Salayze

Version 1.0

Amir Raminfar

Computer Science 342

The George Washington University

Wednesday, April 25, 2007

Contents

Introduction

Salayze is a static SQL analyzer for Java applications. Before analyzing any Java files, Salayze establishes a connection to MySQL database and parses tables and columns that are affiliated with each table. This data is later used to cross reference the column and table names found in the Java files.

Salayze looks for possible SQL queries by finding JDBC entry points, such as the "excuteQuery" method, to MySQL JDBC driver. For each entry point, a collection of all possible SQL statements are generated. All the generated SQL statements are first parsed for correctness and then cross referenced to ensure that column and table names are correct.

Due to lack of time, Salayze only supports SELECT statements. Column types can be of type VARCHAR or INT which respectively maps to a String or Integers in Java. Java files are assumed to have been compiled successfully.

Design

Salayze consists of three major components: 1) Database Parser 2) Java File Parser 3) SQL Analyzer.

Database Parser

Database Parser needs a host address, username and password to connect. After a successful connection, "SHOW TABLES" is executed to get a list of all the tables. For each table, "SHOW COLUMNS FROM <table_name>" is executed. This query will return all columns and the associated column types. This information is later used to check the correctness of SQL statements by the SQL analyzer.

Java Parser

This component needs to first parse the Java file. The parser will search for entry points that use the JDBC driver. A suggested method is to look for "executeQuery". This method takes one SQL string as an input and returns the results. For example, statement.executeQuery("SHOW * FROM table") is valid; however, statement.executeQuery(*sqlString*) or statement.executeQuery(*someString* + *anotherString*) are also valid. For each entry point, Java Parser will look for all variables or strings that are used as inputs.

The second task is to search for each variable and generate a collection of all possible values. For example, the above variable *sqlString* will be searched in the Java file and return a collection of all possible values that *sqlString* can hold. A major hurdle is handling condition statements such as if-else statements.

Below is an example of a possible Java code:

```
String q = "SELECT";
if (1 == 1) {
    q += " bad_column";
} else if (3 == 3) {
    q += " int_type, string_type";
}
```

The above example will result in three possible values: “SELECT bad_column”, “SELECT int_type, string_type” or just “SELECT”. Note that the above condition can actually result in “SELECT” because it is an if-elseif statement. If it was just an if-else statement then only two possible values would be valid.

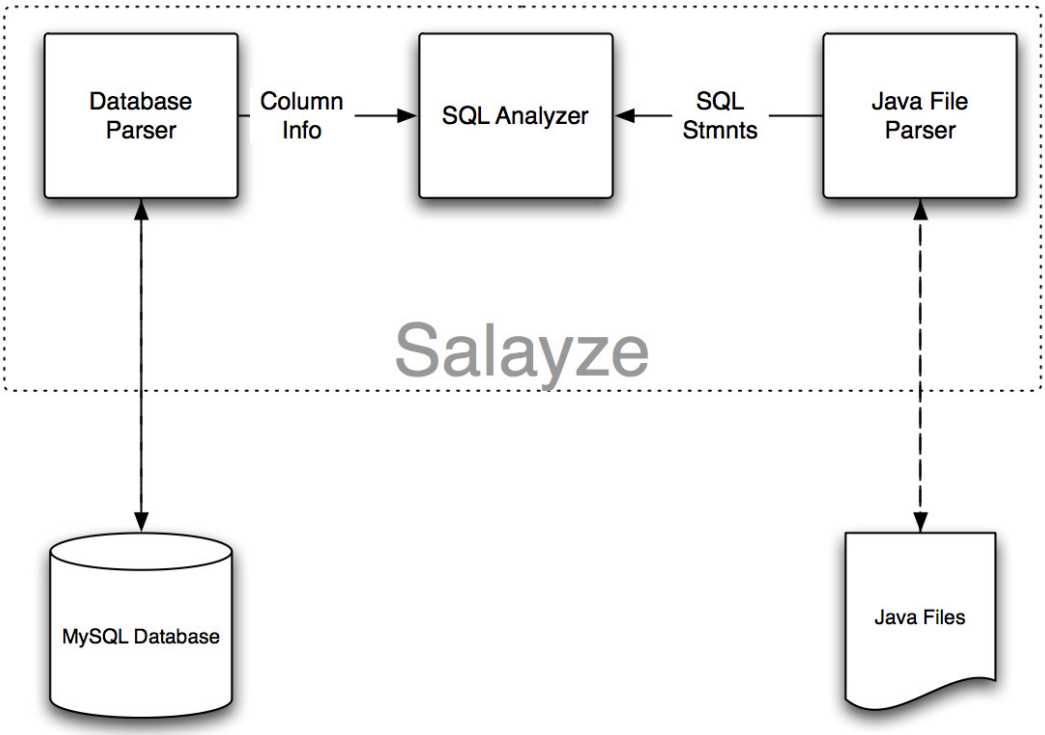
If our example had been executeQuery(*someString* + *anotherString*) then first all possible values had to be computed for *someString* and then all possible values for *anotherString*. All possible values would be a result of joining *someString* with *anotherString*. In other words, if *someString* has 3 possible values and *anotherString* has 4 possible values then it would result in a total of 12 possible values.

