Information Management and Business Review
Vol. 5, No. 3, pp. 108-112, Mar 2013 (ISSN 2220-3796)

Metadata Extraction in Database Testing

*Zuhoor A. Al-Khanjari, Sara Al-Kindy
College of Science, Sultan Qaboos University, Sultanate of Oman
*zuhoor@squ.edu.om

Abstract: The need for an automated testing tool to test the correctness of the database applications is crucial in our current day since databases play an important role in almost all organizations. Also, database's behavior need to be verified in order to avoid costly errors and false information being extracted from them. The main aim of this paper was to create a component-based tester called DBSoft that tests the correctness of database application systems. The DBSoft toolkit consists of five tools as follows: information collection with the Parser tool, test case generation with the Input Generator tool, test case implementation with the Output Generator tool, test case validation with the Output Validator tool and report generation with the Report Generator tool.

Keywords: *Databases, Metadata, Testing, Metadata Extraction*

1. Introduction

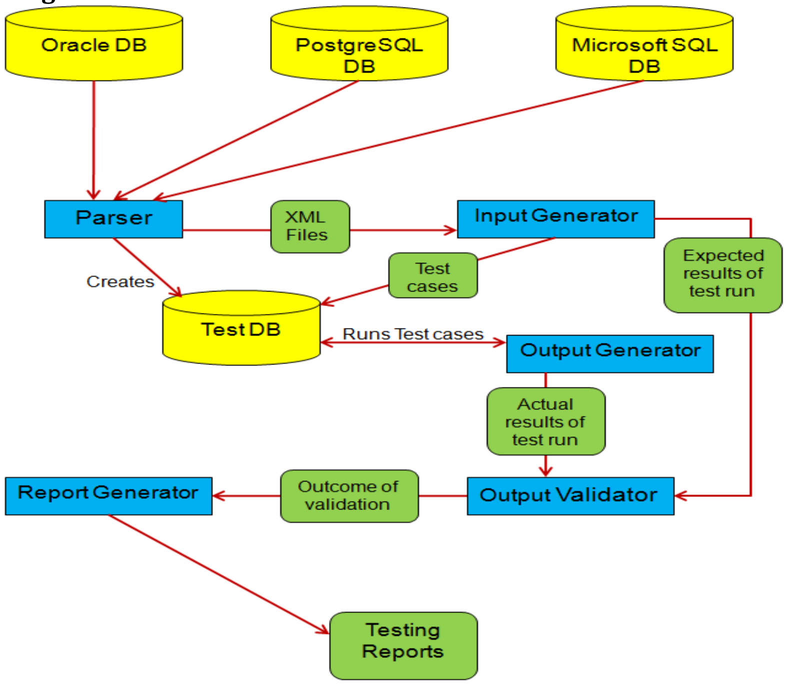
The amount of data being collected in today's world is at an explosive rate. Database management systems (DBMSs) play crucial role in providing a means of storing and accessing data. Most organizations deal with a certain form of DBMS, as they provide ease at interacting with large amounts of data efficiently by stripping away the low-level technicalities of how the data is being accessed. However, dealing with DBMSs can also prove to be expensive in terms of time and money to organizations when fallacies exist within their application program. Mistakes, or bugs, may appear at any time during the development of the database application program; the planning phase, the implementation phase or the data population phase. Thus, there is a crucial need for an automated method of testing the correctness of database application programs. One would think that the amount of research being invested in the field would be vast given the importance of database systems in our day to day life. Unfortunately, the opposite is true. Testing databases is not easy. The state of the database needs to be considered while testing, in addition to it having performed the specified actions correctly and produced the desired output. Databases are large entities that contain several tables structured in a specific way, and the states of these tables will have to be validated as test cases are run. The goal of our research is to ultimately create a component-based tester toolkit for database application systems. Our toolkit, deemed DBSoft, consists of five components:

- **The Parser:** involved in collecting information about the database being tested, creating XML files consisting of the collected information and generating a test database.
- **The Input Generator:** involved in creating test cases using the information in the XML files from the parser tool, populating the test database with the test cases and creating an XML file of the expected results of each test case.
- **The Output Generator:** involved in running the test database with the populated test cases and saving the results.
- **The Output Validator:** involved in validating the results produced by the Output Generator by comparing it with the expected results from the Input Generator.
- **Report Generator:** involved producing reports, graphs, and other information about the test runs.

