



[ANDROID](#) |
 [CORE JAVA](#) |
 [DESKTOP JAVA](#) |
 [ENTERPRISE JAVA](#) |
 [JAVA BASICS](#) |
 [JVM LANGUAGES](#) |
 [SOFTWARE DEVELOPMENT](#) |
 [DEVOPS](#)

△ Home » Core Java » junit » JUnit HSQLDB Example

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.

Want to be a JUnit Master ?

Subscribe to our newsletter and download the JUnit Programming Cookbook [right now!](#)

In order to help you master unit testing with JUnit, we have compiled a kick-ass guide with all the major JUnit features and use cases! Besides studying them online you may download the eBook in PDF format!

Email address:

[Sign up](#)

2. Technologies Used

We will use the following technologies in this example.

CARRER OPPORTUNITIES



Get the latest jobs to your ir

[Send me jobs!](#)

[Job Search](#)



- **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.

182,185 insiders are already enjoying weekly updates and complimentary whitepapers!

Join them now to gain **exclusive access** to the latest news in the Java world, as well as insights about Android, Scala, Groovy and other related technologies.

Email address:

Your email address

☒ Receive Java & Developer job alerts in your Area from our partners over at ZipRecruiter

Sign up

JOIN US



With **1,240,600** unique visitors and **500** authors placed among related sites and forums, we are constantly being looked out for and encourage you to be a **guest writer** for Java Code Geek your writing skills!

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),

```
HSQLDB jar
```

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

JUnitHSqlDBTest

```

001 package junithsqldb;
002
003 import static org.hamcrest.CoreMatchers.is;
004 import static org.junit.Assert.assertThat;
005
006 import java.io.IOException;
007 import java.sql.Connection;
008 import java.sql.DriverManager;
009 import java.sql.ResultSet;
010 import java.sql.SQLException;
011 import java.sql.Statement;
012
013 import org.junit.AfterClass;
014 import org.junit.BeforeClass;
015 import org.junit.Test;
016
017 public class JUnitHSqlDBTest {
018
019     @BeforeClass
020     public static void init() throws SQLException, ClassNotFoundException, IOException {
021         Class.forName("org.hsqldb.jdbc.JDBCDriver");
022
023         // initialize database
024         initDatabase();
025     }
026
027
028     @AfterClass
029     public static void destroy() throws SQLException, ClassNotFoundException, IOException {
030         try (Connection connection = getConnection(); Statement statement = connection.createStatement()) {
031             statement.executeUpdate("DROP TABLE employee");
032             connection.commit();
033         }
034     }
035
036     /**
037      * Database initialization for testing i.e.
038      * <ul>
039      * <li>Creating Table</li>
040      * <li>Inserting record</li>
041      * </ul>
042      *
043      * @throws SQLException
044      */
045     private static void initDatabase() throws SQLException {
046         try (Connection connection = getConnection(); Statement statement = connection.createStatement()) {
047             statement.executeUpdate("CREATE TABLE employee (id INT NOT NULL, name VARCHAR(50) NOT NULL, "
048                 + "email VARCHAR(50) NOT NULL, PRIMARY KEY (id))");
049             connection.commit();
050             statement.executeUpdate("INSERT INTO employee VALUES (1001,'Vinod Kumar Kashyap', 'vinod@javacodegeeks"
051                 + "statement.executeUpdate("INSERT INTO employee VALUES (1002,'Dhwani Kashyap', 'dhwani@j"
052                 + "statement.executeUpdate("INSERT INTO employee VALUES (1003,'Asmi Kashyap', 'asmi@javac"
053                 + "connection.commit();
054             }
055         }
056     }
057
058     /**
059      * Create a connection
060      *
061      * @return connection object
062      * @throws SQLException
063      */
064     private static Connection getConnection() throws SQLException {
065         return DriverManager.getConnection("jdbc:hsqldb:mem:employees", "vinod", "vinod");
066     }
067
068     /**
069      * Get total records in table
070      *
071      * @return total number of records. In case of exception 0 is returned
072      */
073     private int getTotalRecords() {
074         try (Connection connection = getConnection(); Statement statement = connection.createStatement()) {
075             ResultSet result = statement.executeQuery("SELECT count(*) as total FROM employee");
076             if (result.next()) {
077                 return result.getInt("total");
078             }
079         } catch (SQLException e) {
080             e.printStackTrace();
081         }
082         return 0;
083     }
084
085     @Test
086     public void getTotalRecordsTest() {
087         assertThat(3, is(getTotalRecords()));
088     }
089
090     @Test
091     public void checkNameExistsTest() {
092         try (Connection connection = getConnection();
093             Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
094                 ResultSet.CONCUR_READ_ONLY);) {
095
096             ResultSet result = statement.executeQuery("SELECT name FROM employee");
097

```

```
102         if (result.last()) {  
103             assertThat("Asmi Kashyap", is(result.getString("name")));  
104         }  
105     } catch (SQLException e) {  
106         e.printStackTrace();  
107     }  
108 }  
109 }
```

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