

Automatic Test Case Generation for SQLUnit Testing Framework

By Jaswinder Bal (jbal2@illinois.edu),

Jianmei Fan (jmfan@illinois.edu),

Joshua Lintz (lintz2@illinois.edu)

Table of Contents

1	Introduction.....	3
1.1	SQLUnit (http://sqlunit.sourceforge.net/) ²	3
1.2	Theory of Operation.....	4
1.3	Terminology.....	5
2	Project Organization	6
2.1	Extract and Parse Database Schema	6
2.2	Auto Generation of Test Cases in SQLUnit XML Format	6
2.3	Execution Test Cases within SQLUnit	7
2.4	GUI Interface	7
3	Evaluation	8
3.1	Test Database preparation.....	8
3.2	Test Results.....	8
4	Continued Development	8
4.1	Test Case Run within SQLUnit	9
4.2	Supported Database Server	9
4.3	Written Language.....	9
4.4	Extending Test Case Generation Other Than Stored Procedures	9
5	Conclusion	9
Appendix A.	References.....	10
Appendix B.	Historical Trivia.....	10
Appendix C.	License Information	11
Appendix D.	Installation Guide.....	11
Appendix E.	Using PHP Script to Covert SQL File to XML	12

1 Introduction

Database management systems are widely used in many applications. The data stored in the databases is an important corporate asset and it is therefore important that the database system is error-free and stable. Many applications rely on the DBMS for actual implementation as well. The best way to ensure the reliability of this is to do DBMS testing regularly. Testing can also help eliminate bugs early on and save a lot of time in implementing the system. DBMS testing, in general, is a labor intensive, time-consuming process, often performed manually. Automating DBMS testing not only reduces development costs, but also increases the reliability in the developed systems.

Currently, SQLUnit is one unit-testing tool that regression tests stored procedures in databases. But it requires that testers manually choose values for the database tables, choose inputs, and design checks for the output. It also requires the tester to write these test cases manually in XML format. Our goal is to provide ability to auto-generate test cases for SQLUnit. This will make using this open source DBMS testing tool easier, more efficient, less manual work, and less error prone.

1.1 SQLUnit (<http://sqlunit.sourceforge.net/>)

An SQLUnit test suite is written as an XML file. The SQLUnit harness, which is written in Java, uses the JUnit unit-testing framework to convert the XML test specifications to JDBC calls and compares the results generated from the calls with the specified results. We have shown a specific example below.

```
<test name="Adding department HR again">
  <call>
    <stmt>{call add_dept(?,?)}</stmt>
    <param id="1" type="VARCHAR" inout ="in" is-null="false">Human Resources</param>
    <param id="2" type="VARCHAR" inout ="out" is-null="false">@status</param>
  </call>
  <result>
    <outparam id="2" type="VARCHAR">add_dept: Department already exists</outparam>
  </result>
</test>
```

SQLUnit runs this test file from the command line from within Ant.

1.2 Theory of Operation

Figure 1 shows the SQLUnit test generator we have developed. It has a GUI interface, so a user can interact with it directly. First, it allows the user to enter server/database credentials to connect. Doing so, it also automatically generates the database schema file (schema.xml). Or if the user already has a schema file he wants to use, he can load it by using the "Browse" button. The lists (tables, columns, stored procedures, parameters) will be auto filled after schema file has been loaded. User can then click on the "Generate Tests" button to generate the SQLUnit tests. The tests generated will show in the "Tests" list box below. It will state each stored procedure name, parameters, input values, etc. The user will then have the ability to modify these values. Finally, the user can click on the "Save" button to save the tests in XML format to disk. The user can then take the test.xml file and run it in SQLUnit manually. Or he can click on the "Run" button to launch SQLUnit with test.xml