In this paper, we will discuss the Parser. It is the first component of the toolkit and is involved in information collection. This will be achieved by means of accessing RDBMSs metadata, and will be outputted in a standard XML format. The XML documents will be used in generating test cases and their expected outcome as well as generating a test database. The rest of the paper is organized as follows. Section 2 provides an overview of related works and tools. Section 3 discusses DBMSs metadata. Section 4 introduces the DBSoft Parser. Section

5 highlights the applications of DBSoft. Section 6 includes conclusion remarks as well as future direction of work.

Figure 1: the internals of the DBSoft Toolkit



2. Related works and tools

The most closely related work to what is being proposed in this paper is that done by Chays et al (Chays et al. 2004; Chays and Deng 2003; Deng et al., 2005) with the AGENDA toolkit. It is composed of five components: a parser, a state generator, an input generator, a state validator and an output validator. By means of the five interacting components the toolkit parses a database application, generates test cases for it and validates the result. The AGENDA parser is the main point of interest for now. Based on a modified form of PostgreSQL's (PostgreSQL 2012) internal parser, it collects relevant information about a database supplied into it and stores the information in an internal database called the AGENDA DB. The main two differences between our approach and that of Chays et al are:

- Instead of developing our own parser, we chose to extract the relevant information from an RDBMS's metadata.

- Our toolkit does not store the information in tables, but represents it internally as Java objects and also outputs it as standard XML documents. An earlier version of the toolkit stored the information in an internal database, just as AGENDA did. However, it was decided that it is not efficient in terms of simplicity and extensibility. Instead, XML was chosen as our method of information storage due to its nature of being self-descriptive, easily processed and human-readable. The XML tags and attributes are based on AGENDA DB's tables.

In (Haftmann et al., 2007) a framework is presented to perform efficient regression tests on database applications, while in (Haftmann et al., 2005) issues with the automatic running of database regression tests are listed. One of the aims of the DBSoft toolkit is for it to enable performing regression tests on database applications as they are updated. DBSoft will generate test cases and a test database state by means of the information stored in XML documents produced in the data extraction step. The test cases will be used to run the test database. There a number of different methods in creating the test cases, and these are discussed in (Chen et al., 2003; Bati et al., 2007; Mishra et al. 2008; Tuya et al., 2007). A framework for creating a test database is presented in (Lo et al., 2010) and (Bruno and Chaudhuri, 2005), while in (Binnig et al., 2007) the framework tests the features of RDBMSs and also includes an automatic database generator called QAGen. Due to security and confidentially issues tied with "live" database data, an automatic data generating tool is proposed in (Lyons, 1977) called ADG. Automatic data generation is also highly useful in terms of being able to create a desired problem situation within a database for testing. DBSoft will contain an automatic data generator in order to populate the test database with. There is no evident related work in the academic community to that of collecting information from RDBMS metadata to be used in database testing, and it is safe to say the DBSoft toolkit is the first to apply this concept.

3. DBMS Metadata

Metadata is defined to be data about data. Within the internals of RDBMSs, there lies storage of data about the tables contained within them. A RDBMS metadata contains information on its databases, such as table names and the number of rows within each, to procedures names and their implementation code. The SQL standard defines a uniform means to accessing this data, but not all RDBMSs implement this feature. Hence, a number of different mechanisms have evolved with accessing metadata over different systems:

- With PostgreSQL these will be in the form of system catalogs and the INFORMATION_SCHEMA.
- SQL Server also contains system catalogs and the INFORMATION_SCHEMA (but the method for querying it differes from that of PostgreSQL)
- With Oracle it is by means of data dictionary tables and metadata registry.

PostgreSQL is an object-relational DBMS, originally developed at UC Berkeley. It is open-source, and therefore we will bring our focus to PostgreSQL's (version 8.4.5) metadata in this paper and shall concentrate upon its system catalogs and INFORMATION_SCHEMA. System catalogs in PostgreSQL are regular tables that store schema metadata on a particular database. There are a total of 60 system catalog tables, each dealing with a specific type of metadata. The INFORMATION_SCHEMA is a set of views that provides information about objects defined in the current database. It is portable and more stable due to be it being defined in the SQL standard, contrary to the system catalogs mentioned above which are specific to PostgreSQL. There are a total of 51 views. The information stored within each form of metadata is nearly the same. However there does exist some small differences between the two with the amount of information stored. Hence, they each complement the other in terms of the information that they contain, and so we employ information extraction upon both rather than concentrating on one only. PostgreSQL's metadata storages can be accessed using an application programming interface (API) such as the Java Database Connectivity API (JDBC) (JDBC, 2012). JDBC enables Java applications to interact with databases. The relevant information can be extracted from the metadata sources using SQL commands. Figure 2 shows an example for if one wishes to know the names of all the tables in a database by means of the INFORMATION_SCHEMA. The last part of the command ensures that the names of the tables contained within the system catalogs and the INFORMATION_SCHEMA will not be retrieved as well.

