),

email varchar(255),

address varchar(255),

marks integer,

primary key (id)

)

Hibernate:

create table hibernate\_unique\_key (

next\_hi integer

)

Hibernate:

insert into hibernate\_unique\_key values ( 0 )

Hibernate:

select

next\_hi

from

hibernate\_unique\_key for update

Hibernate:

update

hibernate\_unique\_key

set

next\_hi = ?

where

next\_hi = ?

Hibernate:

insert

into

Mas.Student1

(name, email, address, marks, id)

values

(?, ?, ?, ?, ?)

When hbm2ddl property is changed to update from create

Hibernate:

select

next\_hi

from

hibernate\_unique\_key for update

Hibernate:

update

hibernate\_unique\_key

set

next\_hi = ?

where

next\_hi = ?

Hibernate:

insert

into

Mas.Student1

(name, email, address, marks, id)

values

(?, ?, ?, ?, ?)

**Native: It can use increment, identity or sequence** and it is dependent on database. It mean if database supports increment then hibernate will apply it, if will check for support for identity and if this also not supported the will go for sequence primary key generator type.

On using native generator in db2

Hibernate:

drop table db2admin5.Student1

Hibernate:

create table db2admin5.Student1 (

id integer generated by default as identity,

name varchar(255),

email varchar(255),

address varchar(255),

marks integer,

primary key (id)

)

Seesion Factory Init

Hibernate:

insert

into

db2admin5.Student1

(id, name, email, address, marks)

values

(default, ?, ?, ?, ?)

For Mysql it will generate following code

Hibernate:

drop table if exists Mas.Student1

Hibernate:

create table Mas.Student1 (

id integer not null auto\_increment,

name varchar(255),

email varchar(255),

address varchar(255),

marks integer,

primary key (id)

)

Hibernate:

insert

into

Mas.Student1

(name, email, address, marks)

values

(?, ?, ?, ?)

Consider following example of banking application where a new user comes for account opening and fills the form as follows

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Acc No | Name | Email | Phone | Balance |
| ICIC1001 | Masroor | mas@gmail.com | 9122013084 | 50000 |

Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Email:­­­­­­­­­­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Phone:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Balance:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Submit

On clicking submit button we should get Account no as given in the table and the number should be incremented each time by one. So our primary key will comprise of

ICIC + Number which can be incremented for each user registration

To achieve the primary number generation we need implement the custom primary key generator. We have to write ac custom primary key generator which we have to extend with **IdentityGenerator** Class. We alos implement our PK generator class with **IdentifierGenerator** Interface. By overriding the following method we can write inside our own logic for own custom PK Generator

**public** Serializable generate(SessionImplementor s, Object obj)

**HQL(Hibernate Query Language)**

HQL Hibernate Query Language we can make CRUD operations. Following CRUD operations

1. Insert: If we want to insert the data from one to table to another table then only we can user HQL to do the same. HQL doesn’t insert the user given data into table. We can only pass the data from one table to another table by using HQL
2. Update:
3. Delete:
4. Select:

Difference between HQL and SQL

HQL SQL

Hibernate related Database related

HQL Query are Object Oriented Queries SQL are table related Queries

Ex Select name from Student Ex: Select name from Student

|  |  |  |  |
| --- | --- | --- | --- |
| StuID | Name | EmailId | Marks |

Class Student{

Private String name;

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

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

}

In select parameters we need to use object In select parameters we need to write the field names

Propertied

Database Independent Database dependent

Doing insert operation using HQL. Consider following Example where we want to insert the data from oldStudent Table to NewStudent table

|  |  |  |
| --- | --- | --- |
| Id | Name | Email |
| 123 | Xyz | [xyz@gmail.com](mailto:xyz@gmail.com) |
| 124 | Pqr | [par@gmail.com](mailto:par@gmail.com) |
| 125 | Abc | [abc@gmail.com](mailto:abc@gmail.com) |

OldStudent table

|  |  |  |
| --- | --- | --- |
| Id | Name | Email |
|  |  |  |
|  |  |  |
|  |  |  |

NewStudent

Here fom doing HQL insert operation we need to write a beans class for each namely OldStudent and NewStudent and for doing insert operations we need to write following HQL statement

String hql= “insert into NewStudent(id, name, email)

Select s.id, s.name, s.email from OldStudent s)”

Query q= s.createQuery(hql);

