Tutorial 1: Making SQL queries in Java using Hibernate and Spring

Author: Diana ALLAM

In this tutorial, you will learn how to use the Hibernate framework for mapping an object-oriented domain model to a relational database.
For this lab you have to install:
- at least Java 6
- Eclipse J2EE (Neon or other)
- Maven
- MySQL (server and Workbench)

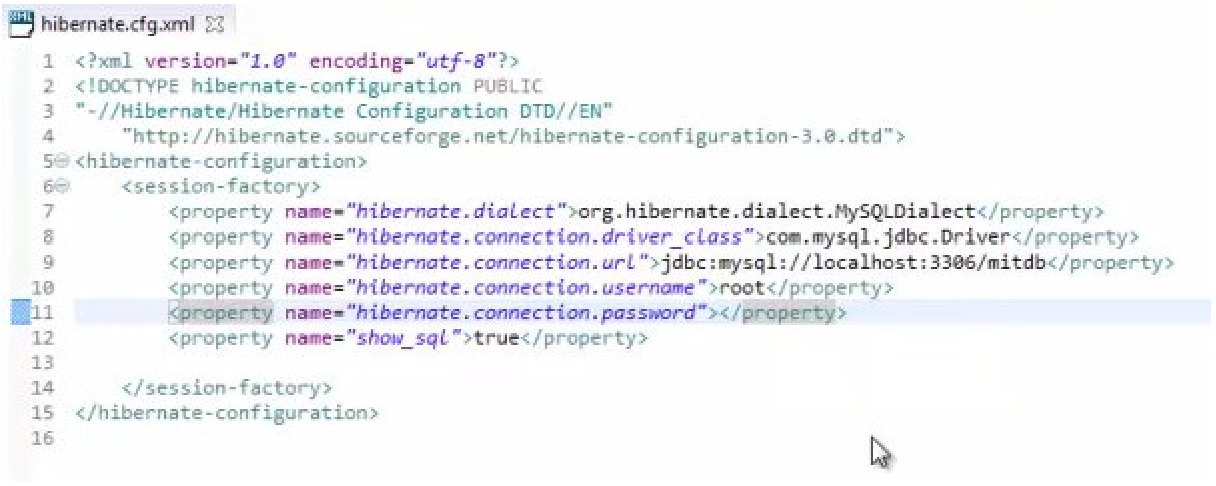
Step 1) Create a table called Users in your database which contains:
- user_id: it is the primary Key, it is an integer, it souldn't be NULL and it should be autoincremented
- user_name: it is a varchar
- email: it is a varchar

Step 2) Add Hibernate plugin to your eclipse IDE
https://marketplace.eclipse.org/content/hibernate-search-plugin
(Drag and drop the Install button in your running Eclipse IDE)

Step 3) Create a dynamic Web project as a maven project
In your Eclipse IDE, create a Maven project, (choose the webapp archetype in order to have a dynamic Web project in your maven project)

Step 4) Create a Hibernate configuration file in your maven project
- Right click on the src directory of your project
- New > Other
- Choose Hibernate Configuration File (cfg.xml)
- A wizard appears with the preselected src directory in your project, Click Next
- Click on Get Values from connection > New > MySQL
- Give a Name to the connection to create for MySQL Data base > Next
- You have to fill then the connection details with the name of your data base, URL, User name and password
- Click on "Test connection" button to test if all is going right
- Next > Finish
- You have now finished the creation of a connection profile to your MySQL data base, Click Ok
- You should complete other informations for the Hibernate Configuration File > Choose MySQL for the Database dialect
- Check the box "create a console configuration" > Next
- In the Option tab choose MySQL for "Database dialect" > Finish
- Congrats you finished the creation of your hibernate.cfg.xml file

Your file should be similar to this:

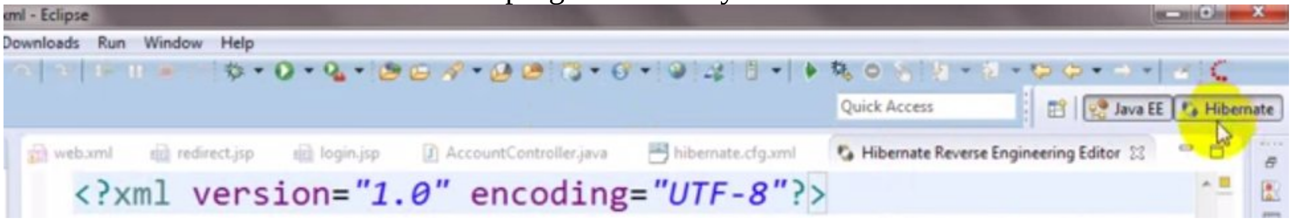


```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mitdb</property>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password"></property>
        <property name="show_sql">true</property>

    </session-factory>
</hibernate-configuration>

```

Step 5) Create a mapping between your java project and your MySQL database using Hibernate and Spring

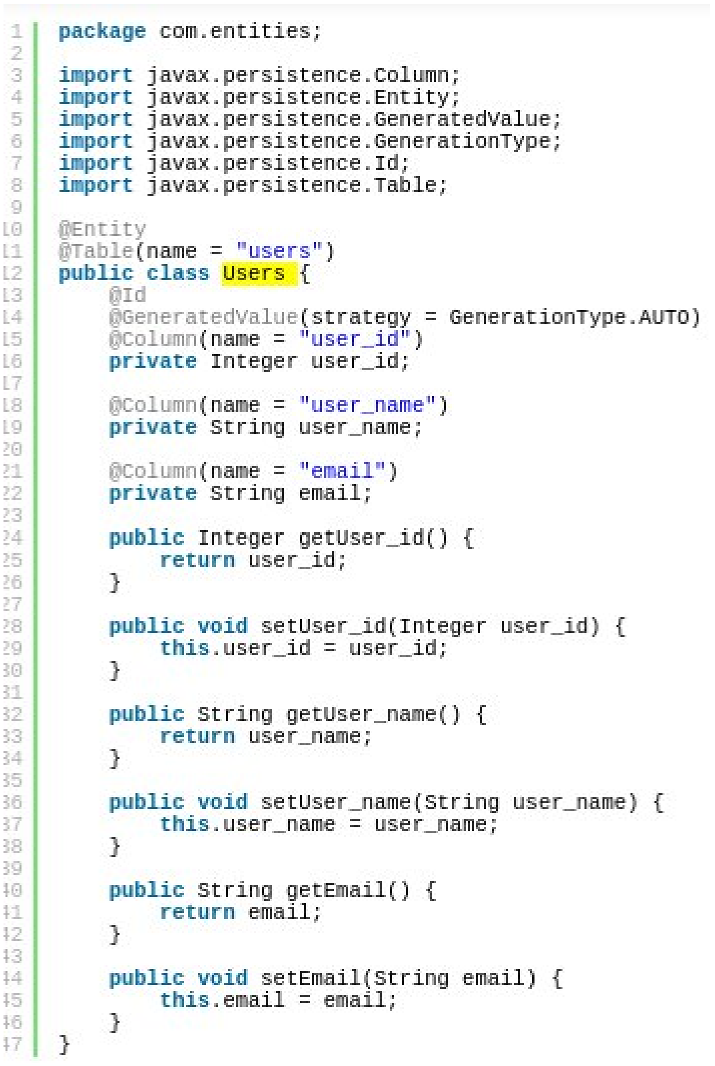
The idea here is to auto-generate the java class Users which corresponds to the table Users using Hibernate. To do it:

- Create a com.entities package in your src directory
- Click on the Hibernate button at the up right corner in your IDE to switch to the Hibernate View



```
<?xml version="1.0" encoding="UTF-8"?>
```

- Click on the options of the run button to select "Hibernate Code Generation Configuration"
- In the Main tab:
-- Choose your project for "Console configuration"
-- Choose the src file of your project as Output directory
-- Check "Reverse engineer from JDBC Connection"
-- Specify the package "com.entities"
- In the Exporters tab:
-- Check "Use Java 5 Syntax", "Generate EJB3 Annotations" and "Domain code (.java)"
- Click Run
- Come back to the Java EE view
- Check that you have "Users" class created in your "com.entities" package, it should be like follows:

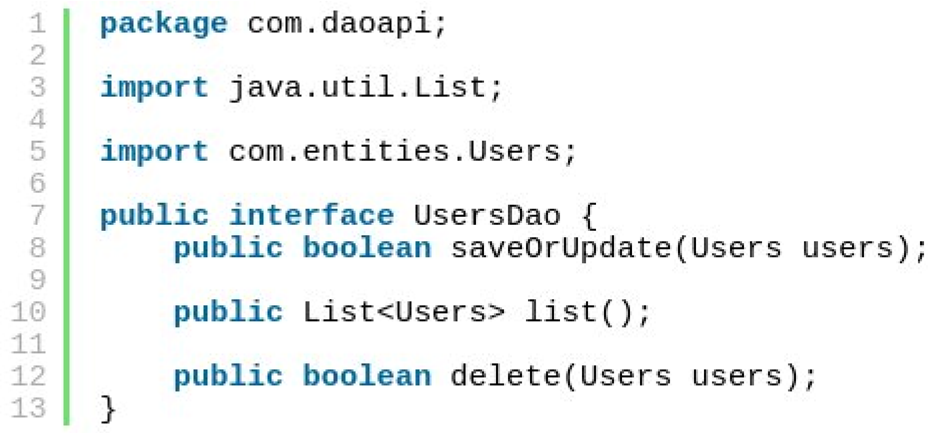


```java
package com.entities;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "users")
public class Users {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "user_id")
    private Integer user_id;

    @Column(name = "user_name")
    private String user_name;

    @Column(name = "email")
    private String email;

    public Integer getUser_id() {
        return user_id;
    }

    public void setUser_id(Integer user_id) {
        this.user_id = user_id;
    }

    public String getUser_name() {
        return user_name;
    }

    public void setUser_name(String user_name) {
        this.user_name = user_name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

The generated annotations: @Entity, @Table, @Id, @GeneratedValue, @Column are specific to Hibernate Library to make the mapping between the Java Class and the MySql table.
In order to inport the hibernate Library with maven, you have to add the following dependencies to your pom.xml file. Maven will automatically import the corresponding jars form the local or remote repository:

```xml
    <dependency>
        <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>1.4</version>
     </dependency>

     <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.36</version>
     </dependency>
         <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>4.1.0.Final</version>
     </dependency>
        <dependency>
        <groupId>org.hibernate.javax.persistence</groupId>
        <artifactId>hibernate-jpa-2.0-api</artifactId>
        <version>1.0.0.Final</version>
     </dependency>
```

Step 6) To create the DAO (Data Access Object) level
The DAO level will allow us to make SQL queries from the java code using Hibernate. To build this level, you should:
- Create a package come.daoapi where you will create the dao interface UsersDao which contains three methods:
-- delete a user from the table
-- add or update a user in the table
-- get the list of users in the tabe



```java
package com.daoapi;

import java.util.List;

import com.entities.Users;

public interface UsersDao {
    public boolean saveOrUpdate(Users users);

    public List<Users> list();

    public boolean delete(Users users);
}
```

- Create a package com.daoimpl where you will create the implementation class of the interface UsersDao.
You should create a mapping bean responsible for interactions with the Users table. To create this bean we will use the annotations from the Spring framework (Available since version 4.2.5).

-- First as we will use Spring framework, you should add the adequate dependencies in the pom.xml file:

```xml
        <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>4.2.6.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.2.6.RELEASE</version>
</dependency>

<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>4.2.6.RELEASE</version>
</dependency>
```

```
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>4.2.6.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>4.2.6.RELEASE</version>
        </dependency>
```

-- Add the Spring Annotation @Repository at the biginning of your class in order to create the mapping bean (you should import org.springframework.stereotype.Repository)
-- Add the Spring Annotation @Transactional in order to allow transactional operation on access to the data base (you should import org.springframework.transaction.annotation.Transactional)
-- In order to create a session on the data base using Hibernate and send your SQL queries to MySQL database, Spring do it for you by using a SessionFactory bean. Spring creates an instance of this bean at runtime. In order to get this instance in your code and to use it to make your queries on MySQL database, you should use the @Autowired annotation.
The SessionFactory bean should be declared in the "applicationContext.xml" file.
This file should be under src/main/webapp/WEB-INF.

The applicationContext.xml should contain three beans as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-3.0.xsd
http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">

    <!-- Enable autowire -->
    <context:annotation-config />
    <context:component-scan base-package="com" />

    <mvc:annotation-driven />

    <mvc:resources mapping="/resources/**" location="/resources/" />

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
```