Finally, there might be such a case that another variable, such an integer is referenced. For example, q = “SELECT * FROM table WHERE id=” + someNumber. In this case someNumber needs to be looked up and find the type it correlates to.

SQL Analyzer

At this point, all SQL strings have been generated and the database has been parsed. However we need to still check for the correctness of these generated strings. SQL Analyzer will first check that the SQL statement has the correct syntax. Assuming that the syntax is correct, all table and column names are cross-referenced to ensure that they do exist. Last, if there exists a case where a Java variable was referenced, then the column type has to be checked to see if it matches the Java variable type.

Below is a block diagram which displays all three components in relation to each other:



Implementation

There are two third-party tools that were used to implement Salayze. Antlr was used as a parser. Antlr parses a Java file and can iterate a collection of tokens. Each token can be a literal “if” or a string such as “Hello World”. Due to the possibility of nested if-statements stacks are used to hold possible queries for each condition scope.

There are a total of three stacks needed. First stack holds all possible SQL queries for the current scope. The second stack holds the scope before the condition-statement and the last stack holds the condition type, such as if-statement, else-if or else. When an if-statement is found, the current scope in the stack is saved twice. First it is saved to the current scope stack because any new queries that are found will manipulate the first item in this stack. Second, the scope before the condition is saved because it will be needed when an if-else or else block is found. The end of each scope causes three pops and current scope stack is appended to the next item in the scope stack. At the end, the scope stack should hold one item which is a list of all possible values.

The second third-party tool that was used is ZQL. ZQL is a SQL analyzer which checks the correctness and syntax of a single SQL statement. This tool does not take Java variables in consideration. When a Java variable is found, such as int number = 0, it is saved as ‘java_var_int_number’. The variable name is ‘number’ and the type is ‘int’. To check the correctness of these statements, dummy values are used in the SQL string to represent a place holder for Java integers and strings. If the SQL statement is valid, then the WHERE clause is parsed for errors by matching variable types with columns types.

In total, Salayze consists of about 1100 lines of code.

Remaining Work

This version includes a lot of features but many had to remain incomplete due to time. The majority of new tasks are in parsing Java files. The list below represents the work that is still remaining:

- Currently queries which are constructed via a loop are not supported.
- Many of the test cases had the Java code written in one scope such a main method; although it has not been tested but results might be unreliable.
- A great feature would be to validate the return data from a SELECT statement.
- Only INT and VARCHAR are supported. One idea would be to also support DATE, TIMESTAMP, TEXT and BOOL.
- VARCHAR can have a length of up to 255. An idea would be to also check the length of Java variables.

Conclusion

This project was designed and implemented as part of a school project. There was limited amount of time and resources. There was limited time to test each component independently. However, due to limited time there might exist many unforeseen bugs. Had there been more time, Salayze could have been a better project. Salayze still generated acceptable and correct results.

References

1. ANTLR - http://www.antlr.org
2. ZQL - http://www.experlog.com/gibello/zql
3. Static Checking of Dynamically Generated Queries in Database Applications - http://www.cs.ucdavis.edu/%7Esu/publications/icse.pdf