
# Hibernate Mappings

# Agenda

1. Mappings

2. Hibernate Application : Using One to One Mapping

# Objectives

At the end of this session, you will be able to:

- Understand about Mappings in hibernate
- Understand how to implement One to One Mapping in hibernate

# Mappings

# Mappings

- Mapping in hibernate expresses the association between the Object (Persistent Class Object) and the Relational Model (As Database Tables).
- Object model represents relationships in terms of object references and inheritance
- Where as Relational model represents relationships in terms of keys (primary & foreign keys)
- With previous examples we just mapped single primitive variables with columns of the table
- Now let us look at properties of the persistent class as another class object or a collection object
- How do we map this to the relational model
- Hibernate Mappings can be classified as
  - **Collection mapping**
  - **Association mapping**
  - **Component Mapping**

# Collection Mapping

- When one or more properties of the persistent class is a collection object like Array, List, Map, Set or Bag
- Which is a group of values associated to a single property
- These collections have their own style of storing and retrieving of data
- Thus Hibernate also provides different mappings for persisting the collection type property
- Collection types can be implemented with one of the following database relationships
  - **One to Many**
  - **Many to One**
  - **Many to Many**

# Association Mapping

- Association between two tables in the database can be expressed as cardinality
- There are 4 ways to achieve this
  - **One to One**
  - **One to Many**
  - **Many to One**
  - **Many to Many**
- An association mapping in hibernate can be implemented as unidirectional or bidirectional
- unidirectional or bidirectional defines the direction in which you can use the relationship in your model and Criteria queries

# Component Mapping

- When the 2 classes are related with Has-A relationship, It can be related in database as
  - **One to One cardinality with 2 tables**
  - **Embedded with the same table as a component (i.e. columns)**
- Component Mapping is when the composed class is not required to be created as a separate table in the database
- Instead the fields of the composed class is added as attributes to the base class table

```
@Entity
@Table(name="STUDENT_TBL")
public class Student {
@Id
@Column(name="stdid")
private int studentId;
@Column(name="stdName",length=20,nullable=false)
private String studentName;
private String course;
@Embedded
private Address studentAddress;
}
```

| STUDENT_TBL |
| --- |
| stdid |
| stdName |
| course |
| address_Id |
| address_streetName |
| address_city |
| address_pin |

# Hibernate Application : Using One to One Mapping

# Hibernate Application : Using One to One Mapping

**To do List:**

- Create a Maven Project (Simple : Skip Archetype Selection)
- Add ojdbc6.jar (to the class path)
- Add hibernate dependencies to the POM.xml

```
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.4.9.Final</version>
</dependency>
```

- Create hibernate.cfg.xml to define the hibernate and DB connection configurations
- Create Address Bean class with required hibernate mappings
- Create Student Bean class with required hibernate mappings
- Create StudentDAO class to preform the CRUD operations
- Create StudentService class to test the CRUD operations

# Hibernate Application : Using One to One Mapping

### Creating Hibernate.cfg.xml

- Hibernate configuration file would the same as old one
- Two mapping class tags has to be added as we would be referring to 2 classes (Student bean and Address bean)

```
<hibernate-configuration>
<session-factory>
<property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
<property name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
<property name="hibernate.connection.username">scott</property>
<property name="hibernate.connection.password">tiger</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.wipro.bean.Student"/>
<mapping class="com.wipro.bean.Address"/>
</session-factory>
</hibernate-configuration>
```

# Hibernate Application : Using One to One Mapping

## Creating Address Bean

- Create Address class under com.wipro.bean package, with all required fields and its getter setter methods

```
package com.wipro.bean;

import javax.persistence.*;

@Entity
@Table(name="Address_TBL")
public class Address {
@Id
private int addressId;
@Column(length = 20)
private String streetName;
@Column(length = 20)
private String city;
@Column(length = 20)
private String pin;
```

```
public Address() {//default constructor}
public Address(int addressId, String streetName, String
city, String pin) { //parameterized constructor
this.addressId = addressId;
this.streetName = streetName;
this.city = city;
this.pin = pin;
}
public int getAddressId() {
return addressId;
} /** All getter setters are included */
@Override
public String toString() {
return "Address [addressId=" + addressId + ", streetName=" +
streetName + ", city=" + city + ", pin=" + pin + "]";
} }
```

