Hibernate MySQL Maven Hello World Example (XML)

Created on: August 2, 2014 | **Last updated on:** July 30, 2016 websystiqueadmin

In this tutorial, we will learn about how to use `Hibernate` to perform CRUD operations on database. We will be using `MySQL database`.

Other interesting posts you may like

- Secure Spring REST API using OAuth2
- AngularJS+Spring Security using Basic Authentication
- Secure Spring REST API using Basic Authentication
- Spring 4 MVC+JPA2+Hibernate Many-to-many Example
- Spring 4 Caching Annotations Tutorial
- Spring 4 Cache Tutorial with EhCache
- Hibernate MySQL Maven Hello World Example (Annotation)
- Spring 4 Email Template Library Example
- Spring 4 Email With Attachment Tutorial
- Spring 4 Email Integration Tutorial
- Spring MVC 4+JMS+ActiveMQ Integration Example
- Spring 4+JMS+ActiveMQ @JmsLister @EnableJms Example
- Spring 4+JMS+ActiveMQ Integration Example
- Spring MVC 4+AngularJS Example
- Spring MVC 4 HelloWorld – XML Example
- Spring MVC 4 HelloWorld – Annotation/JavaConfig Example
- Spring 4 HelloWorld – Annotation/JavaConfig Example
- Spring MVC 4+Hibernate 4+MySQL+Maven integration example
- Spring 4 + Hibernate 4 + MySQL+ Maven Integration example (Annotations+XML)
- Spring Security 4 Hello World Annotation+XML Example
- Spring Batch- MultiResourceItemReader & HibernateItemWriter example
- TestNG Hello World Example
- JAXB2 Helloworld Example

Following technologies being used:

- Hibernate 4.3.6.Final
- MySQL Server 5.6
- Maven 3.1.1
- JDK 1.6
- Eclipse JUNO Service Release 2

Let's begin.

Step 1: Create required Database Table

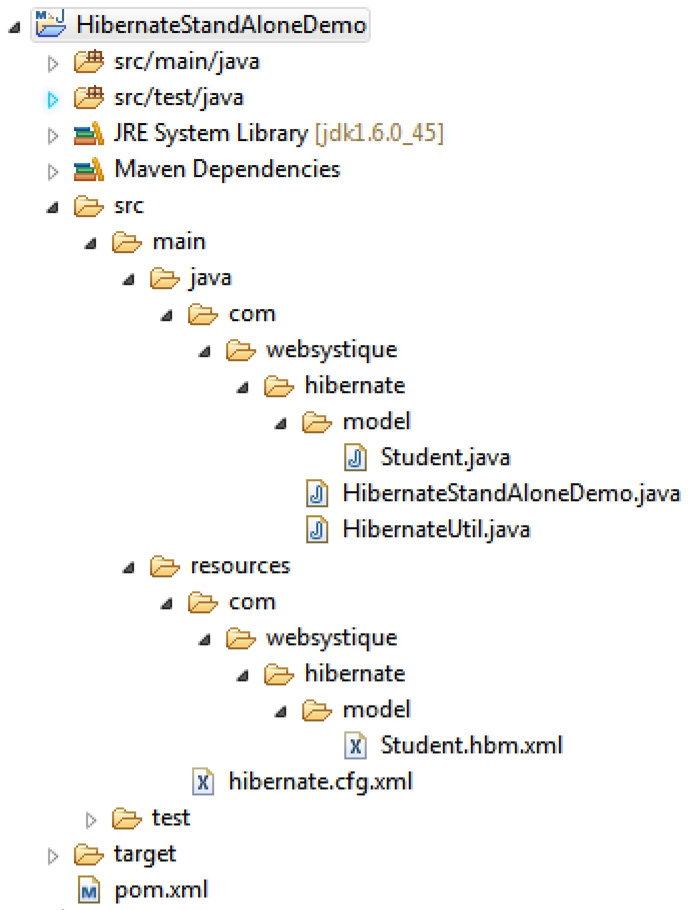
Open MySQL terminal / workbench terminal and execute following MySQL script :

```
create table STUDENT (
  id INT NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(30) NOT NULL,
  last_name  VARCHAR(30) NOT NULL,
  section    VARCHAR(30) NOT NULL,
  PRIMARY KEY (id)
);
```

Please visit MySQL installation on Local PC in case you are finding difficulties in setting up MySQL locally.