Int i= q.executeUpdate();

Will run following Query inside the database

Hibernate:

insert

into

Mas.NewStudent

( id, name, email, adsress ) select

oldstudent0\_.id as col\_0\_0\_,

oldstudent0\_.name as col\_1\_0\_,

oldstudent0\_.email as col\_2\_0\_,

oldstudent0\_.adsress as col\_3\_0\_

from

Mas.OldStudent oldstudent0\_

No of row dumped from from old to new table3

In HQL we can update primary key values as well as we can update non-primary key values without giving other non-key values if we are updating one or more columns in the table unlike update() and merge() method which forces us to provide all the values of column apart from primary otherwise it will put null inside the table if don’t specify any value for other columns.

String hql=”update Student set id=124 where id=123”

Query q= session.createQuery(hql)

Int i= q.executeUpdate();

System.***out***.println("No of rows updated " + i);

Hibernate:

update

Mas.Student2

set

name='ABC',

email='ABC@gmail.com'

where

id=123

No of rows updated 0

In case of updating primary key

Hibernate:

update

Mas.Student2

set

id=222

where

id=111

No of rows updated 1

Delete operation using HQL

String hql= "delete Student where id=222";

Query q=session.createQuery(hql);

**int** i = q.executeUpdate();

t.commit();

System.***out***.println("No of rows deleted " + i);

Hibernate:

delete

from

Mas.Student2

where

id=222

No of rows deleted 1

Select operation using HQL

We have different types SQL Select operation depending upon the nature of record it fetches. It may fetch following types of records:

1. Single row:
2. Muti-row:
3. Single-value: Mostly when applying the Aggregate functions

Single row Select Operations

String hql= “from Student” //It will get all the records from student

String hql=”from Student where id=1” //will fetch single record where id is 1

uniqueObject() is used for 1 record fetching operations

list() is used for multi-record fetching operations

Query q= session.createQuery(hql)

Object o=q.uniqueObject();

Student s=(Student)o;

System.***out***.println(s.getId);

System.***out***.println(s.getName);

System.***out***.println(s.getEmail);

System.***out***.println(s.getAddress);

If we want to fetch all the values of 1 column of fields from database then we do same as follows

In case of retrieving 2 column values hibernate will store the result in the of list and list will contain the elements as Object array of 2 columns

String hql=”select name from Student”

Query q= session.createQuery(hql)

List<String> names=q.List();

for (String name:names){  
 System.***out***.println(name);

}

For fetching 2 or columns data we can do following procedures

String hql=”select name, email from Student”

Query q= session.createQuery(hql)

List<Object> list=q.List();

for (Object o: list){

Object objArray[]= (Object [])o;

for(Object val: objArray ){

System.***out***.println(val);

}

}

Or

for (Object o: list){

Object objArray[]= (Object [])o;

System.***out***.println(objArray[0]);

System.***out***.println(val[1]);

}

**Hibernate Criterias**

Using Criterias we can make select operation on table by applying the criteria ie giving the projections and selections.

If we want to perform SelectAll operations then we have method which can be invoked on session object as follows

Session s= sf.openSession();

Criteria c=s. createCriteria(Employee.class); //We want to apply criteria on Employee.class

List<Employee> emps= c.list();

The above criteria interface reference will internally run select \* from <schemaname>.tablename

Criteria is an Interface which has following methods to get the records

1. List();
2. uniqueObject();

In order to restrict the no of records i.e. for doing selection we need to write following code

Session s= sf.openSession();

Criteria c=s. createCriteria(Employee.class); //We want to apply criteria on Employee.class

Criterion cr=Restriction.eq(“id”, 1);

c.add(cr)

Employee emp=(Employee) c.uniqueResult();

If we want to add selection criteria as to fetch employees who have salary greater than 8000 then will write below criterion

Criterion cr= Restrictions.gt(“salary”, 8000)

c.add(cr)

List<Employee> emps= c.lsit();

Different methods that we can invoke on Restrictions static class

1. eq(“filedname”, “value”) : Equalto
2. gt(“fieldname”, value ): Greater Than
3. lt(“fieldname”, value): Less Than
4. between(5000, 8000):
5. distinct();

If we want values of 1 column only then we have to apply criterion which is given in Projections API

Projection p=Projections.property(“name”);

c.setProjection(p);

List<String> names= c.list();

it will generate query as Select name from Employee.

