Unit Test JPA with JUnit H2 In Memory Database

BY MEMORYNOTFOUND · NOVEMBER 15, 2016

This tutorial shows how to **Unit Test JPA** with **JUnit** and a **H2 In Memory Database**. Some people disagree and don't call these unit tests, but integration tests. Imho there is some truth in both. The following is not a pure unit test and neither is it a pure integration test. 'Cause it doesn't use an identical copy of the production database. That a side, let's look at the example.

Maven Dependencies

We use Apache Maven to manage the projects dependencies.

```
<?xml version="1.0" encoding="UTF-8"?>
<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.memorynotfound.db.hibernate.testing</groupId>
    <artifactId>unit-test-jpa-junit-h2</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>war</packaging>
    <name>HIBERNATE - ${project.artifactId}</name>
    <url>http://memorynotfound.com</url>

    <properties>
        <mysql.driver.version>6.0.5</mysql.driver.version>
        <hibernate.version>5.2.4.Final</hibernate.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${mysql.driver.version}</version>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>${hibernate.version}</version>
        </dependency>

        <!-- testing -->
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <version>1.4.192</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.6.0</version>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>2.19</version>
```

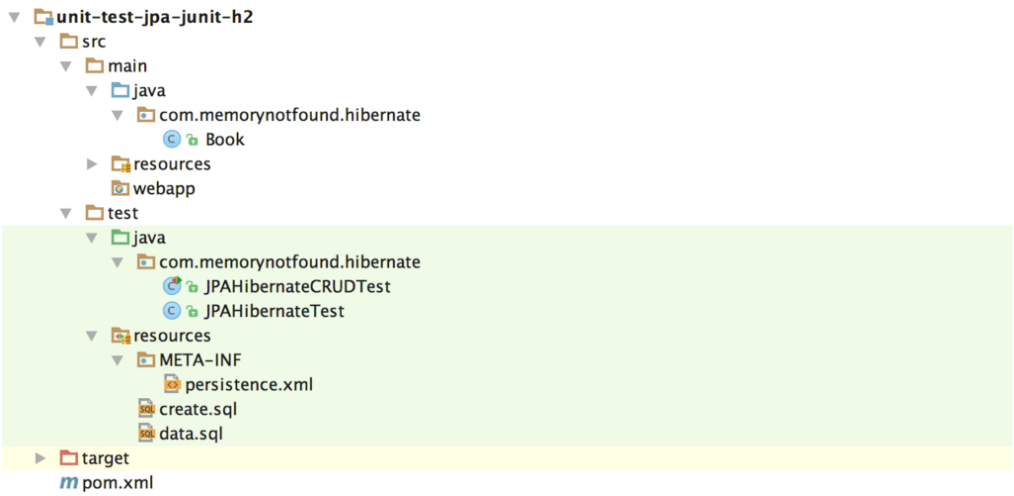
```
            </plugin>
        </plugins>
    </build>

</project>
```

Make sure the `com.h2database:h2` dependency resides on the classpath. This dependency is responsible for creating a pure java in memory database. This results in fast and reliable tests.

Project Structure

Your project structure must look simular to the following.



Model Class + JPA Mappings

For simplicity, we'll work with a single entity. Here is the class, annotated using Java Persistence API Annotations.

```
package com.memorynotfound.hibernate;

import javax.persistence.*;

@Entity
@NamedQueries(value = {
    @NamedQuery(name = "Book.getAll", query = "SELECT b FROM Book b")
})
public class Book {

    @Id
    private Integer id;
    private String title;

    public Book() {
    }

    public Book(Integer id, String title) {
        this.id = id;
        this.title = title;
    }

    public Integer getId() {
        return id;
    }

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

    public String getTitle() {
        return title;
```

```
    }

    public void setTitle(String title) {
        this.title = title;
    }

    @Override
    public String toString() {
        return "Book{" +
                "id=" + id +
                ", title='" + title + '\'' +
                '}';
    }
}
```

Configure JPA + In Memory H2 Database

We create a JPA Persistence Unit called **mnf-pu-test**. This persistence unit is specifically for testing and is located in the `src/test/resources/META-INF/persistence.xml` file. **Note**: this file is located under the *test* folder. This allows us to clearly separate the project with the test environment.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistenc
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
                                 http://xmlns.jcp.org/xml/ns/persistence
    <persistence-unit name="mnf-pu-test" transaction-type="RESOURCE_LOCA