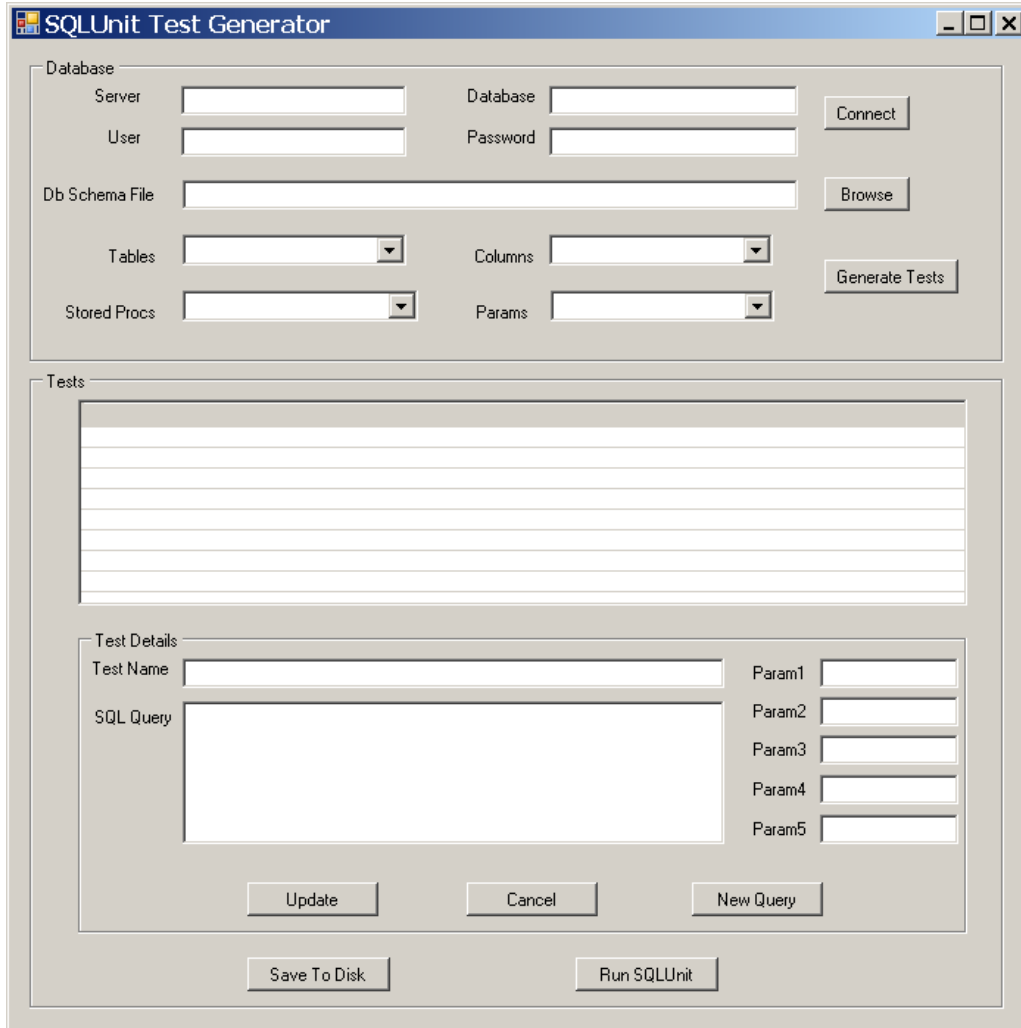


Figure 1 SQLUnit Test Generator

1.3 Terminology

The terminology descriptions contained in this section along with Figure 1 above describe the terminology associated with the SQLUnit Test Generator.

1.3.1 Database

Work is currently being done to use SQLUnit Test Generator on a MySQL database server. User and Password are used to connect to this server. Database is the database we want to use in this server.

1.3.2 DB Schema File

The DB Schema File is a file with the database schema and stored procedures in XML. The user has the choice to load it directly from the file browser or SQLUnit Test Generator will automatically extract this information from the server.

1.3.3 Tests

Tests are XML files that can be used to run SQLUnit tests.

