Introduction

Introduction and Abstract

Testing is an important part of the development of any computer system. In particular, testing needs to be thorough, so that any errors in the program are picked up before the software is distributed to clients and is subject to the tough tests in real usage. The testing also needs to be easily repeated and modified, so that the programmers can easily trace and correct the sources of the errors, and so that the tests can be run again at a later point in time to ensure that changes in the code have not caused it to malfunction.

This testing report outlines the method used to test the query generator (parser) that is fundamental to the operation of the database, for it is the means by which users and the future graphical user interface will communicate with the database and interact with the information stored in it. A query parser that creates trees that are an accurate representation of the user's intentions is critical to the function of the system. Indeed, because of the focus on representing the user's intentions, the determination of a successful parse is driven by not only what is written down in the query parser documentation and the help files, but also what I would expect to be able to do with this parser intuitively.

Specifically, this document outlines tests being performed on the classes in the package `soft1902.server.query`, and in particular `QueryGenerator` and `TokenIterator`.

Testing Method

These ideas were in mind during the development of the testing mechanism for the query parsing system. The test logic, while coded in Java, is fully controlled by an XML file that describes the keywords, constructs, literals and operators that are used by the testing class to create permutations to feed into the query parsing system. The test class is able to take this description and produce all possible permutations[1] fitting a particular pattern:

`[`*`ACTION`*`] [`*`TYPE`*`] [SET | WHERE]` *`(set or where clauses)`*

Because there are an extraordinary number of possible permutations, it would be computationally time-consuming, and the results would be difficult to analyse due to the volume that must be processed. Furthermore, one error in the query parsing system will fail all permutations of the query string that utilises that failed part of the query parser, creating a flood of errors and rendering analysis difficult.

To combat this, smaller XML files that contained a subset of the full set of possible instructions were used to test one aspect of the system at a time. This also allowed for the concept of a control in the experimentation. For example, when testing whether the parser can successfully parse `SELECT` queries, only the constructs specific to the `SELECT` part of the

[1] Note: the permutation methods were tested by hand before use by outputting the permutations into a file and checking them. Permutations are used to ensure thoroughness to ensure that the system does not just pass for one particular form of the query string. It also saves typing multiple configurations out.

query, such as the Types (Event, Project, User, File, Access) and the operators for the `WHERE` clause. In this example, the variable names and string literals were kept constant to simple values that were known have parsed successfully in previous tests.

For each permutation created, it is put into the parser, and a resulting `Query` object with children nodes is created. If any exceptions were raised during parsing, they are recorded in the output file. If the parse proceeds without exceptions, the `toString()` method was called on the resulting `Query` object, and then the input string is (content-wise, not exactly) compared with the `toString()` representation. If this process deems that they are not a match, a fail is recorded in the output file. Statistics are then computed for success rate, and tests may be modified and run again depending on the need for additional data.

Testing Classes

The testing classes reside in `soft1902.util.testing` with the xml files in the project's root folder. When you run the testing class, it will prompt you to specify the location of the master XML file (see `EnochTests.xml` in the appendix) that tells it where to find all the other test descriptor files, and the location of the output file. For convenience, this information was piped into the program at the command prompt.

Tests and Results

Test 1: Boolean Parsing

(Note: Before formal testing using this XML-based system, I was practising how to use the parser when I noticed that it could not parse even simple queries with brackets. For example:

Input: `insert user set (c!=d)`
Output: `soft1902.server.query.QueryException: (E:5)Expected Set operator statement`

Input: `insert user set ((c = d) or (d = f))`
Output: `soft1902.server.query.QueryException: (E:5)Expected Set operator statement`

Input: `select events where (((name=josh`
Output: `SELECT events WHERE name==josh`

I notified Josh, who then updated the parser to correct this problem, which allowed the following tests to be run.)

Listing: See `EnochParsingTestBoolean.xml`
Purpose: To test the ability of the parser to handle AND and OR statements, nested within each other, and within brackets, up to a specified level.
Input: The nesting depth was set to 3 (which is just about how deep a normal user would go when hand-coding queries from a business database), and all the input data would be considered *normal* case.

- Actions: either the delete or select queries would have sufficed (as they use `WHERE`), and select could have been used in place of delete in the XML file
- Boolean Statements: the patterns `#X and #X`, and `#X or #X` are listed for testing
- Types, Operators, Variables and Values: only one was chosen, to act as control

Result in output file:

```
=====================================================================
Running EnochParsingTestBoolean.xml:
=====================================================================
Begin testing ParsingTest::NormalCase...

Statistics for this session:
    total queries executed: 722
    total queries parsed successfully: 722
    percentage: 100.0%
```

Conclusion: The parser handles Boolean queries up to a depth of 3 successfully.