We can add multiple Restrictions to Criteria but we can add only one Projection Criteria. And if we try to apply any more projections to criteria then previous projection will be overridden by new Projection. But if we want to various projection class then we have ProjectionList class where we can add multiple Projections as follows

Projection p1= Projections.property(“email”);

ProjectionList pl = Projections.projectionList();

pl.add(p1);

pl.add(p2) ;

c.setProjection(pl);

List<Object> list=c.list();

If we want apply the aggregate functions on any numerical fields then we can do it as follows

Projection p= Projections.avg(“salary”)

c.setProjection(p);

double avgsal= (Double) c.uniqueResult();

For getting max salary of Employee

Projection p = Projections.max(“salary”);

c.setProjection(p);

float maxsal= (Float) c.uniqueResult ();

**ORM Relations**

UserDetails Emnployee Table

|  |  |  |
| --- | --- | --- |
| Aid | Username | Pwd |
| 1 | [abc@gmail.com](mailto:abc@gmail.com) | Abc123 |
| 2 | xyz@gmail.com | Xyz123 |

|  |  |  |
| --- | --- | --- |
| EmpId | Name | Depid |
| 100 | Masroor | 1 |
| 101 | Pradeep | 2 |

|  |  |
| --- | --- |
| Uid | Username |
| 1 | xbc |
| 2 | xyz |

|  |  |
| --- | --- |
| Did | DeptName |
| 1 | Developer |
| 2 | Tester |

Admin Table Dept Table

Fist figures shows is-a relationship where every admin can also be a user where his credential will be stored

Second figure shows has-a relationship as every employee has department whose details are stored into Department table

Hibernate provides following techniques to achieve **the is-a relationship which in java is known as Inheritance**

1. Table-per-Class
2. Table-per-SubClass
3. Table-per-ConcreteClass

**Association (has-a realtionship)**

1. One-to-many
2. Many-to-one
3. Many-to-many
4. One-to-one

Admin Reg

Id:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name\_\_\_\_\_\_\_\_\_\_\_\_\_

Email\_\_\_\_\_\_\_\_\_\_\_\_\_

Salary:\_\_\_\_\_\_\_\_\_\_\_\_\_

Branch:\_\_\_\_\_\_\_\_\_\_\_\_

Hardware Employee Reg

Id:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name\_\_\_\_\_\_\_\_\_\_\_\_\_

Email\_\_\_\_\_\_\_\_\_\_\_\_\_

Salary:\_\_\_\_\_\_\_\_\_\_\_\_\_

Working\_hours:\_\_\_\_\_\_\_

Software Employee Reg

Id:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name\_\_\_\_\_\_\_\_\_\_\_\_\_

Email\_\_\_\_\_\_\_\_\_\_\_\_\_

Salary:\_\_\_\_\_\_\_\_\_\_\_\_\_

Tool\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If we want to store the details of above forms into a table then we can store the same in one table beacause we have 4 common properties and only others properties are different.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Id | Name | Email | Salary | Tool | Working\_hours | Branch |
| 1 | Abc | [abc@gmail.com](mailto:abc@gmail.com) | 5000 | Hibernate |  |  |
| 2 | Xyz | [xyz@gmail.com](mailto:xyz@gmail.com) | 6000 |  | 9 |  |
| 3 | Admin123 | [Adm123@gmail.com](mailto:Adm123@gmail.com) | 8000 |  |  | SR Nagar |

Fro each from we need to create a bean class ie HardwareRmployee bean, SoftwareEmployee bean and Admin Bean we have to write three mapping files and will have to map each bean class to a single table only as to store all the values in single table only. As per java inheritance we can write common properties under a parent table ad can inherit those properties under child classes and we can write the other child classes related properties.

