**Database connection in Selenium Webdriver**

**Why do we need database automation?**

* **To get test data - If we automate the database, we can directly fetch the test data from database and then work on them in test automation script**
* **To verify result - In automation we can verify the front end result with backend entry in the database**
* **To delete test data created - In automation it is good practice to delete the test data created, using database automation, we directly fire the delete query to delete the test data created**
* **To update certain data - As per the need of test script, the test data can be updated using update query**

Take a look at the two basic terms of software testing: Black box Testing and White box Testing. Both are a part of functional testing. Now think about this type of testing with the reference of **Database Testing.**

**Black Box Testing with Database Testing** => This type of testing is applied on the application to verify the incoming and outgoing data by executing queries on the Database.

**White Box Testing with Database Testing** => This type of testing is applied to validate the internal structure of the Database. Also we check the basic terminology of the Database called ACID (Atomicity, Consistency, Isolation and Durability) with the flow of the application.

**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”.**

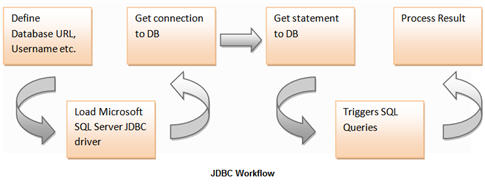
#2) Other use case of performing database testing with Selenium Webdriver may arise when the user is directed to load the test data or expected data from the Database. Thus, in such a case, user would make the connection with the Database using a third party API, execute queries to retrieve data from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

#3) Other use 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 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 region 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. This can be done by 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**



**Steps according to JDBC Workflow are as:-**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.

**Scenario**

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

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

**3)** Close the Database connection.

**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.

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

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

**Step 3**: Loading the required JDBC Driver class, which in our case is com.microsoft.sqlserver.jdbc.SQLServerDriver. You can download the jar from here and add it to your classpath or add it as maven dependency in you pom.xml file in case you are using maven project.

**For Microsoft Sql Server:-**

**Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");**

**For MySQL:-**

**Class.forName("com.mysql.jdbc.Driver”)**

**Step 4:** Creating a connection to the database-

**Connection conn = DriverManager.getConnection("DatabaseURL","UserName", "Password");**

**Database URL for MS SQL Server**

**jdbc:sqlserver://hostname;databaseName=XYZ**

**Database URL for MYSQL**

**jdbc:mysql://hostname/ databaseName**

**Step 5:** Executing SQL queries-

**Statement st = conn.createStatement();**

**String Sql = "select \* from [tableName] where <condition>";**

**ResultSet rs = st.executeQuery(Sql);**

**Step 6:** Fetching data from result set-

**while (rs.next()) {**

**System.out.println(rs.getString(<requiredField>));**

**}**

**Example Code:-**

//create a Connection object

       static Connection con = null;

       //create a Statement object

       private static Statement stmt;

       // Constant for Database URL

       public static String DB\_URL = "jdbc:mysql://localhost:3306/user";

       // Constant for Database Username

       public static String DB\_USER = "root";

       // Constant for Database Password

       public static String DB\_PASS = "root";

@Before

Public void setup() throws exception

{

// Make the database connection

                     String dbClass = "com.mysql.jdbc.Driver";

                     Class.forName(dbClass).newInstance(); [T*he basic idea behind using Class.forName(“”) is to open a class.]*

                     // Get connection to DB

                     Connection con = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASS); *[This is the Connection String]*

                     // Statement object to send the SQL statement to the Database [*As for the Database we know that there are 4 types of queries which are majorly used and these queries are known as SQL (Structured Query Language) queries as****SELECT****,****INSERT****,****UPDATE and DELETE]***

                     stmt = con.createStatement();

                     }

                     catch (Exception e)

                     {

                           e.printStackTrace();

                     }

       }

@Test

       public void test() {

              try{

              String query = "select \* from userinfo";

              // Get the contents of userinfo table from DB *{This is the container in which the program stores the Result of query }*

              ResultSet resultset = stmt.executeQuery(query);

              // Print the result untill all the records are printed

              // resultset.next() returns true if there is any next record else returns false

              while (resultset.next())

              {

                     System.out.print(resultset.getString(1));

              System.out.print("\t" + resultset.getString(2));

              System.out.print("\t" + resultset.getString(3));

              System.out.println("\t" + resultset.getString(4));

              }

              }

              catch(Exception e)

              {

                     e.printStackTrace();

              }

       }

       @After

       public void tearDown() throws Exception {

              // Close DB connection

              if (con != null) {

              con.close();

              }