Hibernate
Step by Step Guide

Mudassar Hakim
mudassar.trainer@gmail.com

What we are going to build

- A simple Hibernate application persisting *Person* objects
- The database table, *person*, has the following columns
  - *int id*
  - *string cname*
- We will use Derby, HSQL and MySQL as databases
- We will add data to and retrieve data from the table through Hibernate

mudassar.trainer@gmail.com

Things to do

- Start database server and populate database tables
- Write source files
- Build and run the applications

mudassar.trainer@gmail.com

Steps for Starting the database and populating the tables

mudassar.trainer@gmail.com

Steps to follow

1. Start the database server
2. Create database schema (optional)
3. Create and populate database tables

mudassar.trainer@gmail.com

Step 1: Start the database server

- 3 Different options
  - At the command line
  - Through an IDE
  - As windows service (Not all databases support this)

mudassar.trainer@gmail.com

Steps for Writing Source Files

mudassar.trainer@gmail.com

Steps to follow for Writing files

1. Write domain classes (as POJO classes)
   - *Person.java*
2. Write or generate Hibernate mapping files for the domain classes
   - *Person.hbm.xml*
3. Write Hibernate configuration file (per application)
   - *hibernate.cfg.xml*

mudassar.trainer@gmail.com

Step 1: Write Domain Classes (Person.java)

```
public class Person implements Serializable {

    private int id;
    private String name;

    protected Person() {
    }

    public Person(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

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

    // Continued to next page
```


mudassar.trainer@gmail.com

Step 1: Write Domain Classes (Person.java)

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```


mudassar.trainer@gmail.com

Step 2: Write Mapping Files for Domain Classes (Person.hbm.xml)

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"
<hibernate-mapping>
   <class name="Person" table="person">
      <id name="id" type="int">
         <generator class="increment"/>
      </id>

      <property name="name" column="cname" type="string"/>
   </class>
</hibernate-mapping>
```


mudassar.trainer@gmail.com

Step 3: Write Hibernate configuration file (hibernate.cfg.xml) – MySQL

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration
PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<property
   name="connection.driver_class">com.mysql.jdbc.Driver</property>
<property
   name="connection.url">jdbc:mysql://localhost:3306/test</property>
<property name="connection.username"></property>
<property name="connection.password"></property>
<property name="show_sql">true</property>
<property
   name="dialect">org.hibernate.dialect.MySQLDialect</property>
<mapping resource="Person.hbm.xml" />
</session-factory>
</hibernate-configuration>
```


mudassar.trainer@gmail.com

1. <property name="show_sql">true</property>

   Enable the logging of all the generated SQL statements to the console

2. <property name="format_sql">true</property>

   Format the generated SQL statement to make it more readable, but takes up more screen space.

3. <property name="use_sql_comments">true</property>

   Hibernate will put comments inside all generated SQL statements to hint what's the generated SQL trying to do

mudassar.trainer@gmail.com

Connection Pooling

- Hibernate's own connection pooling algorithm is, however, quite rudimentary. It is intended to help you get started and is not intended for use in a production system, or even for performance testing. You should use a third party pool for best performance and stability. Just replace the hibernate.connection.pool_size property with connection pool specific settings. This will turn off Hibernate's internal pool. For example, you might like to use c3p0.

- C3P0 is an open source JDBC connection pool distributed along with Hibernate in the lib directory. Hibernate will use its org.hibernate.connection.C3P0ConnectionProvider for connection pooling if you set hibernate.c3p0.*

mudassar.trainer@gmail.com

```
<property name="hibernate.c3p0.min_size">5</property>
```

Minimum number of JDBC connections in the pool.Hibernate default: 1

```
<property name="hibernate.c3p0.max_size">20</property>
```

Maximum number of JDBC connections in the pool. Hibernate default: 100

```
<property name="hibernate.c3p0.timeout">300</property>
```

When an idle connection is removed from the pool (in second). Hibernate default: 0, never expire.

```
<property
name="hibernate.c3p0.max_statements">50</property>
```

Number of prepared statements will be cached. Increase performance. Hibernate default: 0 , caching is disable.

mudassar.trainer@gmail.com

Steps for Building and Running the Application

mudassar.trainer@gmail.com

Steps to follow for Building and Running the application

1. Add Hibernate library to the classpath
2. Add the client JDBC client side driver to the classpath
3. Compile and run the application performing database operation

mudassar.trainer@gmail.com

Step 1: Copy Hibernate Library Files to the classpath

- Hibernate library files from Hibernate distribution
  - hibernate3.jar
  - other dependency files

mudassar.trainer@gmail.com

Step 2: Add the database driver to the classpath

- The application as a database client needs to have database-specific database driver
  - *derbyclient.jar* (for Derby)
  - *mysql-connector-java-5.1.6-bin.jar* (for MySql)
  - *hsqldb.jar* (for HSQLDB)

mudassar.trainer@gmail.com

Step 3: Compile and run the application

- As a standalone application or Web application
- In this example, we will run it as a standalone application (that has a main() method)

mudassar.trainer@gmail.com

SessionFactory

- ```
  SessionFactory sessionFactory=new
  Configuration().configure().buildSessionFactory();
  ```

- ```
  SessionFactory sessionFactory = new
  Configuration().configure("/com/myexample/persistence
  /hibernate.cfg.xml").buildSessionFactory();
  ```

  If the cfg file is to be put in a different place

- ```
  SessionFactory sessionFactory = new
  Configuration().addResource("com/myexample/persistenc
  e/Courses1.hbm.xml").buildSessionFactory();
  ```

  If the hbm file is to be put in a different place

mudassar.trainer@gmail.com
  ```

Using Annotations

- In hibernate.cfg.xml file

```
<mapping class="com.myexample.courses.Courses" />
```

- The session is obtained in a different way

```
SessionFactory sessionFactory= new
AnnotationConfiguration().configure().buildSessionFac
tory();
```


mudassar.trainer@gmail.com

```
@Entity
@Table(name="courses")
public class Courses {

private long courseId;
private String courseName;

@Id
@GeneratedValue
@Column(name="COURSE_ID")
public long getCourseId() {
return courseId;
}

//continued
```


mudassar.trainer@gmail.com

```
public void setCourseId(long courseId) {
this.courseId = courseId;
}

@Column(name="COURSE_NAME")
public String getCourseName() {
return courseName;
}
public void setCourseName(String courseName) {
this.courseName = courseName;
}
}
```


mudassar.trainer@gmail.com

Thank you!

Mudassar Hakim
mudassar.trainer@gmail.com