Figure 2: querying the INFORMATION_SCHEMA for a database's table information

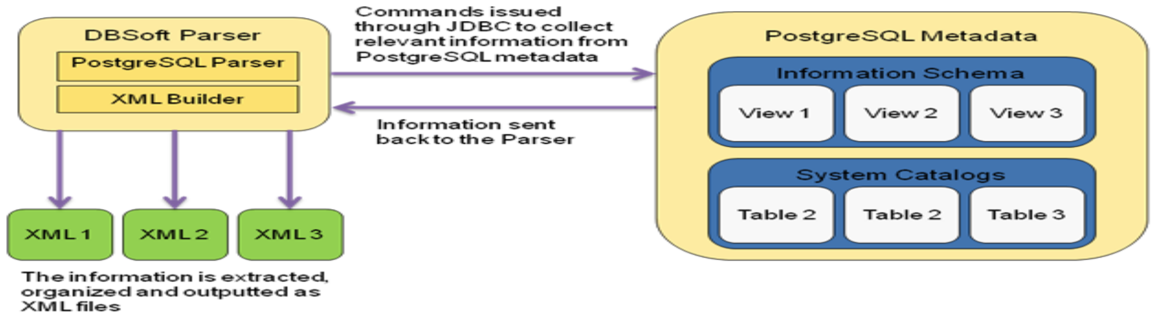
```
     SELECT table_name
       FROM
information_schema.tables  WHERE
table_type = 'BASE TABLE'
        AND table_schema NOT IN
             ('pg_catalog',
'information_schema');
```

This renders both the system catalogs and the INFORMATION_SCHEMA, together, a trove of information fit to be employed in parsing a PostgreSQL database. Additionally, due to the nature of where the information is being collected from it is safe to say that the integrity of the information collected is increased and hence more trustworthy.

4. DBSoft Parser

The DBSoft toolkit consists of five steps: information collection, test case generation, test case implementation, test case validation and report generation. The stepping stone into the DBSoft testing tool is the Parser. With it, a database can be parsed and relevant information will be outputted as XML documents. In order to be able to create an efficient and real-world usable tool for database testing, the first consideration is that databases need to be transformed into a uniform object, as there are a number of RDBMS in existence today with different mechanisms. The creation of standard XML documents, outlining the details of the internals of database schema, will also aid in creation of efficient test cases and a test database for testing, generation of test data for database population and also validating the results of the test runs.

Figure 3: DBSoft Parser collecting information from PostgreSQL metadata sources


DBSoft Parser
PostgreSQL Parser
XML Builder
Commands issued through JDBC to collect relevant information from PostgreSQL metadata
Information sent back to the Parser
PostgreSQL Metadata
Information Schema
View 1
View 2
View 3
System Catalogs
Table 2
Table 2
Table 3
XML 1
XML 2
XML 3
The information is extracted, organized and outputted as XML files

The following are steps taken by the DBSoft Parser:

User inputs location of database: The DBSoft Parser collects the relevant information from the RDBMS's metadata. The program needs to be pointed to the location of the database that is to be tested in order to access the metadata. The user will provide the correct information on the location of the database. This includes the host, the port number, the database name, the username, and the password.

DBSoft parser creates connection: Once the location of the database is inputted correctly, the DBSoft Parser will create a connection to the database by means of JDBC. (Talk more details about how the connection to JDBC is created).

Information collected from metadata: The DBSoft Parser will begin collecting the relevant information from the Database's metadata. This is done by querying the metadata by means of relevant commands through the JDBC API. Since the mechanism of accessing metadata differs from RDBMS to another, the commands issued will be according to the RDBMS in question.