2 Project Organization

We implemented a technique for extracting and parsing target database schemas into an XML file in order to identify objects that need to be unit tested, e.g., stored procedures for SQLUnit. Our tool then uses the resulting files to automatically generate test cases (with test data and expected results) in SQLUnit XML format. The generated test cases then can be executed from within SQLUnit. We provided a GUI interface to do it. We implemented this in C#/.Net and provided independent auto test case generation libraries.

See attachment for code.

2.1 Extract and Parse Database Schema

There are several ways to extract the database schema into XML format. We studied some of these options. We looked into an open source framework ServingXML (<http://servingxml.sourceforge.net/>) for this purpose. The framework provides a conversion from one file format (e.g. SQL) into XML format. We also looked at exporting database schema into SQL format and then converting it into XML. We found we could write a script (in a language such as PHP) to export the database schema into XML (see Appendix E). Eventually, we decided to write our own script to automate this function within our GUI tool, using MySQL.Data.dll, which is an ADO.Net driver for MySQL.

A DBschema Example in XML format.



Dbschema.xml

2.2 Auto Generation of Test Cases in SQLUnit XML Format

After extracting the stored procedure from the database, we parsed the resulting XML file into a format that SQLUnit can use. We then ran it within SQLUnit and got the expected results. Those results were used as test cases to regression test the stored procedure.

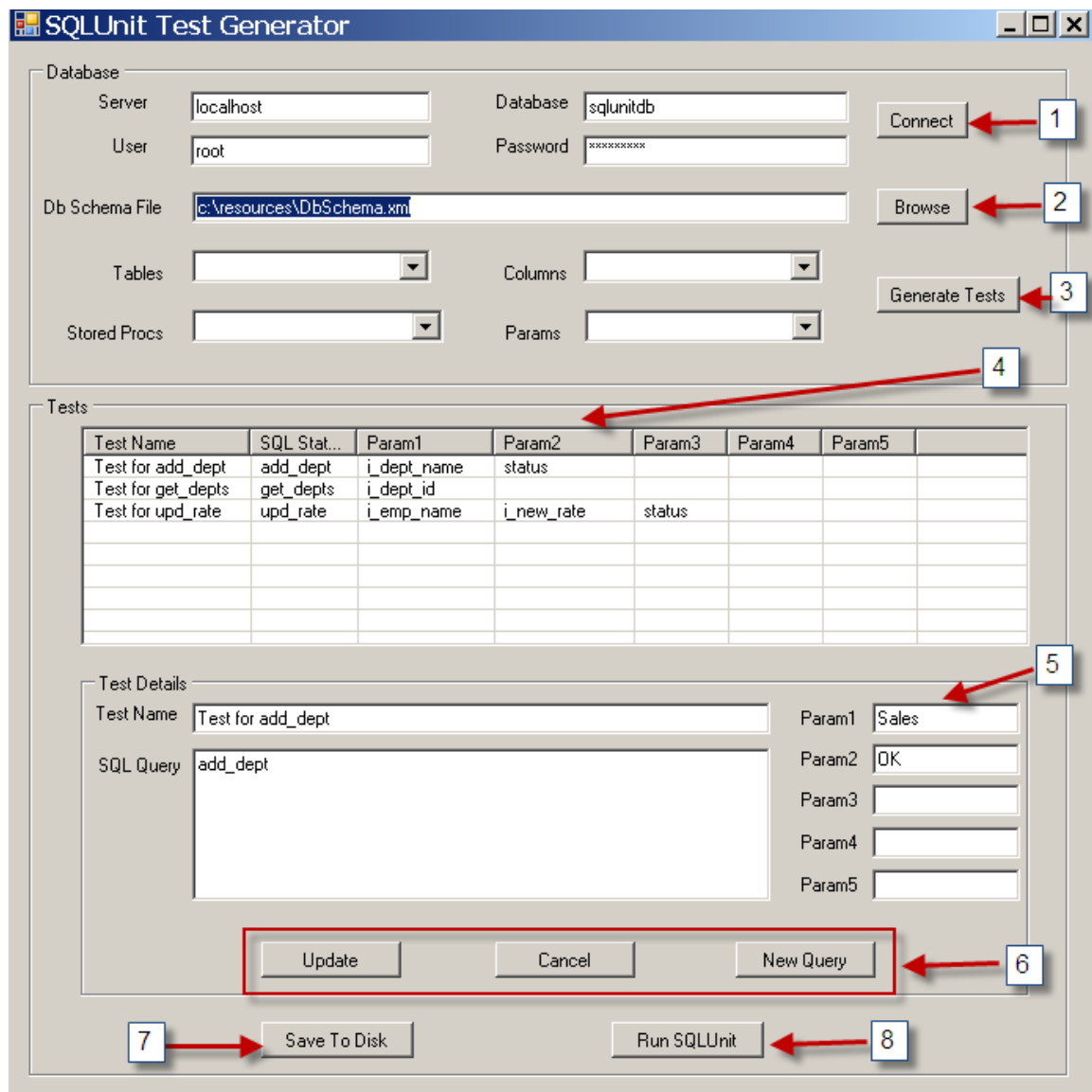
The tool provides default values for stored procedure parameters, keeping into account database field constraints. The user has the option to use default test values or to modify them. The user also has the ability to add more test cases through the GUI. The user also has the option to provide the expected test results themselves through the GUI interface. The tool will then generate test cases based on the results provided by the user. We can however, just parse the database schema SQL file directly into an XML format that SQLUnit can use, but we prefer this way because it is easier to implement.