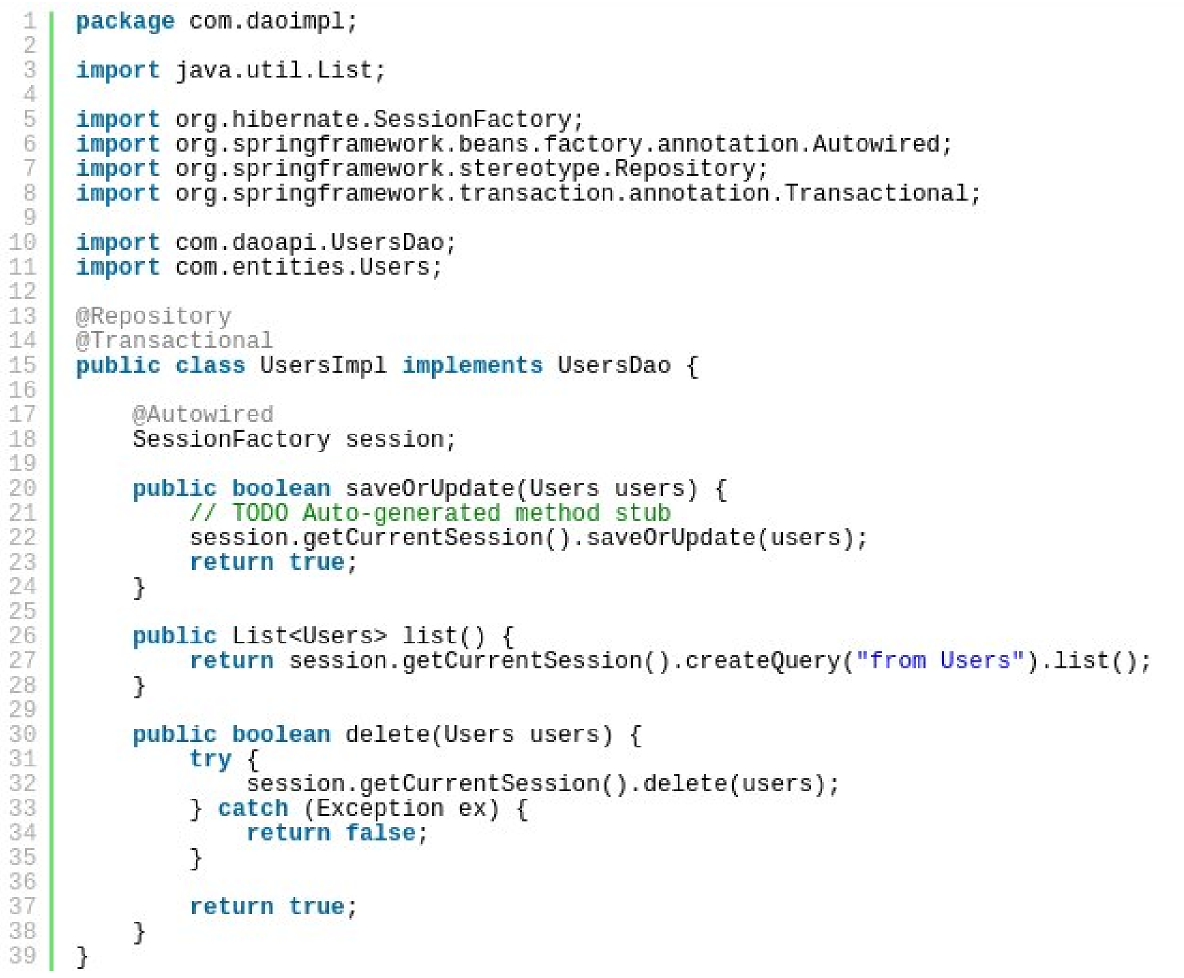
```xml
        <property name="url" value="jdbc:mysql://localhost:3306/tests" />
        <property name="username" value="root" />
        <property name="password" value="12345" />
    </bean>

    <!-- Session Factory Declaration -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="packagesToScan" value="com.entities" />
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
                <prop key="hibernate.show_sql">true</prop>
                <prop key="hibernate.enable_lazy_load_no_trans">true</prop>
                <prop key="hibernate.default_schema">test</prop>
                <prop key="format_sql">true</prop>
                <prop key="use_sql_comments">true</prop>
                <!-- <prop key="hibernate.hbm2ddl.auto">create</prop> -->
            </props>
        </property>
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager" />

    <bean id="transactionManager"
        class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>
</beans>
```