So in case of above example we can write common properties under common parent class Employee and can make three child class of it to inherit its property

Class employee{  
 private int id;

Private String name;

Private String email;

Private String salary;

}

Class SoftwareEmployee extends Employee{

Private String tool;

}

Class HardwareEmployee extends Employee{  
 private String workingHours

}

Class Admin extends Employee{

Private String branch;

}

To move these data into table we can map these three beans to a single mapping file. We can recognize the Software Employee if we find Tool as not null, Hardware Employee if workingHours Not Null and Admin if we find Branch as not null. What if Branch for Admin is not mandatory and it left null then how to find whether the record belongs to admin, Hardware Employee or Software Employee. To know whether record belongs to admin, Hardware Employee or Software Employee we need create a column known as **discriminator column** which will store some value to differentiate between them.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Id | Name | Email | Salary | Tool | Working\_hours | Branch | EmpType |
| 1 | Abc | [abc@gmail.com](mailto:abc@gmail.com) | 5000 | Hibernate |  |  | Se |
| 2 | Xyz | [xyz@gmail.com](mailto:xyz@gmail.com) | 6000 |  | 9 |  | He |
| 3 | Admin123 | [Adm123@gmail.com](mailto:Adm123@gmail.com) | 8000 |  |  | SR Nagar | Admin |
| 4 | Admin124 | [Adm124@gmail.com](mailto:Adm124@gmail.com) | 9000 |  |  |  | Admin |

Mapping file for mapping all the beans above

<hibernate-mapping>

<class name=”beans.Employee” table=”Employee” schema=”Mas”>

<id name=”id”></id>

<discriminator column=”emptype”

<property name=”name” column=” name”/>

<property name=”email” column=” email”/>

<property name=”salary” column=” salary”/>

<sub-class name=” beans.SoftwareEmployee” discriminator-value=”se”>

<property name=”tool” column=”tool”/>

</sub-class>

<sub-class name=”beans.HardwareEmployee” discriminator-value=”he”>

<property name=”workingHours” column=” workinghours”/>

</sub-class>

<sub-class name=”beans.Admin” discriminator-value=” admin”>

<property name=”Branch” column=”branch”/>

</sub-class>

</class>

</hibernate-mapping>

SQL Code generated wjhen we implement above using hibernate

Hibernate:

drop table if exists Mas.Employee3

Hibernate:

create table Mas.Employee3 (

id integer not null,

emptype varchar(255) not null,

name varchar(255),

email varchar(255),

salary integer,

tool varchar(255),

workingHours integer,

branch varchar(255),

primary key (id)

)

Hibernate:

insert

into

Mas.Employee3

(name, email, salary, tool, emptype, id)

values

(?, ?, ?, ?, 'se', ?)

Hibernate:

insert

into

Mas.Employee3

(name, email, salary, workingHours, emptype, id)

values

(?, ?, ?, ?, 'he', ?)

Hibernate:

insert

into

Mas.Employee3

(name, email, salary, branch, emptype, id)

values

(?, ?, ?, ?, 'admin', ?)

The table above that we have used is not normalized as we have several null values which can’t be utilized and hence will render the database as non-efficient and also we will have inconsistency in the database.

We can normalize the above as follows by breaking them as follows

Employee Table

|  |  |  |  |
| --- | --- | --- | --- |
| Id | Name | Email | Salary |
| 1 | Abc | [abc@gmail.com](mailto:abc@gmail.com) | 5000 |
| 2 | Xyz | [xyz@gmail.com](mailto:xyz@gmail.com) | 6000 |
| 3 | Admin123 | [Adm123@gmail.com](mailto:Adm123@gmail.com) | 8000 |
| 4 | Admin124 | [Adm124@gmail.com](mailto:Adm124@gmail.com) | 9000 |

|  |  |
| --- | --- |
| heid | WorkingHours |
| 2 | 9 |

SoftwareEmployee Table HardwareEmployee Table

|  |  |
| --- | --- |
| Seid | Tool |
| 1 | Hibernate |

Admin Table

|  |  |
| --- | --- |
| Aminid | Branch |
| 3 | SR Nagar |
| 4 |  |

Mapping file for achieving above result

<hibernate-mapping>

<class name=*"beans.Employee"* table=*"Employee4"* schema=*"Mas"*>

<id name=*"id"*></id>

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

<property name=*"email"* column=*"email"*/>

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

<joined-subclass name=*"beans.SoftwareEmployee"* table=*"SoftwareEmployee"*>

<key column=*"id"*/>

<property name=*"tool"* column=*"tool"*/>

</joined-subclass>

<joined-subclass name=*"beans.HardwareEmployee"* table=*"HardwareEmployee"*>

<key column=*"id"*/>

<property name=*"workingHours"* column=*"workinghours"*/>

</joined-subclass>

<joined-subclass name=*"beans.Admin"* table=*"Admin"*>

<key column=*"id"*/>

<property name=*"branch"* column=*"branch"*></property>

</joined-subclass>

</class>

</hibernate-mapping>

SQL staemtents being run by hibernate internally to interact with database are:

Hibernate:

alter table Admin

drop

foreign key FK\_as1rmx4ofyhjguia6ccyf1enx

Hibernate:

alter table HardwareEmployee

drop

foreign key FK\_avagm624ejeemokhc13bab8k0

Hibernate:

alter table SoftwareEmployee

drop

foreign key FK\_dye98f2scj0ecvxygg03qq5eu

Hibernate:

drop table if exists Admin

Hibernate:

drop table if exists HardwareEmployee

Hibernate:

drop table if exists Mas.Employee4

Hibernate:

drop table if exists SoftwareEmployee

Hibernate:

create table Admin (

id integer not null,

branch varchar(255),

primary key (id)

)

Hibernate:

create table HardwareEmployee (

id integer not null,

workinghours integer,

primary key (id)

)

Hibernate:

create table Mas.Employee4 (

id integer not null,

name varchar(255),

email varchar(255),

salary integer,

primary key (id)

)

Hibernate:

create table SoftwareEmployee (

id integer not null,

tool varchar(255),

primary key (id)

)

Hibernate:

alter table Admin

add constraint FK\_as1rmx4ofyhjguia6ccyf1enx

foreign key (id)

references Mas.Employee4 (id)

Hibernate:

alter table HardwareEmployee

add constraint FK\_avagm624ejeemokhc13bab8k0

foreign key (id)

references Mas.Employee4 (id)

Hibernate:

alter table SoftwareEmployee

add constraint FK\_dye98f2scj0ecvxygg03qq5eu

foreign key (id)

references Mas.Employee4 (id)

Hibernate:

insert

into

Mas.Employee4

(name, email, salary, id)

values

(?, ?, ?, ?)

Hibernate:

insert

into

SoftwareEmployee

(tool, id)

values

(?, ?)

Hibernate:

insert

into

Mas.Employee4

(name, email, salary, id)

values

(?, ?, ?, ?)

Hibernate:

insert

into

HardwareEmployee

(workinghours, id)

values

(?, ?)

Hibernate:

insert

into

Mas.Employee4

(name, email, salary, id)

values

(?, ?, ?, ?)

Hibernate:

insert

into

Admin

(branch, id)

values

(?, ?)

If we go for **Table Per Concrete class** approach for creating tables then it will create the three different different tables each for HardwareEmployee, SoftwareEmployee and Admin table respectively as follows:

SoftwareEmployee table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Name | Email | Salary | Tool |
| 1 | Abc | [abc@gmail.com](mailto:abc@gmail.com) | 5000 | hibernate |

HardwareEmployee Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Name | Email | Salary | Workinghours |
| 2 | Xyz | [xyz@gmail.com](mailto:xyz@gmail.com) | 6000 | 9 |

Admin

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Id | Name | Email | Salary | Branch |
| 3 | Admin123 | [Adm123@gmail.com](mailto:Adm123@gmail.com) | 8000 | SR Nagar |
| 4 | Admin124 | [Adm124@gmail.com](mailto:Adm124@gmail.com) | 9000 |  |

Hibernate mapping file for the same is as below

<hibernate-mapping>

<class name=*"beans.Employee"* schema=*"Mas"*>

<id name=*"id"*></id>

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

<property name=*"email"* column=*"email"*/>

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

<union-subclass name=*"beans.SoftwareEmployee"* table=*"SoftwareEmployee1"*>

<property name=*"tool"* column=*"tool"*/>

</union-subclass>

<union-subclass name=*"beans.HardwareEmployee"* table=*"HardwareEmployee1"*>

<property name=*"workingHours"* column=*"workinghours"*/>

</union-subclass>

<union-subclass name=*"beans.Admin"* table=*"Admin1"*>

<property name=*"branch"* column=*"branch"*/>

</union-subclass>

</class>

</hibernate-mapping>

It will create the employee table also but will not insert any data into the table. Following are the commands executed by hibernate internally on the database

Hibernate:

drop table if exists Admin1

Hibernate:

drop table if exists HardwareEmployee1

Hibernate:

drop table if exists Mas.Employee

Hibernate:

drop table if exists SoftwareEmployee1

Hibernate:

create table Admin1 (

id integer not null,

name varchar(255),

email varchar(255),

salary integer,

branch varchar(255),

primary key (id)