2.3 Executing Test Cases within SQLUnit

Our GUI provides an interface to SQLUnit that automates loading of unit test cases into SQLUnit. The GUI has the ability to automatically launch SQLUnit within itself and provide results. Alternatively, the user can take generated test case file from the tool and save it to run later with SQLUnit manually.

2.4 GUI Interface

The GUI Tool is a .Net based application for generating test cases by running the stored procedure against the database. It provides the following features as shown below.



1. User can connect to a database system (MySQL) by providing the login parameters. This will also automatically generate the database schema. Current release only supports MySQL, but this can be extended to support other RDBMSs.
2. Alternatively, user can load a database schema file by clicking on the Browse button.

After the database schema file has been loaded into the tool, it populates the tables, fields, stored procedures, and their parameters into the GUI.

3. User can click on the "Generate Tests" button to automatically generate SQLUnit test cases.
4. The tool displays test cases generated in above step.
5. User has the option to view and modify the test case inputs as required.
6. User can update test case inputs by clicking on the "Update" button. User can also enter a new SQL query and add it to test cases by clicking on the "New Query" button. Currently only supports SELECT, but we can add support for INSERT, DELETE , UPDATE.
7. User can save the test cases to disk for running them later with SQLUnit.
8. User can run it with SQLUnit from within the GUI tool.

3 Evaluation

3.1 Test Database preparation

We prepared two small test databases in MySQL with stored procedures. One company database has three tables: employee, department and timecard. It has seven stored procedure: add_dept, add_employee, add_timecard, get_depts, get_emp_by_name, get_emp_in_dept, and upd_rate. Another database is a small model of a bank which has five tables: AccountType, Account, Address, Customer, and TransactionHistory. It has the stored procedures cust_account, cust_info, and trans_history. We also looked into a large test database from <https://launchpad.net/test-db/+download>. (See attachment files).

3.2 Test Results

We ran these databases within our tool. We were able to successfully automatically generate test cases for these databases. See screenshots above for the example.

4 Continued Development

The SQLUnit Test Generator is still under development. There are a number of functions that have not been implemented that are being worked on, or will be worked on in the future.

4.1 Test Case Run within SQLUnit

The "Run" button is supposed to launch SQLUnit automatically from the GUI itself. At the time of this report, this feature is still in development and hasn't been fully implemented. We plan to complete this shortly. Until then, user will need to "Save" the test XML file to the file system and then supply it to SQLUnit when running it manually.