# Hibernate Application : Using One to One Mapping

## Creating Student Bean

- Create Student class under com.wipro.bean package, with all required fields and its getter setter methods
- On the address field add the @OneToOne annotation to indicate the one to one mapping
- With @OneToOne we also give optional parameters

```
package com.wipro.bean;

import javax.persistence.*

@Entity
@Table(name="STUDENT_TBL")
public class Student {
@Id
@Column(name="stdid")
private int studentId;
@Column(name="stdName",length=20,nullable=false)
private String studentName;
@Column(length = 20)
private String course;
@OneToOne
private Address address;
/** All Constructors , Getters & Setters and
toString methods are given */
}
```

# Hibernate Application : Using One to One Mapping

## Creating Student DAO class

- Create StudentDAO class under the com.Wipro.dao package
- This class contains the methods to perform the database related transactions
- InsertStudent method is created to insert record into the Student table and address table

```
public class StudentDAO {
SessionFactory sessionFactory;
public StudentDAO() {
sessionFactory = new Configuration().
            configure().buildSessionFactory();
}
public String insertStudent(Student student)
{
Session session =
sessionFactory.openSession();
Transaction trx = session.beginTransaction();
      session.persist(student.getAddress());
      session.persist(student);
trx.commit();
return "success";
}
}
```

# Hibernate Application : Using One to One Mapping

## Creating StudentService class

- Create StudentService class under the com.Wipro.service package to test the created functionality

```
public class StudentService {

	public static void main(String[] args) {
	Address address = new Address(101, "Tiger Street", "Bangalore", "572001");
	Student student = new Student(1001, "Sandy", "Java", address);
	StudentDAO dao = new StudentDAO();
	System.out.println("Result : "+dao.insertStudent(student));
	}
}
```

# Hibernate Application : Using One to One Mapping

### Understanding the Execution result

- On executing the StudentService class we can see both student and address class getting created and records getting inserted.

```
Hibernate: create table Address1_TBL (addressId number(10,0) not null, city varchar2(20 char), pin
varchar2(20 char), streetName varchar2(20 char), primary key (addressId))

Hibernate: create table STUDENT_TBL (stdid number(10,0) not null, course varchar2(20 char), stdName
varchar2(20 char) not null, address_addressId number(10,0), primary key (stdid))

Hibernate: alter table STUDENT_TBL add constraint FKawdqkipqtw1frxhds8xy86ke0 foreign key
(address_addressId) references Address1_TBL
```

```
Hibernate: insert into Address1_TBL (city, pin, streetName, addressId) values (?, ?, ?, ?)
Hibernate: insert into STUDENT_TBL (address_addressId, course, stdName, stdid) values (?, ?, ?, ?)
Result : success
```

# Hibernate Application : Using One to One Mapping

## Database Status after Execution

- Table Description

```
SQL> desc student_tbl;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 STDID                                     NOT NULL NUMBER(10)
 COURSE                                             VARCHAR2(20 CHAR)
 STDNAME                                   NOT NULL VARCHAR2(20 CHAR)
 ADDRESS_ADDRESSID                                  NUMBER(10)

SQL> desc address1_tbl;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------
 ADDRESSID                                 NOT NULL NUMBER(10)
 CITY                                               VARCHAR2(20 CHAR)
 PIN                                                VARCHAR2(20 CHAR)
 STREETNAME                                         VARCHAR2(20 CHAR)

SQL>
```

```
SQL> select * from student_tbl;

     STDID COURSE               STDNAME              ADDRESS_ADDRESSID
---------- -------------------- -------------------- -----------------
      1002 Java                 Sandy                              101

SQL> select * from address1_tbl;

 ADDRESSID CITY                 PIN                  STREETNAME
---------- -------------------- -------------------- --------------------
       101 Bangalore            572001               Tiger Street
```

# Mapping Annotation Parameters

- There are optional Parameters with Mapping Annotations
  - cascade – operations that can be cascaded to the target
  - targetEntity – name of the target entity class
  - mappedBy – field denoting relationship, can be used to expressed explicit
  - orphanRemoval – to apply or not the removal of operations from the targets which have been removed from relationship base entity

```
@OneToOne(cascade =
CascadeType.PERSIST)
private Address address;
```

```
public String insertStudent(Student student) {
System.out.println("Insert called");
Session session = sessionFactory.openSession();
Transaction trx = session.beginTransaction();
//session.persist(student.getAddress());
/**
 * Gets automatically persisted
 * As the operation is cascaded
 */
session.persist(student);
trx.commit();
return "success";
}
```

# Summary

In this session we learnt:

- Mappings in hibernate
    - Collection Mapping
    - Association Mapping
    - Component Mapping
- how to implement One to One Mapping in hibernate


# Thank You