ABOUT VINOD KUMAR KASHYAP

Vinod is Sun Certified and love to work in Java and related technologies. Having more than 10 years of experience, he had developed software's including technologies like Java, Hibernate, Struts, Spring, HTML 5, jQuery, CSS, Web Services, MongoDB, AngularJS. He is also a JUG Leader of Chandigarh Java User Group.

JUnit HSQLDB Example

Posted by: Vinod Kumar Kashyap in junit March 27th, 2017

Here is the new JUnit example, but with a difference. In this example we shall show users how they can use JUnit with HSQLDB for testing. In JUnit HSQLDB example, we will try to explain the usage of HSQLDB. Why we are using HSQLDB and not any other DB?

We will try to explain the answer to this question in this example. Let's start by a little introduction of the HSqlDB.

1. Introduction

HSQLDB is a 100% Java database. HSQLDB (HyperSQL DataBase) is the leading SQL relational database software written in Java. Latest version 2.3.4 is fully multi-threaded and supports high performance 2PL and MVCC (multi version concurrency control) transaction control models.

We can use this database as a in memory database also. This answers our question that why we are using HSQLDB for our example. We will create in memory database, create a table, insert data into tables and after test cases are executed we will drop table. So all in all we will use database that will work in memory. We will not start any server to run DB nor we stop it.

2. Technologies Used

We will use the following technologies in this example.

- **Maven**: Build and Dependency Tool
- **HSQLDB**: In Memory 100% Java database
- **Eclipse**: IDE for coding

3. Project Setup

Tip
You may skip project creation and jump directly to the **beginning of the example** below.
We will start by creating a Maven project. Open Eclipse. Select

```
File -> New -> Maven Project
```

. Fill in the details and click on the **Next** button.

Figure 1: JUnit HSqlDB Example Setup 1

On this screen, fill in the details as mentioned below and click on **Finish** button.

Figure 2: JUnit HSqlDB Example Setup 2

With this, we are ready with the blank Maven project. Let’s start filling up the details.

4. JUnit HSQLDB Example

Starting by writing the below line in the

```
pom.xml
```

file. This will fetch all dependencies for our example to work.

pom.xml

```
01      <dependencies>
02          <dependency>
03              <groupId>junit</groupId>
04              <artifactId>junit</artifactId>
05              <version>4.12</version>
06          </dependency>
07
08          <dependency>
09              <groupId>org.hsqldb</groupId>
10              <artifactId>hsqldb</artifactId>
11              <version>2.3.4</version>
12          </dependency>
13      </dependencies>
14
15      <properties>
16          <maven.compiler.source>1.8</maven.compiler.source>
17          <maven.compiler.target>1.8</maven.compiler.target>
18      </properties>
```

Now this will fetch

```
JUnit jar
```

(**line 3**),

```
HSLDB jar
```

(**line 9**) and also tell maven to use Java 1.8 for compiling of this example(**line 16,17**).

JUnitHSqlDBTest

```java
package junithsqldb;

import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertThat;

import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class JUnitHSqlDBTest {

    @BeforeClass
    public static void init() throws SQLException, ClassNotFoundException, IOException {
        Class.forName("org.hsqldb.jdbc.JDBCDriver");

        // initialize database
        initDatabase();
    }


    @AfterClass
    public static void destroy() throws SQLException, ClassNotFoundException, IOException {
        try (Connection connection = getConnection(); Statement statement = connection.createState
            statement.executeUpdate("DROP TABLE employee");
            connection.commit();
        }
    }

    /**
     * Database initialization for testing i.e.
     * <ul>
     * <li>Creating Table</li>
     * <li>Inserting record</li>
     * </ul>
     *
     * @throws SQLException
     */
    private static void initDatabase() throws SQLException {
        try (Connection connection = getConnection(); Statement statement = connection.createState
            statement.execute("CREATE TABLE employee (id INT NOT NULL, name VARCHAR(50) NOT NULL,"
                    + "email VARCHAR(50) NOT NULL, PRIMARY KEY (id))");
            connection.commit();
            statement.executeUpdate(
                    "INSERT INTO employee VALUES (1001,'Vinod Kumar Kashyap', 'vinod@javacodegeeks
            statement.executeUpdate("INSERT INTO employee VALUES (1002,'Dhwani Kashyap', 'dhwani@j
            statement.executeUpdate("INSERT INTO employee VALUES (1003,'Asmi Kashyap', 'asmi@javac
            connection.commit();
        }
    }

    /**
     * Create a connection
     *
     * @return connection object
     * @throws SQLException
     */
    private static Connection getConnection() throws SQLException {
        return DriverManager.getConnection("jdbc:hsqldb:mem:employees", "vinod", "vinod");
    }

    /**
     * Get total records in table
     *
     * @return total number of records. In case of exception 0 is returned
     */
    private int getTotalRecords() {
        try (Connection connection = getConnection(); Statement statement = connection.createState
            ResultSet result = statement.executeQuery("SELECT count(*) as total FROM employee");
            if (result.next()) {
                return result.getInt("total");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return 0;
    }

    @Test
    public void getTotalRecordsTest() {
        assertThat(3, is(getTotalRecords()));
    }

    @Test
    public void checkNameExistsTest() {
        try (Connection connection = getConnection();
                Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE
                        ResultSet.CONCUR_READ_ONLY);) {

            ResultSet result = statement.executeQuery("SELECT name FROM employee");

            if (result.last()) {
                assertThat("Asmi Kashyap", is(result.getString("name")));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Now let’s see each step in this class.
Line 19: This method will execute before all

```
@Test
```

cases. It will initialize our DB that will be used to test.
Line 28: This method will execute after all

```
@Test
```

cases are executed. We will drop table in this method
Line 45: Initialize DB with table creation and insertion of records.
Line 64: Creating a connection.
Line 73: Return total number of records in DB.
Line 86:

```
@Test
```

method to test case for total number of records.
Line 91:

```
@Test
```

method to test for fetch records.

Output

Figure 3: JUnit HSqlDB Example Output

5. Conclusion