Step 2: Create project directory structure

Post Creating a maven project with command line contains step-by-step instruction to create a maven project with command line and importing to eclipse.
Following will be the final project structure:



```
HibernateStandAloneDemo
  src/main/java
  src/test/java
  JRE System Library [jdk1.6.0_45]
  Maven Dependencies
  src
    main
      java
        com
          websystique
            hibernate
              model
                Student.java
              HibernateStandAloneDemo.java
              HibernateUtil.java
      resources
        com
          websystique
            hibernate
              model
                Student.hbm.xml
        hibernate.cfg.xml
    test
  target
  pom.xml
```

Now let’s add/update the content mentioned in above project structure.

Step 3: Update pom.xml to include required Hibernate and MySQL dependency

Following is the updated minimalistic `pom.xml`

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>com.websystique.hibernate</groupId>
	<artifactId>HibernateStandAloneDemo</artifactId>
	<version>1.0.0</version>
	<packaging>jar</packaging>

	<name>HibernateStandAloneDemo</name>

	<properties>
		<hibernate.version>4.3.6.Final</hibernate.version>
		<mysql.connector.version>5.1.31</mysql.connector.version>
	</properties>

	<dependencies>
		<!-- Hibernate -->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>${hibernate.version}</version>
		</dependency>

		<!-- MySQL -->
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>${mysql.connector.version}</version>
		</dependency>
	</dependencies>
	<build>
		<pluginManagement>
			<plugins>
				<plugin>
					<groupId>org.apache.maven.plugins</groupId>
					<artifactId>maven-compiler-plugin</artifactId>
					<version>3.2</version>
					<configuration>
						<source>1.6</source>
						<target>1.6</target>
```

```
                                        </configuration>
                                </plugin>
                        </plugins>
                </pluginManagement>
        </build>

</project>
```

On saving pom.xml with above content, Eclipse will download all the dependencies.

Step 4: Create Model class

Model class **Student** is a simple POJO class which we will be mapping to a database table(created is step 1).

```
package com.websystique.hibernate.model;

import java.io.Serializable;

@SuppressWarnings("serial")
public class Student implements Serializable {

        private int id;
        private String firstName;
        private String lastName;
        private String section;

        public int getId() {
                return id;
        }

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

        public String getFirstName() {
                return firstName;
        }

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

        public String getLastName() {
                return lastName;
        }
```

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

	public String getSection() {
		return section;
	}

	public void setSection(String section) {
		this.section = section;
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + id;
		return result;
	}

	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (!(obj instanceof Student))
			return false;
		Student other = (Student) obj;
		if (id != other.id)
			return false;
		return true;
	}

	@Override
	public String toString() {
		return "Student [id=" + id + ", firstName=" + firstName + ", lastName="
				+ lastName + ", section=" + section + "]";
	}

}
```

It's a good practice to override `hashcode and equals method`. It becomes even mandatory when working with collections or detached instances. Hibernate manual contains a detailed description about them.

Step 5: Create Mapping file

To instruct hibernate to save instances of our Model class, we need to help hibernate about how classes and fields should be mapped to Table and Columns in database. To do that, we will create a mapping file `Student.hbm.xml`. Since it's not a java file, we can create it in `src/main/resources` folder as shown in project structure diagram above.

```
<?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="com.websystique.hibernate.model.Student" table="STUDENT">
                <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="section" column="SECTION" type="string" />
        </class>
</hibernate-mapping>
```

Above file tells hibernate that class Student should be mapped to table STUDENT in database.Properties firstName,lastName & section of class Student will be mapped to columns FIRST_NAME, LAST_NAME & SECTION in database respectively.Id tag maps id property of Student class to primary key of table and with native generator instruct hibernate to choose any of sequeance, identity or hilo algorithm to generate the primary key.

Step 6: Create Hibernate configuration file

We need to inform hibernate about how to connect to database, which database dialect we will be using so that hibernate can generate the instruction specific to that database and what all mapping files we would like hibernate to keep an eye upon.

We define all these information in `hibernate.cfg.xml`. Create this file with below content and save it in `src/main/resources` folder.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM "http://www.hibernate.org/dtd/hibernate-configuration-
3.0.dtd">

<hibernate-configuration>
   <session-factory>
      <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
      <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
      <property name="hibernate.connection.username">myuser</property>
      <property name="hibernate.connection.password">mypassword</property>
```

```
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/websystique</property>
        <property name="show_sql">true</property>
        <property name="format_sql">false</property>
        <mapping resource="com/websystique/hibernate/model/Student.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

`dialect` property informs hibernate to generate database specific (MySQL here) instructions. `driver_class`defines the database specific driver hibernate will use to make connection. Next 3 properties corresponds to database connection.`show_sql` will instruct hibernate to log all the statements on console and `format_sql`instructs it to display properly formatted sql. `mapping` tag instructs hibernate to perform mapping for all the classes defined in file referred by resource attribute