)

Hibernate:

create table HardwareEmployee1 (

id integer not null,

name varchar(255),

email varchar(255),

salary integer,

workinghours integer,

primary key (id)

)

Hibernate:

create table Mas.Employee (

id integer not null,

name varchar(255),

email varchar(255),

salary integer,

primary key (id)

)

Hibernate:

create table SoftwareEmployee1 (

id integer not null,

name varchar(255),

email varchar(255),

salary integer,

tool varchar(255),

primary key (id)

)

Hibernate:

insert

into

SoftwareEmployee1

(name, email, salary, tool, id)

values

(?, ?, ?, ?, ?)

Hibernate:

insert

into

HardwareEmployee1

(name, email, salary, workinghours, id)

values

(?, ?, ?, ?, ?)

Hibernate:

insert

into

Admin1

(name, email, salary, branch, id)

values

(?, ?, ?, ?, ?)

**Has-a relationship**

1. One-to Many

Consider following example of has-a one-to-many relationship

|  |  |  |
| --- | --- | --- |
| Mid | Name | Aic |
| 1 | pk | 1 |
| 2 | Lagan | 1 |

Actor Table Movie Table

|  |  |
| --- | --- |
| Aid | Name |
| 1 | Aamir Khan |
| 2 | Sharukh Khan |

In above example the movie table has movie details and also has the actor id and actor table is maintained for actor details i.e. Aid in Movie table is foreign key for primary key in Actor table.

Hibernate mapping for Actor Bean class

<hibernate-mapping>

<class name=*"beans.Actor"* table=*"Actor"* schema=*"Mas"*>

<id name=*"aid"* column=*"aid"*/>

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

<set name=*"movies"* table=*"movie"*>

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

<one-to-many class=*"beans.Movie"*/>

</set>

</class>

</hibernate-mapping>

Where set tag denotes that Movie bean can exist inside as Many which means one actor can do several movies

Following are the SQL statements run by Hibernate on MySQL database

Hibernate:

alter table Mas.Movie

drop

foreign key FK\_d9gqhlm6jgypig433ptdk145w

Hibernate:

drop table if exists Mas.Actor

Hibernate:

drop table if exists Mas.Movie

Hibernate:

create table Mas.Actor (

aid integer not null,

name varchar(255),

primary key (aid)

)

Hibernate:

create table Mas.Movie (

mid integer not null,

moviename varchar(255),

aid integer,

primary key (mid)

)

Hibernate:

alter table Mas.Movie

add constraint FK\_d9gqhlm6jgypig433ptdk145w

foreign key (aid)

references Mas.Actor (aid)

Hibernate:

insert

into

Mas.Movie

(moviename, mid)

values

(?, ?)

Hibernate:

insert

into

Mas.Movie

(moviename, mid)

values

(?, ?)

Hibernate:

insert

into

Mas.Actor

(name, aid)

values

(?, ?)

Hibernate:

update

Mas.Movie

set

aid=?

where

mid=?

Hibernate:

update

Mas.Movie

set

aid=?

where

mid=?

If we add cascade=”all” to <set> tag then we do not have to do session.save(m1) and session.save(m2) as it will taken care by hibernate to persist the same into database automatically.

<hibernate-mapping>

<class name=*"beans.Actor"* table=*"Actor"* schema=*"Mas"*>

<id name=*"aid"* column=*"aid"*/>

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

<set name=*"movies"* table=*"movie"* cascade=*"all"* >

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

<one-to-many class=*"beans.Movie"*/>

</set>

</class>

</hibernate-mapping>

**Many to 1 has-a Relationship**

Movie table Actor Table

|  |  |  |
| --- | --- | --- |
| Mid | Name | Aic |
| 1 | pk | 1 |
| 2 | Lagan | 1 |

|  |  |
| --- | --- |
| Aid | Name |
| 1 | Aamir Khan |
| 2 | Sharukh Khan |

The relationship above is Many movies can be done by a single actor.