Test 2: Value Parsing

Listing: See `EnochParsingTestValue.xml`
Purpose: To test the ability of the parser to recognise different types of values that may appear on the right hand side of an operator sign (such as =). This is important, as users are allowed to enter anything into the system, and the parser should recognise a diverse range of characters.
Input: Some of the data could be considered normal case, while others boundary. The following table lists the values selected for testing, and provides a short justification:

Value Text	*Explanation*
`normal`	Normal case with just alphabet characters
`“normal”`	Normal case with quotes
`p`	Boundary case with one character long string
`“z”`	Boundary case with one character long string with quotes
`“select users where name=fred”`	Normal case with a query-like string, the system should treat it as plaintext and not tokenize
`123`	Normal case of numerals
`-9876`	Normal case of numerals starting with minus sign
`.29378`	Normal case of numerals starting with decimal point
`“384.192”`	Normal case of numerals in quotation marks
`“-2.3928E-13”`	Normal case of numerals in quotation marks
`“;&gt;&lt;&amp;-,;~!@#$%^*().:[]{}/+=-_”`	Normal case, using all of the non-alphanumeric printable characters on keyboard except for \ and “
`“\\\\”`	Normal case with escape characters (slashes)
`“\”\””`	Normal case with escape characters (quotes) – since quotes are a special character, the user expects some way of entering them in a string

- Actions: The update query was chosen as it uses both `WHERE` and `SET` statements, thus testing values in both locations at once.
- Types, Variables and Boolean Statements: only one of each were chosen, for control purposes (note use of `#X` in the Boolean statement part, this indicates that no Boolean operators are to be used)
- Operator: one of each type so that the `WHERE` and `SET` statements function

Result in output file:

```
=====================================================================
Running EnochParsingTestValue.xml:
=====================================================================
Begin testing ParsingTest::NormalCase...

Failure #1 (mismatch)
    input: update event where (test<normal) set test-=p
    query: UPDATE events WHERE test<normal SET test-=

Failure #2 (query exception)
    input: update event where (test<normal) set test-=-9876
    query: null

Failure #3 (query exception)
    input: update event where (test<normal) set test-="\"\""
    query: null

Failure #4 (mismatch)
    input: update event where (test<normal) set test=p
    query: UPDATE events WHERE test<normal SET test==
```

… [truncated since it is too long] …

```
Failure #947 (query exception)
    input: update event where (((test="\"\""))) set test="\\\\"
    query: null

Failure #948 (query exception)
    input: update event where (((test="\"\""))) set test="\"\""
    query: null

Statistics for this session:
    total queries executed: 2028
    total queries parsed successfully: 1080
    percentage: 53.25443786982249%
```

Note: if the query is `null` it means that it could not parse successfully

Analysis: Clearly, there are errors in the parsing of values on the right hand side of an operator. This encouraged the idea of testing each value separately to see which one(s) were causing the errors (the individual XML files have not been included in the appendix). This is summarised as follows:

Value Text	*Problem*
p	One character strings are not parsed correctly; they appear as blank text
-9876	Could not parse; this is probably because the symbol - is reserved for the token -= and hence a solution is to simply put it in brackets
“\\\\”	Escape characters are not recognised
"\"\""	Escape characters are not recognised

Indeed, the pass rate changes to 100% when these lines have been removed from the XML file.

Conclusion: Most normal case values are parsed successfully, although there is a bug in that one character strings disappear. In terms of features, a normal user would expect that -9876 would parse without brackets, and that escape characters be implemented.

Test 3: Variable Name Parsing

Listing: See `EnochParsingTestVariable.xml`
Purpose: To test the ability of the parser to parse variable names that appear on the left hand side of operators.
Input: The four cases listed appear to be quite fair variable names to the normal user. It is therefore not unrealistic that the user may expect that these four will parse correctly. Below is a table listing the four cases, with a justification as to why they are included:

Variable Name	*Justification*
test	Normal case
test123	Normal case involving numeric characters
under_score_	Normal case with underscore characters
hyp-hen-ated	Normal case with hyphen characters

- Actions: again, update is chosen because it utilises both `WHERE` and `SET` statements
- Types and Boolean Statements: one of each again for control purposes
- Operators: one of each type so that `WHERE` and `SET` statements work
- Values: one was chosen for control

