DB Project1: SQL Parser

2014 - 17184 Yeon-Woo Kim

Explanation on Algorithm and Implementation

Writing a .jj file according to the given grammar is trivial for the most part. The few things that have to be considered are left-recursion and need for lookahead within the grammar. Left recursions can be eliminated by changing the grammar in a certain way. The other problem can easily be solved by using lookahead function provided by JavaCC. While a lot of students must have solved the problem using lookahead, I tried to solve it without any lookahead by eliminating the initial token duplicates in the grammar.

First, I defined a new non-terminal named *tableColumn*.

<tableColumn> ::= <legalIdentifier> (<period> <legalIdentifier>)?

Then, I replaced all *tableName* and *columnName* with *legalIdentifier*, and (<tableName><period>)? <columnName> with *tableColumn*. This effectively eradicates the need for lookahead, since the problem comes from not being able to distinguish the type of the legal identifier (table or column) when receiving input of the form of *tableColumn*.

Next, I removed left recursion from *booleanValueExpression* and *booleanTerm*. This is done simply by changing the grammar as follows:

<booleanValueExpression> ::= <booleanTerm> (<or><booleanTerm>)*

<booleanTerm> ::= <booleanFactor>(<and><booleanFactor>)*

Note that there might be a problem when further proceeding the project because this change of grammar actually changes the semantics of the or and and operation from left-associative to right-associative.

Finally, the non-terminal *predicate* has two cases with the same initial token. That is, *comparisonPredicate* and *nullPredicate* can all start with *legalIdentifier*. This yields a problem of not being able to differentiate between the two when the parser sees a *legalIdentifier* from the token stream. To solve this problem, I divided *predicate* into two different cases: *identifierPredicate* and *constantPredicate*. Concrete grammar is as follows:

<predicate> ::= <identifierPredicate> | <constantPredicate>

<identifierPredicate> ::= <tableColumn> (<compOp><compOperand>|<nullOperation>)

<constantPredicate> ::= <comparableValue> <compOp> <compOperand>

identifierPredicate represents predicates that start with an identifier. *constantPredicate* denotes predicates that start with a constant comparable value.

In addition to the changes in grammar, I also made a message printer class that prints all the prompts and result messages.

* Note that I implemented <is null> and <is not null> as a separate token,

This makes the design more compact and elegant but typing multiple whitespace or tab between the words would result in syntax error.

How to compile & execute program

java -classpath <your javacc.jar path> javacc ScratchDBMSGrammar.jj

javac ScratchDBMSParser.java

java ScratchDBMSParser

Take a look at Makefile for more concrete instructions.

Comments on project 1-1

I listened to compiler course and there I implemented a parser almost from scratch. It was a lot of work. However, by using JavaCC, I could easily construct a parser just by typing in the given grammar into .jj file.