Mapping for Movies bean class will be

<hibernate-mapping>

<class name=*"beans.Movie"* table=*"Movie"* schema=*"Mas"*>

<id name=*"mid"* column=*"mid"*/>

<property name=*"movieName"* column=*"moviename"*/>

<many-to-one name=*"actor"* class=*"beans.Actor"*>

<column name=*"aid"*></column>

</many-to-one>

</class>

</hibernate-mapping>

SQL Command ran by Hibernate Internally on the MySQL Database are:

Hibernate:

alter table Mas.Movie

drop

foreign key FK\_d9gqhlm6jgypig433ptdk145w

Hibernate:

drop table if exists Mas.Actor

Hibernate:

drop table if exists Mas.Movie

Hibernate:

create table Mas.Actor (

aid integer not null,

name varchar(255),

primary key (aid)

)

Hibernate:

create table Mas.Movie (

mid integer not null,

moviename varchar(255),

aid integer,

primary key (mid)

)

Hibernate:

alter table Mas.Movie

add constraint FK\_d9gqhlm6jgypig433ptdk145w

foreign key (aid)

references Mas.Actor (aid)

Hibernate:

select

actor\_.aid,

actor\_.name as name2\_0\_

from

Mas.Actor actor\_

where

actor\_.aid=?

Hibernate:

insert

into

Mas.Movie

(moviename, aid, mid)

values

(?, ?, ?)

Hibernate:

insert

into

Mas.Movie

(moviename, aid, mid)

values

(?, ?, ?)

Hibernate:

insert

into

Mas.Actor

(name, aid)

values

(?, ?)

Hibernate:

update

Mas.Movie

set

moviename=?,

aid=?

where

mid=?

Hibernate:

update

Mas.Movie

set

moviename=?,

aid=?

where

mid=?

Has-a 1to1 relationship

Foreign Generator is used when we want to generate the primary key in a table and the same primary key can be generated inside the other table which is referenced by the former table.

Consider following tables

Voter table Vote table

|  |  |  |
| --- | --- | --- |
| Vid | VName | Age |
| 1 | Xyz | 25 |
| 2 | Abc | 20 |

|  |  |  |
| --- | --- | --- |
| VoterId | PartyName | CastDate |
| 1 | BJP | 25/11/2015 |
| 2 | BDP | 24/11/2016 |

The above tables relationship is one-to-one ie one Voter can vote to single party only so the relationship between Voter and Vote table is one-to-one. In order to implement we have to write following mapping file for Vote Bean class

<hibernate-mapping>

<class name=*"model.Vote"* table=*"vote"* schema=*"mas"*>

<id name=*"voterid"*>

<generator class=*"foreign"*>

<param name=*"property"*>voter</param>

</generator>

</id>

<property name=*"partyName"* column=*"partyname"*/>

<property name=*"castDate"* column=*"castdate"*/>

<one-to-one name=*"voter"* class=*"model.Vote"*></one-to-one>

</class>

</hibernate-mapping>

If we try to save mutilple voterid for a single party in the vote table then following exceptions will occur at RunTime.

org.hibernate.NonUniqueObjectException: A different object with the same identifier value was already associated with the session

following SQL commands will be executed by Hibernate at runtime for inserting the data according to one-to-one relationship between tables

Hibernate:

drop table if exists mas.vote

Hibernate:

drop table if exists mas.voter

Hibernate:

create table mas.vote (

voterid integer not null,

partyname varchar(255),

castdate datetime,

primary key (voterid)

)

Hibernate:

create table mas.voter (

vid integer not null,

vname varchar(255),

age integer,

primary key (vid)

)

Hibernate:

insert

into

mas.voter

(vname, age, vid)

values

(?, ?, ?)

Hibernate:

insert

into

mas.vote

(partyname, castdate, voterid)

values

(?, ?, ?)

Has-A Many to Many Realtionship

Consider following example of many to many relationship

The SQL Commands executed by Hibernate interanally into the database to achieve this result is

Hibernate:

alter table facultycourse

drop

foreign key FK\_pcpl5s7mb0ry51ypni29t5qc1

Hibernate:

alter table facultycourse

drop

foreign key FK\_rcj6i6hgljyc8trl12xb7rxeh

Hibernate:

drop table if exists Mas.course

Hibernate:

drop table if exists Mas.faculty

Hibernate:

drop table if exists facultycourse

Hibernate:

create table Mas.course (

cid integer not null,

cname varchar(255),

fee integer,

primary key (cid)

)

Hibernate:

create table Mas.faculty (

fid integer not null,

fname varchar(255),

totalexp integer,

primary key (fid)

)