Result in output file:

```
=====================================================================
Running EnochParsingTestVariable.xml:
=====================================================================
Begin testing ParsingTest::NormalCase...

Failure #1 (query exception)
    input: update event where (test<normal) set hyp-hen-ated-=normal
    query: null

Failure #2 (query exception)
    input: update event where (test<normal) set hyp-hen-ated=normal
    query: null

Failure #3 (query exception)
    input: update event where (test=normal) set hyp-hen-ated-=normal
    query: null

Failure #4 (query exception)
    input: update event where (test=normal) set hyp-hen-ated=normal
    query: null
```

… [truncated] …

```
Failure #107 (query exception)
    input: update event where (((s=normal))) set hyp-hen-ated-=normal
    query: null

Failure #108 (query exception)
    input: update event where (((s=normal))) set hyp-hen-ated=normal
    query: null

Statistics for this session:
    total queries executed: 300
    total queries parsed successfully: 192
    percentage: 64.0%
```

Conclusion: By examining the results, it is quite clear that all the errors involve the `hyp-hen-ated` variable name. As before, this is by design, so this is not really a problem. Note also that Java does not support the use of hyphens in variable names, so this failure is quite fair.

Test 4: SELECT Query Parsing

Listing: See `EnochParsingTestSelect.xml`
Purpose: To test the ability of the parser to correctly parse a `SELECT` query, and all the operators that go with a `SELECT` query.
Input: All cases in this test are considered to be *normal* case.

- Types: we have listed the 5 types of objects in the database with their alternate spellings
- Operators: all operators relevant to the `SELECT` query have been listed, that is, the where operators and the generic operators
- Boolean Statements and Literals: one from each only, to act as control

Result in output file:

```
===================================================================
Running EnochParsingTestSelect.xml:
===================================================================
Begin testing ParsingTest::NormalCase...

Statistics for this session:
    total queries executed: 243
    total queries parsed successfully: 243
    percentage: 100.0%
```

Conclusion: It can be concluded that SELECT query parsing occurs correctly.

Test 5: INSERT Query Parsing

Listing: See `EnochParsingTestInsert.xml`
Purpose: To test the ability of the parser to correctly parse an `INSERT` query, and all the operators that go with a `INSERT` query.
Input: All cases in this test are considered to be *normal* case.

- Types: we have listed the 5 types of objects in the database with their alternate spellings
- Operators: all operators relevant to the `INSERT` query have been listed, that is, the set operators and the generic operators
- Boolean Statements and Literals: one from each only, to act as control

Result in output file:

```
===================================================================
Running EnochParsingTestInsert.xml:
===================================================================
Begin testing ParsingTest::NormalCase...

Statistics for this session:
    total queries executed: 36
    total queries parsed successfully: 36
    percentage: 100.0%
```

Conclusion: It can be concluded that INSERT query parsing occurs correctly.

Test 6: DELETE Query Parsing

Listing: See `EnochParsingTestDelete.xml`
Purpose: To test the ability of the parser to correctly parse a `DELETE` query, and all the operators that go with a `DELETE` query.
Input: All cases in this test are considered to be *normal* case.

- Types: we have listed the 5 types of objects in the database with their alternate spellings
- Operators: all operators relevant to the `DELETE` query have been listed, that is, the where operators and the generic operators
- Boolean Statements and Literals: one from each only, to act as control

Result in output file:

```
=====================================================================
Running EnochParsingTestDelete.xml:
=====================================================================
Begin testing ParsingTest::NormalCase...

Statistics for this session:
    total queries executed: 243
    total queries parsed successfully: 243
    percentage: 100.0%
```

Conclusion: It can be concluded that `DELETE` query parsing occurs correctly.

Test 7: UPDATE Query Parsing #1

Listing: See `EnochParsingTestUpdate1.xml`
Purpose: To test the ability of the parser to correctly parse an `UPDATE` query *without a `WHERE` clause*, and all the operators that go with a `UPDATE` query.
Input: All cases in this test are considered to be *boundary* case, as the `WHERE` clause has been left out (which is allowed).

