# Hibernate - Mapping Files

An Object/relational mappings are usually defined in an XML document. This mapping file instructs Hibernate — how to map the defined class or classes to the database tables?

Though many Hibernate users choose to write the XML by hand, but a number of tools exist to generate the mapping document. These include **XDoclet, Middlegen** and **AndroMDA** for the advanced Hibernate users.

Let us consider our previously defined POJO class whose objects will persist in the table defined in next section.

```
public class Employee {
   private int id;
   private String firstName;
   private String lastName;
   private int salary;

   public Employee() {}

   public Employee(String fname, String lname, int salary) {
      this.firstName = fname;
      this.lastName = lname;
      this.salary = salary;
   }

   public int getId() {
      return id;
   }

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

   public String getFirstName() {
      return firstName;
   }

   public void setFirstName( String first_name ) {
      this.firstName = first_name;
   }

   public String getLastName() {
      return lastName;
   }

   public void setLastName( String last_name ) {
      this.lastName = last_name;
   }

   public int getSalary() {
      return salary;
   }

   public void setSalary( int salary ) {
      this.salary = salary;
   }
}
```

There would be one table corresponding to each object you are willing to provide persistence. Consider above objects need to be stored and retrieved into the following RDBMS table −

```
create table EMPLOYEE (
   id INT NOT NULL auto_increment,
   first_name VARCHAR(20) default NULL,
   last_name  VARCHAR(20) default NULL,
   salary     INT  default NULL,
   PRIMARY KEY (id)
);
```

Based on the two above entities, we can define following mapping file, which instructs Hibernate how to map the defined class or classes to the database tables.

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name = "Employee" table = "EMPLOYEE">

    <meta attribute = "class-description">
      This class contains the employee detail.
    </meta>

    <id name = "id" type = "int" column = "id">
      <generator class="native"/>
    </id>

    <property name = "firstName" column = "first_name" type = "string"/>
    <property name = "lastName" column = "last_name" type = "string"/>
    <property name = "salary" column = "salary" type = "int"/>

  </class>
</hibernate-mapping>
```

You should save the mapping document in a file with the format <classname>.hbm.xml. We saved our mapping document in the file Employee.hbm.xml.

Let us see understand a little detail about the mapping elements used in the mapping file −

- The mapping document is an XML document having **<hibernate-mapping>** as the root element, which contains all the **<class>** elements.
- The **<class>** elements are used to define specific mappings from a Java classes to the database tables. The Java class name is specified using the **name** attribute of the class element and the database **table** name is specified using the table attribute.

- The **<meta>** element is optional element and can be used to create the class description.
- The **<id>** element maps the unique ID attribute in class to the primary key of the database table. The **name** attribute of the id element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.
- The **<generator>** element within the id element is used to generate the primary key values automatically. The **class** attribute of the generator element is set to **native** to let hibernate pick up either **identity, sequence**, or **hilo** algorithm to create primary key depending upon the capabilities of the underlying database.
- The **<property>** element is used to map a Java class property to a column in the database table. The **name** attribute of the element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

## Hibernate - Mapping Types

When you prepare a Hibernate mapping document, you find that you map the Java data types into RDBMS data types. The **types** declared and used in the mapping files are not Java data types; they are not SQL database types either. These types are called **Hibernate mapping types**, which can translate from Java to SQL data types and vice versa.

This chapter lists down all the basic, date and time, large object, and various other builtin mapping types.

### Primitive Types

| Mapping type | Java type | ANSI SQL Type |
|---|---|---|
| integer | int or java.lang.Integer | INTEGER |
| long | long or java.lang.Long | BIGINT |
| short | short or java.lang.Short | SMALLINT |

| | | |
|---|---|---|
| float | float or java.lang.Float | FLOAT |
| double | double or java.lang.Double | DOUBLE |
| big_decimal | java.math.BigDecimal | NUMERIC |
| character | java.lang.String | CHAR(1) |
| string | java.lang.String | VARCHAR |
| byte | byte or java.lang.Byte | TINYINT |
| boolean | boolean or java.lang.Boolean | BIT |
| yes/no | boolean or java.lang.Boolean | CHAR(1) ('Y' or 'N') |
| true/false | boolean or java.lang.Boolean | CHAR(1) ('T' or 'F') |

**Date and Time Types**

| **Mapping type** | **Java type** | **ANSI SQL Type** |
|---|---|---|
| date | java.util.Date or java.sql.Date | DATE |
| time | java.util.Date or java.sql.Time | TIME |
| timestamp | java.util.Date or java.sql.Timestamp | TIMESTAMP |
| calendar | java.util.Calendar | TIMESTAMP |
| calendar_date | java.util.Calendar | DATE |

**Binary and Large Object Types**

| **Mapping type** | **Java type** | **ANSI SQL Type** |
|---|---|---|
| binary | byte[] | VARBINARY (or BLOB) |
| text | java.lang.String | CLOB |
| serializable | any Java class that implements java.io.Serializable | VARBINARY (or BLOB) |
| clob | java.sql.Clob | CLOB |
| blob | java.sql.Blob | BLOB |

**JDK-related Types**

| **Mapping type** | **Java type** | **ANSI SQL Type** |
|---|---|---|
| class | java.lang.Class | VARCHAR |
| locale | java.util.Locale | VARCHAR |
| timezone | java.util.TimeZone | VARCHAR |
| currency | java.util.Currency | VARCHAR |