As applicationContext.xml file contains the configuration we used in the "hibernate.cfg.xml" file, you can now remove this one. Indeed, it was useful for Java code generation to create the classes mapped to the corresponding tables.

-- Come back now to your UsersImpl class which should implements UsersDao interface.
This class should implement the queries to Create/Read/Update/Delete a user from the data base using the SessionFactory methods (take a look to the SessionFactory documentation from Spring API at the following URL:

https://docs.spring.io/spring/docs/2.5.6/javadoc/api/org/springframework/orm/toplink/SessionFactory.html)

To get an instance of the sessionFactory created by Spring, you should declare SessionFactory attribute using @Autowired annotation.
The code of your UsersImpl Class should be as follows:



```java
package com.daoimpl;

import java.util.List;

import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.daoapi.UsersDao;
import com.entities.Users;

@Repository
@Transactional
public class UsersImpl implements UsersDao {

    @Autowired
    SessionFactory session;

    public boolean saveOrUpdate(Users users) {
        // TODO Auto-generated method stub
        session.getCurrentSession().saveOrUpdate(users);
        return true;
    }

    public List<Users> list() {
        return session.getCurrentSession().createQuery("from Users").list();
    }

    public boolean delete(Users users) {
        try {
            session.getCurrentSession().delete(users);
        } catch (Exception ex) {
            return false;
        }

        return true;
    }
}
```

Step 8) Make a test with Junit

- Add the junit dependency in pom.xml file:

```xml
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
     </dependency>
```

- Right click on src/test/java directory > New > Project > JUnit
- Choose JUNit 4, a name for package and a name for your test class
- Choose UsersImpl as a class under test > Next > select the methods you want to test (saveOrUpdate) > finish
- In the generated code in your test class, you have:
-- a setUp method preceeded with @Before annotation: you should put here the preconditions of your test
-- a tearDown method preceeded with @After annotation: you should put here the postconditions of your test
-- a testSaveOrUpdate method preceeded with @Test annotation: you should write here your test.

- in the test preconditions, you should get an instance of the UsersDao bean created by the Spring context.

To do that, you have to:
-- Change the annotation @Repository in UsersImpl class to @Respository("UsersDao")
-- Add two attributes to your test class:

```
private static ClassPathXmlApplicationContext context
private static UsersDao usersDao
```

- In the setUp method of your JUnit test, to get the instance of UsersImpl, you should write:

```
context = new  ClassPathXmlApplicationContext("application-context.xml")
usersDao = (UsersDao) context.getBean("UsersDao");
```

- In the tearDown method, you should close the context: context.close();

- In testSaveOrUpdate method, the idea is to create an instance of Users, then we want to add the created user in the data base using the SaveOrUpdate methode implemented in UsersImpl, then we should verify in MySQL workbench that the user appear in the Users table.
You have so to write the following code:

```
Users newUser = new Users();
newUser.setUser_name("ALLAM");
newUser.setEmail("allam@gmail.com");
usersDao.saveOrUpdate(newUser);
```

- To test your code, Right click on your test class > Run as > JUnit test
- Check your Users table to verify that you have a new added line.
- **To do:** modify your testSaveOrUpdate method in order to check in Java that the user is well added in the dataBase.