Milestone 1

Calvin Li, Ben Mishkanian

1 Introduction

Database engines, like any other software, must be tested for functional correctness. Random testing, which involves randomly generating input values, is easy to implement but on its own cannot test for correctness; it can only test for crashes or hangs. Instead, we can use differential testing, where we run the same input on two programs, in order to test correctness. This approach works well in testing compilers, but not as well for testing database engines, because SQL is not as standardized as other programming languages. Different engines have slightly different implementations of SQL, so pure differential testing cannot fully cover all the cases.

We will approach this problem from a different angle. We will generate a random query, then run it to get the correct output. Then, we will make some transformations to the query in a way where the new query is functionally equivalent to the old one, and run this new query. If the results are different, then the query is possibly error-revealing.

Initially, we'll target SQLite. It uses a relatively simple flavor of SQL, making it easier to work with. It also has not been tested as much as other engines, which means there are more bugs, so we should have more interesting results.

2 Motivation

Say we have a database with tables `A` and `B`. `A` has a column `Income`, and `B` has column `Age`. Before we start generating random queries, we determine that nobody has an income over 10000 and an age over 50. Then we can add the clause

```
A.Income > 10000 & B.Age > 50
```

to the `where` clause of any query that uses `A join B`. Then, the output should be the same. We can do this and similar transformations, along with combinations of them, in order to look for discrepancies.

3 Technical Approach

Our project will make use of Random Query Generator (`https://launchpad.net/randgen`) in order to generate data and queries. We will first use it to generate some data, which is done by giving it the number of tables, number of rows, and types of data we want in the columns. Then, we will run queries to find some predicates that are universally true of false for the data.

Afterwards, we must generate a SQLite query using Random Query Generator or a similar tool. These tools require a file listing all the rules involved, which allows us to exclude and parts of the grammar that might give us difficulties in the later steps.

Then, we parse the query so we can do the transformations. For this, we use ANTLR (`http://www.antlr.org/`), which also requires a grammar file, but in a different format. We can use it to build a parse tree showing which rules were used for which tokens in the query.

After we obtain a parse tree, we can manipulate it by adding, removing, and moving around nodes based on our transformation rules. Then we take the transformed query and run it, comparing the output to the original.

4 Evaluation

We will evaluate by repeatedly generating, transforming, and running queries. By incrementing the seed in our random number generator, we can guarantee that each run will be different but repeatable. If we encounter any discrepancies between a transformed query and its original, we will first check for bugs in our code, and then examine the query for possible bugs in the database engine. We will report any bugs we find. Our system is based on random testing, so we can keep it running in the background, constantly looking for bugs.

5 Related Work

A large body of work deals with testing and creating queries for SQL databases. Tuya, Cabal, and Riva[Tuya et al. 2006] provide the SQLMutation tool, which generates mutants of a given SQL query, useful for detecting bugs in database test suites. Khalek and Khurshid[Abdul Khalek and Khurshid 2010] describe a method that ensures syntactic and semantic correctness of generated queries by using the Alloy tool to reduce it to a SAT problem. These queries can then be fed to the ADUSA tool, which automates black-box database testing by running queries and comparing the results to generated test oracles. Bati, Giakoumakis, Herbert, and Surna[Bati et al. 2007] present a genetic programming algorithm which generates a set of queries that achieve higher code coverage in the DBMS under test.

[Elmongui et al. 2009] Presented a framework and general principles for testing SQL query transformations. Particularly relevant is their ideas concerning test coverage, ensuring that there are are multiple queries that utilize each transformation rule. There is also a MSR paper [Grabs et al. 2008] that outlines testing query optimizers. Since query optimizers also transform queries in an output-preserving way, some of the guidelines used to test them also apply to our project. Finally, [Mishra et al. 2008] presented a paper on generating target queries for database engine testing. However, their work mainly deals with producing outputs near a target cardinality. Although this might help us generate test cases that run faster, it is an optimization that is inessential for our testing.

6 Contributions

Since there was a lot of reading, we up split the articles when looking for related work. Afterwards, we summarized the relevant parts. Ben then worked on the related work section, while Calvin wrote the introduction, motivation, technical approach, and evaluation using parts of the proposal. After everything else was done, Ben wrote the future work section.

7 Future Work

Before the milestone #2 deadline, we will use Random Query Generator to generate sample databases and random queries. We will create grammars describing valid query transformations and use these grammars to transform the random queries. In order to provide preliminary proof-of-concept results, we may manually manipulate a few grammars to generate transformed queries, and run these queries against the SQL server. The output will be compared to that of the original query, in order to evaluate our approach.

References

ABDUL KHALEK, S., AND KHURSHID, S. 2010. Automated sql query generation for systematic testing of database engines. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ACM, New York, NY, USA, ASE '10, 329–332.

BATI, H., GIAKOUMAKIS, L., HERBERT, S., AND SURNA, A. 2007. A genetic approach for random testing of database systems. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB Endowment, VLDB '07, 1243–1251.

ELMONGUI, H. G., NARASAYYA, V., AND RAMAMURTHY, R. 2009. A framework for testing query transformation rules. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ACM, New York, NY, USA, SIGMOD '09, 257–268.

GRABS, T., HERBERT, S., AND ZHANG, X. S. 2008. Testing challenges for extending sql server's query processor: A case study. In *Proceedings of the 1st International Workshop on Testing Database Systems*, ACM, New York, NY, USA, DBTest '08, 2:1–2:6.

MISHRA, C., KOUDAS, N., AND ZUZARTE, C. 2008. Generating targeted queries for database testing. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ACM, New York, NY, USA, SIGMOD '08, 499–510.

TUYA, J., SUAREZ-CABAL, M., AND DE LA RIVA, C. 2006. Sqlmutation: A tool to generate mutants of sql database queries. In *Mutation Analysis, 2006. Second Workshop on*, 1–1.