**Database Testing Using Selenium WebDriver and JDBC API**

In our last Selenium module we learned how to troubleshoot some recurrent problems in selenium scripts. We discussed some advance concepts wherein we would deal with mouse and keyboard events, accessing multiple links by implementing lists.

Moving ahead with our advance topics , we would introduce you with the concept of **Database testing using Selenium WebDriver.**

We would discuss the basic processes like database connection, executing queries, fetching data and disconnecting database instances etc. We would also discuss various practical implications where we need Database testing with automation testing in order to test the **complete end-to-end scenarios.**

Before moving ahead with the technical implications associated with Automated Database testing. Let us discuss a few scenarios where we require performing Database testing along with the Automation Testing. But before that, I would like to affirm here that Database testing is a very peculiar type of testing whereas Selenium WebDriver is a tool used to simulate and automate user interactions with the Application UI.

So technically speaking we are not precisely performing Database Testing rather we are testing our application in conjunction with Database in order to ensure that the changes are reflected at both the ends thereby identifying defects early.

Absolutely all the web applications need a backend to store the Data. Databases like MySQL, Oracle, and SQL Server are reasonably popular these days.

Now motioning back to the original topic, let us discuss a few scenarios to exemplify the demand of Database testing along with Automation Testing.

### ****Consider the following scenarios:****

**#1)** At times, we are required to make sure that the data entered from the UI is consistently reflected at the database. Thus we retrieve the information from the Database and verify the retrieved information against the information supplied from the UI. For example, registration forms, user data, user profiles, updates and deletes of user data. Thus, the test scenario to automate can be “To verify that the user’s information is successfully saved into the database as soon as the user registers in the application”.

**#2)** Another use case of performing database testing with Selenium WebDriver may arise when the user is directed to load the test data or expected data from the Database. Thus, in such a case, user would make the connection with the Database using a third party API, execute queries to retrieve data from the dataset and then asserting the data fetched from the Database with the actual data which is populated on the Application UI.

**#3)** Another user case is to perform associative Database Testing. Assume that we performed an operation on the application’s UI, and we want to test the reflection in the Database. It may be a case that the impacted data resides in various tables of the database due to association. Therefore it is always advisable to test data reflection at all the impacted areas.

Selenium like I said simulates the user interactions with the application under test. It can simulate keyboard events, mouse actions etc. But if the user desires to automate anything outside the vicinity of browser – web application interactions, then selenium can’t be of much help. Thus we require other tools or capabilities to perform end –to –end testing.

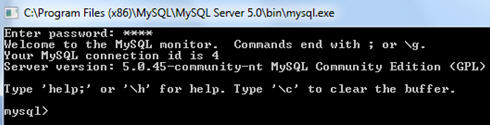
Thus, in all the above scenarios, we may require to perform Database Testing along with UI Automation. We may check business logics by manipulating the data and verifying its reflection. We may also check the technical aspects of the Database itself like soft delete, field validation etc.

Let us now move ahead with the actual implementation. Before developing Selenium WebDriver scripts to extract data from the data source, let us create a test data in the database. For this module, we would use MySQL as a database.

### ****Creation of test data in the Database****

If you haven’t downloaded the database yet, download it using the link – “<http://dev.mysql.com/downloads/>”. The user is expected to follow some basic steps to download and install the database.

Once the database is successfully installed, user can launch the MySQL Command Line Prompt which would look like the following screenshot. The application might ask the user to enter the password. The default password is “root”.

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/DB-testing-using-Selenium-11.jpg)

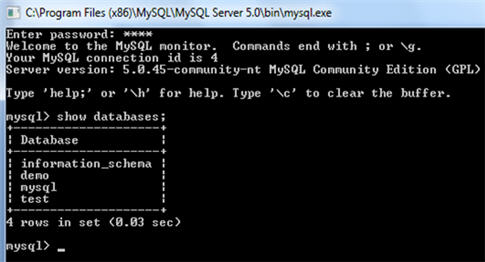
**Note**: User can also find GUI based clients over the internet to connect with the database. To name a few, user can download and install Query Browser or Work Bench.

### ****Creation of new Database****

The next step is to create the test database with few tables and records stored in those tables in order to make connection with the database and execute queries.