- Types: we have listed the 5 types of objects in the database with their alternate spellings
- Operators: all operators relevant to the `UPDATE` query have been listed, that is, the set operators and the generic operators
- Boolean Statements and Literals: one from each only, to act as control

Result in output file:

```
=====================================================================
Running EnochParsingTestUpdate1.xml:
=====================================================================
Begin testing ParsingTest::NormalCase...

Failure #1 (mismatch)
    input: update event set varname123+="normal"
    query: UPDATE events WHERE name>= SET varname123+=normal

Failure #2 (mismatch)
    input: update event set varname123-="normal"
    query: UPDATE events WHERE name>= SET varname123-=normal

Failure #3 (mismatch)
    input: update event set varname123="normal"
    query: UPDATE events WHERE name>= SET varname123==normal

Failure #4 (mismatch)
    input: update event set varname123=="normal"
    query: UPDATE events WHERE name>= SET varname123==normal
```

… [truncated] …

```
Failure #35 (mismatch)
    input: update access set varname123="normal"
    query: UPDATE access WHERE name>= SET varname123==normal

Failure #36 (mismatch)
    input: update access set varname123=="normal"
    query: UPDATE access WHERE name>= SET varname123==normal

Statistics for this session:
    total queries executed: 36
    total queries parsed successfully: 0
    percentage: 0.0%
```

Analysis: We really do not have a failure as the query generator is writing a default `WHERE` clause where it does not appear. The failures are reported simply because the simple matcher method does not know this and returns false.
Conclusion: The parser does `UPDATE` queries (without `WHERE` clause) correctly.

Test 8: UPDATE Query Parsing #2

Listing: See `EnochParsingTestUpdate2.xml`
Purpose: To test the ability of the parser to correctly parse an `UPDATE` query *with a WHERE clause*, and all the operators that go with a `UPDATE` query.
Input: All cases in this test are considered to be *normal* case.

- The input parameters are exactly the same as Test 7.

Result in output file:

```
======================================================================
Running EnochParsingTestUpdate2.xml:
======================================================================
Begin testing ParsingTest::NormalCase...

Statistics for this session:
    total queries executed: 972
    total queries parsed successfully: 972
    percentage: 100.0%
```

Conclusion: The parser can handle `UPDATE` queries (with a `WHERE` clause) correctly.

Test 9: Illegal and Special Cases

Listing: See `EnochParsingTestIllegal.xml`
Purpose: To test the ability of the parser to fail to parse a malformed query, and its ability to correctly parse special cases built into the querying system for ease of typing.
Input: The cases considered in the first section of Test 9 are *illegal*, and the cases considered in the latter section are *boundary* cases. The reason why illegal cases have been written specifically into the XML file is that a computer cannot be taught to permute over all illegal cases, because there are an infinite number of them. Instead, we have decided to check that the parser fails to parse specific classes of malformed queries. See the XML source for details as to the justification for including each of the tests in.
Result in output file:

```
======================================================================
Running EnochParsingTestIllegal.xml:
```

```
======================================================================
Begin testing ParsingTest::IllegalCase...

Failure #1 (parsed successfully)
    input: select users where name==beowulf and
    query: SELECT users WHERE name==beowulf AND null

Failure #2 (parsed successfully)
    input: select users where bobsuncle
    query: SELECT users WHERE name>=

Statistics for this session:
    total queries executed: 20
    total queries parsed successfully: 18
    percentage: 90.0%

======================================================================
Begin testing ParsingTest::SpecialCase...

Failure #1 (warning - need manual checking)
    input: select access
    query: SELECT access WHERE name>=

Failure #2 (warning - need manual checking)
    input: delete events
    query: DELETE events WHERE name>=

Statistics for this session:
    total queries executed: 2
    total queries parsed successfully: 0
    percentage: 0.0%
```

Analysis:

- Illegal case: the query generator appears to default the right hand side of a Boolean statement if it is missing. This is undesirable, and it should be changed to throw an exception.
- Illegal case: the query generator appears to use the default `WHERE` clause if it just consists of a single variable name followed by nothing else. This is undesirable, and it should be changed to throw an exception.
- Special cases: the matcher built into the test class cannot determine special cases and calls on the user to check manually that they have been parsed correctly. The two queries have been formed correctly and have defaulted to the expected values.

Conclusion: The parser seems to handle illegal cases quite sufficiently, denying the large proportion (90%) of the illegal cases specified. However, it did allow some malformed queries to pass through, and as such, it needs to be tightened up to prevent such queries from being parsed.

Conclusion

The query parser functions in an expected manner for the large majority of times. The only 'bug' found is that single character values on the right hand side of an operator are not parsed, and disappear. The two illegal cases that parsed are not fatal to the application, but it would be desirable if their behaviour were changed. In conclusion, the query parser was tested comprehensively, by using permutations to ensure that no combination is left unchecked, and by testing normal, boundary and illegal cases. These tests are readily available for repetition if the author of the query generator wishes to fix those problems.