Hibernate:

create table facultycourse (

cid integer not null,

fid integer not null,

primary key (fid, cid)

)

Hibernate:

alter table facultycourse

add constraint FK\_pcpl5s7mb0ry51ypni29t5qc1

foreign key (fid)

references Mas.faculty (fid)

Hibernate:

alter table facultycourse

add constraint FK\_rcj6i6hgljyc8trl12xb7rxeh

foreign key (cid)

references Mas.course (cid)

Hibernate:

select

faculty\_.fid,

faculty\_.fname as fname2\_1\_,

faculty\_.totalexp as totalexp3\_1\_

from

Mas.faculty faculty\_

where

faculty\_.fid=?

Hibernate:

select

faculty\_.fid,

faculty\_.fname as fname2\_1\_,

faculty\_.totalexp as totalexp3\_1\_

from

Mas.faculty faculty\_

where

faculty\_.fid=?

Hibernate:

select

faculty\_.fid,

faculty\_.fname as fname2\_1\_,

faculty\_.totalexp as totalexp3\_1\_

from

Mas.faculty faculty\_

where

faculty\_.fid=?

Hibernate:

select

course\_.cid,

course\_.cname as cname2\_0\_,

course\_.fee as fee3\_0\_

from

Mas.course course\_

where

course\_.cid=?

Hibernate:

select

course\_.cid,

course\_.cname as cname2\_0\_,

course\_.fee as fee3\_0\_

from

Mas.course course\_

where

course\_.cid=?

Hibernate:

select

course\_.cid,

course\_.cname as cname2\_0\_,

course\_.fee as fee3\_0\_

from

Mas.course course\_

where

course\_.cid=?

Hibernate:

select

course\_.cid,

course\_.cname as cname2\_0\_,

course\_.fee as fee3\_0\_

from

Mas.course course\_

where

course\_.cid=?

Hibernate:

insert

into

Mas.faculty

(fname, totalexp, fid)

values

(?, ?, ?)

Hibernate:

insert

into

Mas.faculty

(fname, totalexp, fid)

values

(?, ?, ?)

Hibernate:

insert

into

Mas.faculty

(fname, totalexp, fid)

values

(?, ?, ?)

Hibernate:

insert

into

Mas.course

(cname, fee, cid)

values

(?, ?, ?)

Hibernate:

insert

into

Mas.course

(cname, fee, cid)

values

(?, ?, ?)

Hibernate:

insert

into

Mas.course

(cname, fee, cid)

values

(?, ?, ?)

Hibernate:

insert

into

Mas.course

(cname, fee, cid)

values

(?, ?, ?)

Hibernate:

insert

into

facultycourse

(fid, cid)

values

(?, ?)

Hibernate:

insert

into

facultycourse

(fid, cid)

values

(?, ?)

Hibernate:

insert

into

facultycourse

(fid, cid)

values

(?, ?)

Hibernate:

insert

into

facultycourse

(fid, cid)

values

(?, ?)

Hibernate:

insert

into

facultycourse

(fid, cid)

values

(?, ?)

Hibernate:

insert

into

facultycourse

(fid, cid)

values

(?, ?)

Hibernate:

insert

into

facultycourse

(fid, cid)

values

(?, ?)

**POJO Classes mapping to Table by using Annotations**

By using above approach we can get rid of mapping xml file we can just use annotations to map our POJO class with corresponding database tables.

In mysql.cfg.xml we have to give the path of mapping file

<mapping resource=”resources/c.hbm.xml”>

Instead of using above mapping we can use pojo class mapping as follows

<mapping class=”POJO class”>

To process the cfg.xml file containing the resource mapping xml file we have to use the container class called Configuration as follows.

Configuration cfg = new Configuration();

Cfg.configure(“mysql.cgf.xml”);

To process annotation based mappings we have to use AnnotationConfiguration Class as follows

Configuration cfg= new AnnonationConfiguration();

Cfg.configure(“mysql.cgf.xml”);

**Hibernate 4.x** onwards for both xml and annotation based we can use one Container class only that is Congiguration as follows

Configuration cfg = new Configuration();

Cfg.configure(“mysql.cgf.xml”);

**Hibernate Validations**

The famous api to impelement the Validation is jsr303 Validations. JSR stands for Java Standard Releases. From hibernate 4.x we have these validations. We will be doing annotation bases validations