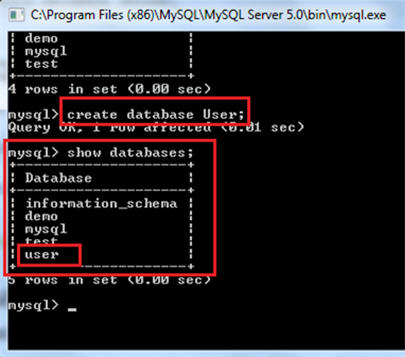
**Step 1)** Type “show databases” to see all the already available databases

show databases;

[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/DB-testing-using-Selenium-2.jpg)

**Step 2)** Type “create database user;” to create a database named “user”.

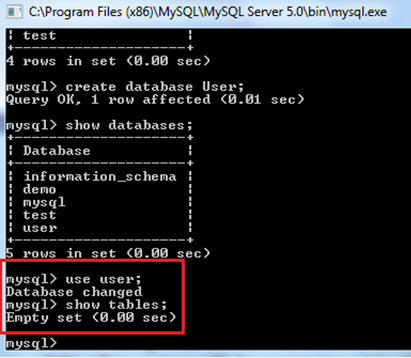
create database user;

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/DB-testing-using-Selenium-3.jpg)

Take a note that the database name as user is created and can be seen in the list of databases.

**Step 3)** Type “use user;” to select the newly created database. Also type “show tables;” to view all the tables available within the user database.

use user;show tables;

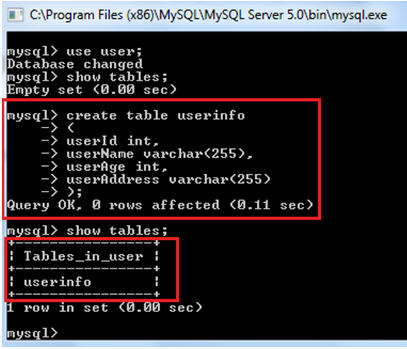
[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/DB-testing-using-Selenium-41.jpg)

Take a note that Empty set is shown in the result of the “show tables;” query as there were no tables available within the user database.

Let us now a few tables and add records in them.

**Step 4)** Type the following command to create a table with 4 fields/columns (userId, userName, userAge, userAddress).

create table userinfo  
(  
userId int,  
userName varchar(255),  
userAge int,  
userAddress varchar(255)  
);

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/DB-testing-using-Selenium-5.jpg)

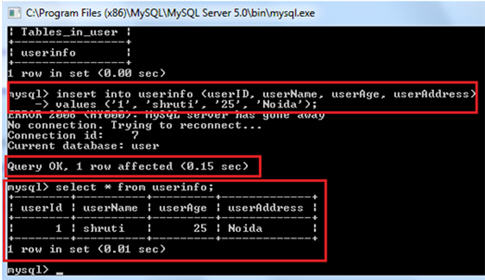
The next step is to add some data records in the “userinfo” table.

**Step 5)** Type the following command to insert data into the table a table for all the four fields 4 fields/columns (userId, userName, userAge, userAddress).

insert into userinfo (userID, userName, userAge, userAddress) values (‘1’, ‘shruti’, ’25’, ‘Noida’);

To view the added data, type the following command:

select \* from userinfo;

[](http://cdn2.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/DB-testing-using-Selenium-62.jpg)

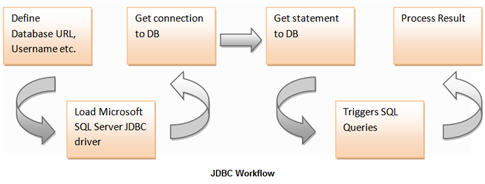
------------

Similarly, you can add more data in your table and can create other tables as well.

Now, that we have created our database. We can move ahead and understand the **implementation of automated queries to fetch the records from the database.**

As we also iterated earlier, Selenium WebDriver is a tool for UI Automation. Thus, Selenium WebDriver alone is ineligible to perform database testing but this can be done using Java Database Connectivity API (JDBC). The API lets the user connect and interact with the data source and fetch the data with the help of automated queries. To be able to exploit the JDBC API, it is required to have Java Virtual Machine (JVM) running on the system.

**JDBC Workflow**

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/DB-testing-using-Selenium-7.jpg)

**We would keep our focus aligned with the following processes:**

1. Creating connection with the database
2. Executing queries and update statements in order to extract/fetch data (CRUD Operations)
3. Using and manipulating the data extracted from the Database in the form of result set. (Result set is a collection of data organized in the rows and columns)
4. Disconnecting the database connection.

As said earlier, to be able to test database automatically from our Selenium WebDriver test scripts, we would connect with the Database via JDBC connectivity within our test scripts. Post to the connection, we can trigger as many CRUD (Create, Read, Update, and Delete) operations on the Database.