4.2 Supported Database Server

The current version of our tool is only tested on MySQL. Since SQLUnit supports a variety of databases such as Oracle, PostgreSQL, Microsoft SQL Server, Informix SE, etc. The team's intention is to have future versions supported on these various databases.

4.3 Written Language

We implemented our tool in C#.Net. However, SQLUnit is written in Java and uses the JUnit unit-testing framework. Our tool can be better integrated into SQLUnit if we had written it in Java. If time and resources permit in the future, another version can be written in Java.

4.4 Extending Test Case Generation Other Than Stored Procedures

Since SQLUnit is only for testing database stored procedures, our tool is designed to auto-generate test cases only for stored procedures as well. If we want to extend our tool to test triggers, table constraints, and transactions, we first need to extend SQLUnit to do these as well. This is not trivial and requires a lot of research and work.

5 Conclusion

Although SQLUnit has its own GUI tool to execute test cases within SQLUnit, it doesn't offer the ability to automatically parse database schema, extract stored procedures and generate test cases from it. We have developed SQLUnit Test Generator that can retrieve database and stored procedure information and automatically generate test cases into the SQLUnit XML file format. Generated test cases can then be run in the same GUI interface. This makes database regression testing faster and easier.

This class gave us a broad and thorough introduction to software testing, while this project offered us the opportunity to learn more about database testing in particular. We have read quite a few research papers. Now we understand the need for database testing, what the current research interest is, and how people are doing their research. After we came up with this idea to extend SQLUnit for auto-generation of test cases, we played with it quite a bit and were able to use it fluently. Eventually we were able to develop a GUI interface to realize this idea. We have learned a lot from this whole process.

Appendix A. References

- [1] Testing Database Transactions with AGENDA
by Deng, Frankl, Chays (ICSE 2005)
- [2] Using an SQL Coverage Measurement for Testing Database Applications
by Suarez-Cabal, Tuya (SIGSOFT 2004)
- [3] A Family of Test Adequacy Criteria for Database-Driven Applications
by Kapfhammer, Soffa (ESEC/FSE 2003)
- [4] Query Aware Test Generation using the Alloy Tool-set
by Abdul Khalek, Elkarablieh, Laleye, Khurshid (ASE 2008)
- [5] QAGen: Generating Query-Aware Test Databases
By Carsten Binnig, Donald Kossmann, Eric Lo and M. Tamer Ozsu
- [6] Query-based test generation for database applications
By David Chays, John Shahid, Phyllis G. Frankl
- [7] Database Testing: How to Regression Test a Relational Database
By Scott W. Ambler
- [8] Automatic Test Generation for Database-Driven Applications
By Zhenyu Dai, Mei-Hwa Chen

Appendix B. Historical Trivia

Coming up with a good research project topic was a challenge for this CS527 class. Originally we had the idea of doing an automatic database test generation project, but there are so many areas to work on. Before focusing our effort on the current topic, we also did some research on database testing. Automatic test case generation might include automatic data generation and automatic query generation.

Here is a list of tools that either make this process partially or totally automatic.

- A. Automatic data generation: DB2 test data generator, DTM Data generator, MUDD
- B. Automatic query generation: RAGS, QGen
- C. Test execution and comparison: unit test tool such as SQLUnit and DBUnit
- D. One interesting tool that combines test case generation, test execution, and test comparison for database test: ADUSA (automatic query aware test generation plus expected query result as test oracle)
- E. There are some tools that combine test case generation, test execution, and test comparison for testing applications that interact with a database: AGENDA (focus on transaction test); QAGen (focus on query-aware test databases generation)

Scott W. Ambler gives a good summary of database testing in this essay on his website at <http://www.agiledata.org/essays/databaseTesting.html>

We have three potential project topics focusing on different aspects:

1. Extending SQLUnit to do automatic test case generation, execution and comparison.
2. Understand and implement ADUSA testing tool.
3. Extending jCUTE to generate test databases for database application. Alloy, in this case, can be used as a relational constraint solver to generate the databases.

