Hibernate Data Types

From the first chapter, I am saying that Hibernate works as a transformer it converts Object to database row or vice versa. Now, If we minutely follow the statement one question may arise in our mind is.

**How Java data type saved to Database data type?**

This is a very valid question, and the answer is Hibernate defines a new kind of Datatype which purely belongs to Hibernate we called them Hibernate Data types. Which acts as a transformer takes Java data type and convert it underlying Database data type. So whatever we declare in Java POJO using @Type it will be a Hibernate type.

For all possible Java and Database type, there is a mapping type defined by Hibernate, In this chapter, we will know about those types.

**Primitive type**

|  |  |  |
| --- | --- | --- |
| **Hibernate Type** | **Java Type** | **RDBMS Type** |
| **integer** | **Integer/int** | **INTEGER** |
| **long** | **Long/long** | **BIGINT** |
| **short** | **Short/short** | **SMALLINT** |
| **float** | **Float/float** | **FLOAT** |
| **double** | **Double/double** | **Double** |
| **big\_decimal** | **BigDecimal** | **Numeric** |
| **character** | **String** | **CHAR(1)** |
| **string** | **String** | **VARCHAR** |
| **byte** | **Byte/byte** | **TINYINT** |
| **boolean** | **Boolean/boolean** | **BIT** |
| **yes/no** | **Boolean/boolean** | **CHAR(1) ('Y' or 'N')** |
| **true/false** | **Boolean/boolean** | **CHAR(1) ('T' or 'F')** |

**Java To RDBMS type**

|  |  |  |
| --- | --- | --- |
| **Hibernate Type** | **Java Type** | **RDBMS Type** |
| **class** | **Class** | **VARCHAR** |
| **locale** | **Locale** | **VARCHAR** |
| **timeZone** | **TimeZone** | **VARCHAR** |
| **currency** | **Currency** | **VARCHAR** |

**Binary Object type**

|  |  |  |
| --- | --- | --- |
| **Hibernate Type** | **Java Type** | **RDBMS Type** |
| **binary** | **byte[]** | **BLOB** |
| **text** | **String** | **CLOB** |
| **serializable** | **Serializable** | **BLOB** |
| **clob** | **CLOB** | **CLOB** |
| **blob** | **BLOB** | **BLOB** |

**Date Types**

|  |  |  |
| --- | --- | --- |
| **Hibernate Type** | **Java Type** | **RDBMS Type** |
| **date** | **java.util.Date / java.sql.Date** | **DATE** |
| **time** | **java.util.Date / java.sql.Time** | **TIME** |
| **timestamp** | **java.util.Date / java.sql.Timestamp** | **TIMESTAMP** |
| **calendar** |  | **TIMESTAMP** |
| **calendar\_date** | **Calendar** | **DATE** |

**Datatype conversion**



**Example :**

**Player POJO**

package com.example.hibernate.pojo;

import java.util.Date;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@org.hibernate.annotations.Entity(dynamicUpdate = true)

@Table(name = "Player")

public class Player {

@Id

@Column(name = "ID", unique = true, nullable = false)

private Integer id;

@Column(name = "NAME", nullable = false, length = 100)

private String name;

@Column(name = "BIRTH\_DATE",nullable = false)

private Date birthDate;

@Column(name = "SEX", length = 1)

private Character gender;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Date getBirthDate() {

return birthDate;

}

public void setBirthDate(Date birthDate) {

this.birthDate = birthDate;

}

public Character getGender() {

return gender;

}

public void setGender(Character gender) {

this.gender = gender;

}

@Override

public String toString() {

return "Player [employeeId=" + id + ", name=" + name

+ ", birthDate=" + birthDate + ", gender=" + gender + "]";

}

}

**hibernate.cfg.xml**

<?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>

<property name="hibernate.archive.autodetection">class,hbm</property>

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

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

<property name="hibernate.connection.driver\_class">org.hsqldb.jdbcDriver</property>

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

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

<property name="hibernate.connection.url">jdbc:hsqldb:mem:hibernatetutorial</property>

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

<mapping class="com.example.hibernate.pojo.Player"/>

</session-factory>

</hibernate-configuration>

**Hibernate Test**

package com.example.hibernate.test;

import java.util.Date;

import org.hibernate.Session;

import com.example.hibernate.pojo.Employee;

import com.example.hibernate.pojo.Player;

import com.example.hibernate.util.HibernateUtil;

public class HibernateTest

{

public static void main(String[] args)

{

HibernateTest test = new HibernateTest();

test.inserPlayer();

test.showPlayer();

HibernateUtil.shutdown();

}

public void inserPlayer()

{

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

session.beginTransaction();

// Add new Employee object

Player player = new Player();

player.setId(1);

player.setBirthDate(new Date("12/03/1983"));

player.setName("Shamik Mitra");

player.setGender('M');

session.save(player);

session.getTransaction().commit();

}

public void showPlayer()

{

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

session.beginTransaction();

// Add new Employee object

Player player = (Player) session.get(Player.class, new Integer(1));

System.out.println(player);

session.getTransaction().commit();

}

}

Output

Hibernate: drop table Player if exists

Hibernate: create table Employee (ID integer not null, EMAIL varchar(100) not null, FIRST\_NAME varchar(100) not null, LAST\_NAME varchar(100) not null, primary key (ID))

Hibernate: create table Player (ID integer not null, BIRTH\_DATE timestamp not null, SEX char(1), NAME varchar(100) not null, primary key (ID))

Hibernate: alter table Employee add constraint UK\_ardf0f11mfa6tujs3hflthwdv unique (EMAIL)

Hibernate: insert into Player (BIRTH\_DATE, SEX, NAME, ID) values (?, ?, ?, ?)

Hibernate: select player0\_.ID as ID1\_1\_0\_, player0\_.BIRTH\_DATE as BIRTH\_DA2\_1\_0\_, player0\_.SEX as SEX3\_1\_0\_, player0\_.NAME as NAME4\_1\_0\_ from Player player0\_ where player0\_.ID=?

Player [employeeId=1, name=Shamik Mitra, birthDate=1983-12-03 00:00:00.0, gender=M]

Here I create a Player entity and mapped the same entity in hibernate.cfg.xml.

After that, we first insert the Player entity using **insertPlayer** method. ***It has to be done in order to fetch Player as we work on in-memory DB***. Then in showPlayer method, we find the inserted player entity by Session.get() and print the Player.