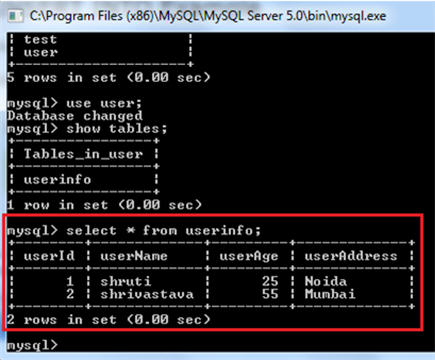
In this module we would discuss “Read operation and its variants” and about their implementation in Selenium WebDriver script. But prior to that, let us check the test scenario manually using “MySQL command line”.

**Scenario**

**1)** Open the Database server and connect to “user” database.

**2)** List down all the records from the “userinfo” table.

Syntax : select \* from userinfo;

[](http://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/DB-testing-using-Selenium-8.jpg)

**3)** Close the Database connection.

Notice that the read query will list down all the user data present in the userinfo table. The table is consisting of the following columns.

* userId
* username
* userAge
* userAddress

The result also shows that there is only a single data set present within the table.

**Now, let us execute the same scenario using Java Class.**

To be able to access Database, user is leveraged to choose amongst the diverse connector options available to connect with the Database. Most of the database connectors are freely distributed as “jar” files. As we are using MySQL as a data source, therefore we are required to download the jar file specific to MySQL.

The jar file can be downloaded from:

[here](http://www.java2s.com/Code/Jar/m/Downloadmysqlconnectorjava3114bingjar.htm) or [here.](http://en.sourceforge.jp/projects/sfnet_wwqjavaproject/downloads/mysql-connector-java-3.1.13-bin.jar/)

**Step 1**: The first and the foremost step is to configure the project’s build path and add “mysql-connector-java-3.1.13-bin.jar” file as an external library.

**Step 2**: Create a java class named as “DatabaseTesingDemo”.

**Step 3**: Copy and paste the below code in the class created in the above step.

**Code Sample**

|  |  |
| --- | --- |
| 1 | import org.junit.After; |
| 2 | import org.junit.Before; | |

|  |  |
| --- | --- |
| 3 | import org.junit.Test; |
| 4 | import java.sql.Connection; | |

|  |  |  |
| --- | --- | --- |
| 5 | import java.sql.DriverManager; | |
| 6 | import java.sql.ResultSet; |

|  |  |  |
| --- | --- | --- |
| 7 | import java.sql.Statement; | |
| 8 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | public class DatabaseTesingDemo { | | |
| 10 | | // Connection object |

|  |  |  |
| --- | --- | --- |
| 11 | static Connection con = null; | |
| 12 | // Statement object |

|  |  |  |
| --- | --- | --- |
| 13 | private static Statement stmt; | |
| 14 | // Constant for Database URL |

|  |  |  |
| --- | --- | --- |
| 15 | public static String DB\_URL ="jdbc:mysql://localhost:3306/user"; | |
| 16 | // Constant for Database Username |

|  |  |  |
| --- | --- | --- |
| 17 | public static String DB\_USER = "root"; | |
| 18 | // Constant for Database Password |

|  |  |  |
| --- | --- | --- |
| 19 | public static String DB\_PASSWORD = "root"; | |
| 20 |  |

|  |  |
| --- | --- |
| 21 | @Before |
| 22 | public void setUp() throws Exception { | |

|  |  |
| --- | --- |
| 23 | try{ |
| 24 | // Make the database connection | |

|  |  |  |
| --- | --- | --- |
| 25 | String dbClass ="com.mysql.jdbc.Driver"; | |
| 26 | Class.forName(dbClass).newInstance(); |

|  |  |
| --- | --- |
| 27 | // Get connection to DB |
| 28 | Connection con = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD); | |

|  |  |  |
| --- | --- | --- |
| 29 | // Statement object to send the SQL statement to the Database | |
| 30 | stmt = con.createStatement(); |

|  |  |
| --- | --- |
| 31 | } |
| 32 | catch (Exception e) | |

|  |  |
| --- | --- |
| 33 | { |
| 34 | e.printStackTrace(); | |

|  |  |  |
| --- | --- | --- |
| 35 | } | |
| 36 | } |

|  |  |
| --- | --- |
| 37 |  |
| 38 | @Test | |

|  |  |  |
| --- | --- | --- |
| 39 | public void test() { | |
| 40 | try{ |

|  |  |
| --- | --- |
| 41 | String query = "select \* from userinfo"; |
| 42 | // Get the contents of userinfo table from DB | |