Step 7: Create Hibernate Utility class

This class is well-known in hibernate community, and used for configuring hibernate on startup and managing session factory.

```
package com.websystique.hibernate;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

@SuppressWarnings("deprecation")
public class HibernateUtil {

	private static final SessionFactory sessionFactory;

	static{
		try{
			sessionFactory = new Configuration().configure().buildSessionFactory();
		}catch (Throwable ex) {
			System.err.println("Session Factory could not be created." + ex);
			throw new ExceptionInInitializerError(ex);
		}
	}

	public static SessionFactory getSessionFactory() {
		return sessionFactory;
	}

}
```

Step 7: Create executable class to Run and perform CRUD operations on Database

Put simply, Hibernate operations follow a pattern like:

```
Open session -> begin transaction-> do something with entities -> commit transaction
```

We will be using same pattern in our run class. Create a main class with below mentioned content and run it.

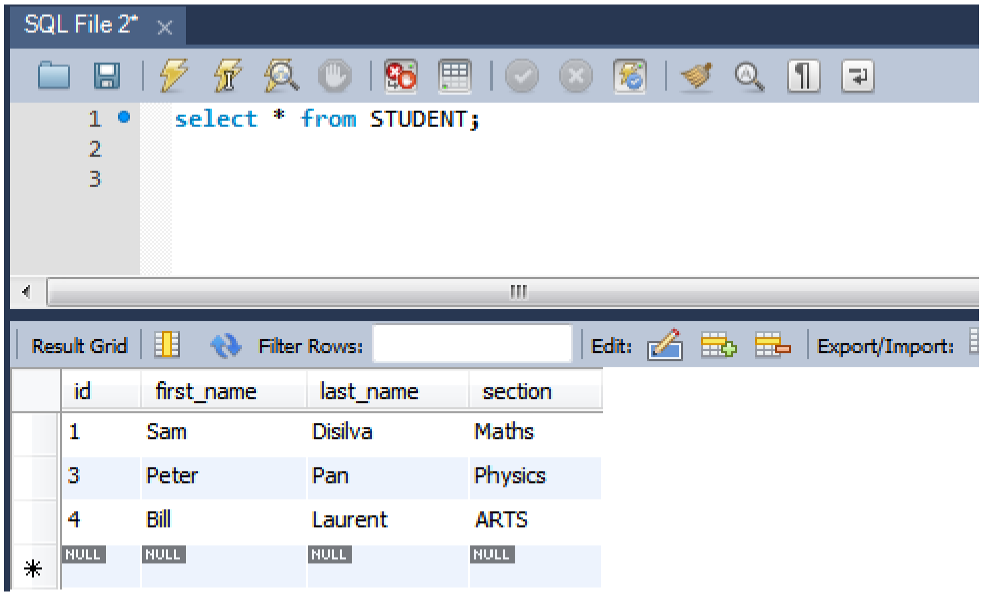
This class will save few records, list all of the persisted records from database, update a specific record in database and finally delete a specific record in database.

```
package com.websystique.hibernate;

import java.util.List;

import org.hibernate.Session;

import com.websystique.hibernate.model.Student;

/**
 * Class used to perform CRUD operation on database with Hibernate API's
 *
 */
public class HibernateStandAloneDemo {

	@SuppressWarnings("unused")
	public static void main(String[] args) {

		HibernateStandAloneDemo application = new HibernateStandAloneDemo();

		/*
		 * Save few objects with hibernate
		 */
		int studentId1 = application.saveStudent("Sam", "Disilva", "Maths");
		int studentId2 = application.saveStudent("Joshua", "Brill", "Science");
		int studentId3 = application.saveStudent("Peter", "Pan", "Physics");
		int studentId4 = application.saveStudent("Bill", "Laurent", "Maths");

		/*
		 * Retrieve all saved objects
		 */
		List<Student> students = application.getAllStudents();
		System.out.println("List of all persisted students >>>");
		for (Student student : students) {
			System.out.println("Persisted Student :" + student);
		}
		/*
		 * Update an object
		 */
		application.updateStudent(studentId4, "ARTS");

		/*
		 * Deletes an object
		 */
		application.deleteStudent(studentId2);

		/*
		 * Retrieve all saved objects
		 */
		List<Student> remaingStudents = application.getAllStudents();
		System.out.println("List of all remained persisted students >>>");
		for (Student student : remaingStudents) {
			System.out.println("Persisted Student :" + student);
		}

	}

	/**
	 * This method saves a Student object in database
	 */
	public int saveStudent(String firstName, String lastName, String section) {
		Student student = new Student();
		student.setFirstName(firstName);
		student.setLastName(lastName);
		student.setSection(section);

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

		int id = (Integer) session.save(student);
		session.getTransaction().commit();
		return id;
	}

	/**
	 * This method returns list of all persisted Student objects/tuples from
	 * database
	 */
	public List<Student> getAllStudents() {
		Session session = HibernateUtil.getSessionFactory().openSession();
		session.beginTransaction();
		@SuppressWarnings("unchecked")
		List<Student> employees = (List<Student>) session.createQuery(
				"FROM Student s ORDER BY s.firstName ASC").list();

		session.getTransaction().commit();
		return employees;
	}

	/**
	 * This method updates a specific Student object
	 */
	public void updateStudent(int id, String section) {
		Session session = HibernateUtil.getSessionFactory().openSession();
		session.beginTransaction();

		Student student = (Student) session.get(Student.class, id);
		student.setSection(section);
		//session.update(student);//No need to update manually as it will be updated
automatically on transaction close.
		session.getTransaction().commit();
	}

	/**
	 * This method deletes a specific Student object
	 */
	public void deleteStudent(int id) {
		Session session = HibernateUtil.getSessionFactory().openSession();
		session.beginTransaction();

		Student student = (Student) session.get(Student.class, id);
		session.delete(student);
		session.getTransaction().commit();
	}
}
```