1 is easier than 2, 3 (1>2>3).

1 can be used to help 2 to do test execution and comparison.

2 can be used to help 3 since 2 also use the Alloy tool.

For our project, we decided to focus on SQLUnit alone and are working to develop a tool/technique for auto-generation of unit test cases for SQLUnit. We also looked into topic 2, ADUSA implementation. The core part is using Alloy, because of Alloy's relational basis it provides a natural fit for modeling relational databases and query operations. It needs SQL2Alloy and Alloy2SQL translation tools. [Here are few examples to illustrate this process.](#) However, as professor Darko Marinov pointed out, the general translation is fairly hard, we would need to parse SQL and manipulate the trees. So we abandoned this idea. Nevertheless, we have tremendously benefited from this process and learned a lot from this class.

Appendix C. License Information

All source code of the SQLUnit Test Generator project is licensed under GNU GPL.

Appendix D. Installation Guide

Install SQLUnit and MySQL

There are several software components that need to be installed properly in order for SQLUnit to run. The SQLUnit manual itself doesn't provide detail on these installation processes. So the following procedure is written for beginner. It should be relatively easy to follow.

1. Download Ant (apache-ant-1.7.1), sqlunit-5.0, MySQL, Java with JDK (jdk1.6.0_07, jre folder itself will not work) and JDBC driver (<http://dev.mysql.com/downloads/connector/j/5.1.html>) under c:\program files folder. Copy mysql-connector-java-5.1.6-bin.jar to C:\Program Files\Java\jdk1.6.0_07\jre\lib\ext

2. Create sqlunitdb database under MySQL,
Create table customer under sqlunitdb
Customer schema under C:\Program Files\sqlunit-5.0\test\mysql\schema.sql

3. Fill in user and password for C:\Program Files\sqlunit-5.0\test\mysql\test.xml
<connection>
<driver>com.mysql.jdbc.Driver</driver>

```
<url>jdbc:mysql://localhost:3306/sqlunitdb</url>  
<user>?your user name?</user>  
<password>?your password?</password>  
</connection>
```

4. Run CMD under c:\program files\sqlunit-5.0
Setup ANT_HOME, JAVA_HOME and PATH
set ANT_HOME=C:\program files\apache-ant-1.7.1
set JAVA_HOME=C:\program files\JAVA\jdk1.6.0_07
set PATH=%PATH%;%ANT_HOME%\bin

5. Run

```
c:\program files\sqlunit-5.0\ant  
c:\program files\sqlunit-5.0\ant install  
c:\program files\sqlunit-5.0\ant sqlunit-flat -Dtestfile=test\mysql\test.xml  
You can run ant help to see SQLUnit commands
```

6. Installing SQLUnit Test Generator GUI tool: The tool is distributed as an executable. Unzip contents of install.zip. Then launch GUI tool by double clicking on the SQLUnitTestGenerator.exe file.

Install Location:

<https://netfiles.uiuc.edu/xythoswfs/webview/fileManager.action?entryName=/jbal2&stk=3C11FF347FD4F77&msgStatus=2%20documents%20have%20been%20successfully%20uploaded%20to%20the%20folder%2C%20jbal2>.

Source Code location:

<https://netfiles.uiuc.edu/xythoswfs/webview/fileManager.action?entryName=/jbal2&stk=3C11FF347FD4F77&msgStatus=2%20documents%20have%20been%20successfully%20uploaded%20to%20the%20folder%2C%20jbal2>.

Appendix E. Using PHP Script to Covert SQL File to XML

The web page http://www.webcheatsheet.com/PHP/export_database_schema_xml.php contains a script to convert a MySQL database schema to XML format. The comments on the bottom give more direction including getting the data as well. We originally used this script to create the schema.xml file but later did it in C#.NET in order to work well with the GUI. PHP string manipulation can also be used to convert SQL files to XML.