|  |  |
| --- | --- |
| 43 | ResultSet res = stmt.executeQuery(query); |
| 44 | // Print the result untill all the records are printed | |

|  |  |  |
| --- | --- | --- |
| 45 | // res.next() returns true if there is any next record else returns false | |
| 46 | while (res.next()) |

|  |  |
| --- | --- |
| 47 | { |
| 48 | System.out.print(res.getString(1)); | |

|  |  |
| --- | --- |
| 49 | System.out.print("\t" + res.getString(2)); |
| 50 | System.out.print("\t" + res.getString(3)); |

|  |  |  |
| --- | --- | --- |
| 51 | System.out.println("\t" + res.getString(4)); | |
| 52 | } |

|  |  |
| --- | --- |
| 53 | } |
| 54 | catch(Exception e) | |

|  |  |
| --- | --- |
| 55 | { |
| 56 | e.printStackTrace(); | |

|  |  |  |
| --- | --- | --- |
| 57 | } | |
| 58 | } |

|  |  |
| --- | --- |
| 59 |  |
| 60 | @After | |

|  |  |  |
| --- | --- | --- |
| 61 | public void tearDown() throws Exception { | |
| 62 | // Close DB connection |

|  |  |  |
| --- | --- | --- |
| 63 | if (con != null) { | |
| 64 | con.close(); |

|  |  |  |
| --- | --- | --- |
| 65 | } | |
| 66 | } |

|  |  |
| --- | --- |
| 67 | } |

**The output of the above code is:**

1      shruti 25     Noida  
2      shrivastava   55     Mumbai

**Read Statement Variants**

**Where clause with single condition**

String query = “select \* from userinfo where userId='” + 1 + “‘”;  
ResultSet res = stmt.executeQuery(query);

**Output:**  
1      shruti 25     Noida

**Where clause with multiple conditions**

String Address =”Mumbai”;  
String query = “select \* from userinfo where userId='” + 2 + “‘ and userAddress='”+Address+”‘”;  
ResultSet res = stmt.executeQuery(query);

**Output:**  
2      shrivastava   55     Mumbai

**Display userId**

String query = “select userId from userinfo”;  
ResultSet res = stmt.executeQuery(query);

**Output:**  
1  
2

**Display userId with where clause**

String Address =”Noida”;  
String query = “select userId,userName from userinfo where userAddress='”+Address+”‘”;  
ResultSet res = stmt.executeQuery(query);

**Output:**  
2  
shrivastava

Thus, in the same way user can execute various queries on the database.

With this, Let us shed some light on result accessibility methods also.

**Result Accessibility Methods:**

| **Method name** | **Description** |
| --- | --- |
| String getString() | Method is used to fetch the string type data from the result set |
| int getInt() | Method is used to fetch the integer type data from the result set |
| boolean getBoolean() | Method is used to fetch the boolean value from the result set |
| float getFloat() | Method is used to fetch the float type data from the result set |
| long getLong() | Method is used to fetch the long type data from the result set |
| short getShort() | Method is used to fetch the short type data from the result set |
| double getDouble() | Method is used to fetch the double type data from the result set |
| Date getDate() | Method is used to fetch the Date type object from the result set |

**Result Navigation Methods:**

| **Method name** | **Description** |
| --- | --- |
| boolean next() | Method is used to move to the next record in the result set |
| boolean previous() | Method is used to move to the previous record in the result set |
| boolean first() | Method is used to move to the first record in the result set |
| boolean last() | Method is used to move to the last record in the result set |
| boolean  absolute(int rowNumber) | Method is used to move to the specific record in the result set |

### ****Conclusion****

Through this module, we tried to make you acquainted with the concept of **Automated Database Testing**. We clearly laid emphasis the technical implications and needs of Database Testing.

As our entire series was focused on Selenium, reader may get misled and can create an impression that this module would teach to perform Database testing using Selenium, but like I mentioned number of times earlier, anything that lies outside the periphery of UI testing, cannot be handled by Selenium. Therefore we introduce Java Database Connectivity (JDBC) API in order to perform Database Testing by embedding the code within the Selenium WebDriver scripts.

JDBC makes it possible for the java class to connect with the Database, retrieve data from the database or for the matter of fact perform any of the CRUD operations, manipulate the resultant data and close the connection.

Thus, the module constitutes of the basic sample implementation of the above mentioned process.