Information outputted as XML documents: The information that has been collected in the parsing step will be organized and outputted as XML documents, such as:

- Tables (containing information on tables, number of attributes, type, etc)
- Attributes (containing information on attributes, tables they are contained in, type, etc)
- Boundary Values (containing information on boundary values, range, type, etc)

Applications of DBSoft Parser: The XML documents produced by the DBSoft parser can be employed in several testing applications. The aim for the DBSoft Toolkit is to cover most if not all database testing methods. Analysis can be made in regards to whether the application program is behaving as has been specified (i.e.: checking correctness), in addition to reflecting upon whether the database schema correctly models the organization of real-world data. Test cases can be produced by using the information in the XML documents, such as creating specific queries that will make a database application 'break'. Data tailored for specific problem areas found within the database can be generated, and in turn will be populated within a test database that has been built using the information in the XML documents. Regressions tests can be performed on databases when they are updated, ensuring that everything still works as specified. Validity of the results will be made using the information extracted in the parsing step (constraints, types, etc) in addition to looking into other automatic means. We hope to have offered the database testing community a means of starting the creation of a standard RDBMSs testing methodology. Though a mythology has not been presented in the paper, a stepping stone into it has been with the standardizing application of the DBSoft parser.

5. Conclusion & Future Direction

The need for an automated method in testing database applications is crucial in our current day. Databases form a large part of an organization's workings, and hence would need to behave correctly to avoid errors and mistakes. The DBSoft toolkit aims to create an efficient real-world application for database testing. For now, the first step of the model has been employed with the DBSoft Parser collecting information about an input database from its metadata. The second step in the creation of the DBSoft model is the creation of test cases, hence the DBSoft Tester. A standard method of inputting database information (i.e.: my means of the XML documents produced by the DBSoft parser) will be employed within this component. A range of different test cases will be generated and employed in checking the correctness of the databases. Another aim is to enable the DBSoft Tester to be used for regression testing on databases, as the test cases generated by it can be stored and run several times.

References

Bati, H., Giakoumakis, L., Herbert, S. & Surna, A. (2007). A genetic approach for random testing of database systems, Proceedings of the 33rd international conference on Very large data bases, 2007.

Binnig, C., Kossmann, D., Lo, E. & Özsu, M. T. (2007). QAGen: generating query-aware test databases, Proceedings of the 2007 ACM SIGMOD international conference on Management of data, 2007.

Bruno, N. & Chaudhuri, S. (2005). Flexible database generators, Proceedings of the 31st international conference on Very large data bases, 2005.

Chays, D. & Deng, Y. (2003). Demonstration of AGENDA Tool Set for Testing Relational Database, ICSE '03 Proceedings of the 25th International Conference on Software Engineering, 2003.

Chays, D., Deng, Y., Frankl, P. G., Dan, S., Vokolos, F. I. & Weyuker, E. J. (2004). An AGENDA for testing relational database applications. Software Testing, verification and reliability, 2004.

Chen, T. Y., Poon, P. L. & Tse, T. H. (2003). A Choice Relation Framework for Supporting Category-Partition Test Case Generation, *IEEE Transactions on Software Engineering,* 29(7), 577-593.

Deng, Y., Frankl, P. & Chays, D. (2005). Testing database transactions with AGENDA, Proceedings of the 27th international conference on Software engineering, 2005.

Haftmann, F., Kossmann, D. & Kreutz, A. (2005). Efficient regression tests for database applications. Conference on Innovative Data Systems Research (CIDR), 95-106, 2005.

Haftmann, F., Kossmann, D. & Lo, E. (2007). A Framework for efficient regression tests on database applications. *The VLDB Journal — The International Journal on Very Large Data Bases,* 16(1), 145-164.

JDBC Drive (Online) http://jdbc.postgresql.org/ [Accessed on 10/12/2012].

Lo, E., Binnig, C., Kossmann, D., Ozsu, M. T. & Hon, W. K. (2010). A Framework for Testing DBMS Features. *VLDB Journal,* 19(2), 203–230.

Lyons, N. R. (1977). An automatic data generating system for data base simulation and testing. *ACM SIGMIS Database,* 8(4), 10-13.

Mishra, C., Koudas, N. & Zuzarte, C. (2008). Generating targeted queries for database testing, Proceedings of the 2008 ACM SIGMOD international conference on Management of data, 2008.

PostgreSQL. (2012). online http://www.postgresql.org/ [Retrieved on 20/12/2012]

Tuya, J., Cabal, M. J. S. & Riva, C. D. l. (2007). Mutating database queries. *Information and Software Technology,* 49(4), 398-417.