        <!-- add classes -->
        <class>com.memorynotfound.hibernate.Book</class>

        <properties>
            <!-- Configuring JDBC properties -->
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:m
            <property name="javax.persistence.jdbc.driver" value="org.h2

            <!-- Hibernate properties -->
            <property name="hibernate.dialect" value="org.hibernate.dial
            <property name="hibernate.hbm2ddl.auto" value="validate"/>
            <property name="hibernate.format_sql" value="false"/>
            <property name="hibernate.show_sql" value="true"/>

        </properties>
    </persistence-unit>
</persistence>
```

Since we are using an In-Memory H2 database, we need some slightly different connection properties than a traditional MySQL, MariaDB, Postgrsql or other database vendor. Make sure you use the correct `dialect` e.g.: **org.hibernate.dialect.H2Dialect**.

jdbc:h2:mem:<database_name>; creates an in-memory database with a given database name. We can optionally initialize the In Memory H2 Database on application startup. We can pass scripts using the `INIT=RUNSCRIPT FROM '<path>'` in the connection string.

Database Initialization Scripts

The database scripts are located on the classpath in the `src/test/resources` folder. The create database tables script is executed on application start.

```
CREATE TABLE `Book` (
  `id` int(11) NOT NULL,
  `title` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
)
```

The data script is executed on application start.

```
DELETE FROM Book;
INSERT INTO Book(id, title) VALUES (1, 'Unit Test Hibernate/JPA with in
```

Abstract JPA Hibernate Test Case

We wrote a simple class, which manages the `EntityManager`. This class initializes the `EntityManager` before the JUnit tests are executed and closes it after the tests are executed. This class also contains a method that resets the database before a method is invoked. This leads to consistent tests across all unit tests. Meaning, tests cannot have an influence on other tests.

```
package com.memorynotfound.hibernate;

import org.h2.tools.RunScript;
import org.hibernate.Session;
import org.hibernate.jdbc.Work;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.sql.Connection;
import java.sql.SQLException;

public class JPAHibernateTest {

    protected static EntityManagerFactory emf;
    protected static EntityManager em;

    @BeforeClass
    public static void init() throws FileNotFoundException, SQLException
        emf = Persistence.createEntityManagerFactory("mnf-pu-test");
        em = emf.createEntityManager();
    }

    @Before
    public void initializeDatabase(){
        Session session = em.unwrap(Session.class);
        session.doWork(new Work() {
            @Override
            public void execute(Connection connection) throws SQLExcepti
                try {
                    File script = new File(getClass().getResource("/data
                    RunScript.execute(connection, new FileReader(script)
                } catch (FileNotFoundException e) {
                    throw new RuntimeException("could not initialize wit
                }
            }
        });
    }

    @AfterClass
    public static void tearDown(){
```

```
        em.clear();
        em.close();
        emf.close();
    }
}
```

Unit Test JPA with JUnit

We extend from the `JPAHibernateTest`, that we created earlier. This leads to clean and easy to understand unit tests. We can use the `EntityManager` to retrieve, insert, update or delete entities from the in-memory database.

```
package com.memorynotfound.hibernate;

import org.junit.Test;
import java.util.List;
import static junit.framework.TestCase.assertNotNull;
import static org.junit.Assert.assertEquals;

public class JPAHibernateCRUDTest extends JPAHibernateTest {

    @Test
    public void testGetObjectById_success() {
        Book book = em.find(Book.class, 1);
        assertNotNull(book);
    }

    @Test
    public void testGetAll_success() {
        List<Book> books = em.createNamedQuery("Book.getAll", Book.class
        assertEquals(1, books.size());
    }

    @Test
    public void testPersist_success() {
        em.getTransaction().begin();
        em.persist(new Book(10, "Unit Test Hibernate/JPA with in memory
        em.getTransaction().commit();

        List<Book> books = em.createNamedQuery("Book.getAll", Book.class

        assertNotNull(books);
        assertEquals(2, books.size());
    }

    @Test
    public void testDelete_success(){
        Book book = em.find(Book.class, 1);

        em.getTransaction().begin();
        em.remove(book);
        em.getTransaction().commit();

        List<Book> books = em.createNamedQuery("Book.getAll", Book.class

        assertEquals(0, books.size());
    }

}
```

References

- H2 In Memory Database Doc
- H2 JavaDoc API
- JUnit JavaDoc API