Below is the generated output of above program (set `show_sql` to false if you don't want sql to be shown in output)

```
Hibernate: insert into STUDENT (FIRST_NAME, LAST_NAME, SECTION) values (?, ?, ?)
Hibernate: insert into STUDENT (FIRST_NAME, LAST_NAME, SECTION) values (?, ?, ?)
Hibernate: insert into STUDENT (FIRST_NAME, LAST_NAME, SECTION) values (?, ?, ?)
Hibernate: insert into STUDENT (FIRST_NAME, LAST_NAME, SECTION) values (?, ?, ?)
Hibernate: select student0_.id as id1_0_, student0_.FIRST_NAME as FIRST_NA2_0_,
student0_.LAST_NAME as LAST_NAM3_0_, student0_.SECTION as SECTION4_0_ from STUDENT
student0_ order by student0_.FIRST_NAME ASC
List of all persisted students >>>
Persisted Student :Student [id=4, firstName=Bill, lastName=Laurent, section=Maths]
Persisted Student :Student [id=2, firstName=Joshua, lastName=Brill, section=Science]
Persisted Student :Student [id=3, firstName=Peter, lastName=Pan, section=Physics]
Persisted Student :Student [id=1, firstName=Sam, lastName=Disilva, section=Maths]
Hibernate: select student0_.id as id1_0_0_, student0_.FIRST_NAME as FIRST_NA2_0_0_,
student0_.LAST_NAME as LAST_NAM3_0_0_, student0_.SECTION as SECTION4_0_0_ from STUDENT
student0_ where student0_.id=?
Hibernate: update STUDENT set FIRST_NAME=?, LAST_NAME=?, SECTION=? where id=?
Hibernate: select student0_.id as id1_0_0_, student0_.FIRST_NAME as FIRST_NA2_0_0_,
student0_.LAST_NAME as LAST_NAM3_0_0_, student0_.SECTION as SECTION4_0_0_ from STUDENT
student0_ where student0_.id=?
Hibernate: delete from STUDENT where id=?
Hibernate: select student0_.id as id1_0_, student0_.FIRST_NAME as FIRST_NA2_0_,
student0_.LAST_NAME as LAST_NAM3_0_, student0_.SECTION as SECTION4_0_ from STUDENT
student0_ order by student0_.FIRST_NAME ASC
List of all remained persisted students >>>
Persisted Student :Student [id=4, firstName=Bill, lastName=Laurent, section=ARTS]
Persisted Student :Student [id=3, firstName=Peter, lastName=Pan, section=Physics]
Persisted Student :Student [id=1, firstName=Sam, lastName=Disilva, section=Maths]
```

Below is the snapshot of MySQL database after execution of above program.



```sql
select * from STUDENT;
```

id	first_name	last_name	section
1	Sam	Disilva	Maths
3	Peter	Pan	Physics
4	Bill	Laurent	ARTS
NULL	NULL	NULL	NULL

That's it.

Download Source Code

Download Now!

References

- Hibernate Home
- Hibernate Entity Detailed Description
- HashCode & Equals in Hibernate Entity