DB2017

Project 1-1 : SQL Parser

2012-11249 컴퓨터공학부 설재완

1. 핵심 모듈과 알고리즘에 대한 설명

default package/SimpleDBMSGrammar-2.jj 파일에서 grammar를 수정해주면 그에 따라 .java파일이 수정된다. SimpleDBMSGrammar-2.jj파일은 크게 options, PARSER\_BEGIN~PARSER\_END, SKIP, TOKEN, Parser functions(Grammar)로 이루어진다.

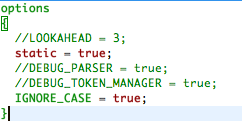
options에서 여러가지 옵션을 줄 수 있다. 대표적인 옵션으로는 DEBUG\_PARSER(Parser가 Parsing할 때 함수의 call과 return을 trace할 수 있게 함), DEBUG\_TOKEN\_MANAGER(Parser가 parsing할 때 TOKEN을 어떻게 인식하고 consume했는지 확인할 수 있게 함), IGNORE\_CASE(대소문자 구분 없이 인식), LOOKAHEAD(Parsing시, 확인하는 TOKEN수 설정)등이 있다.

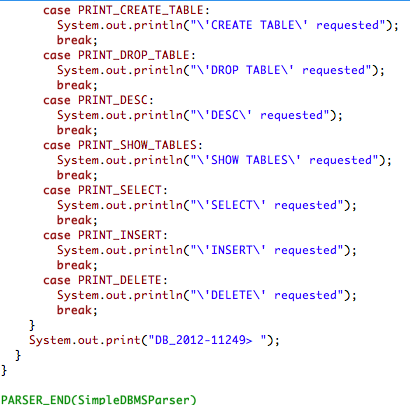
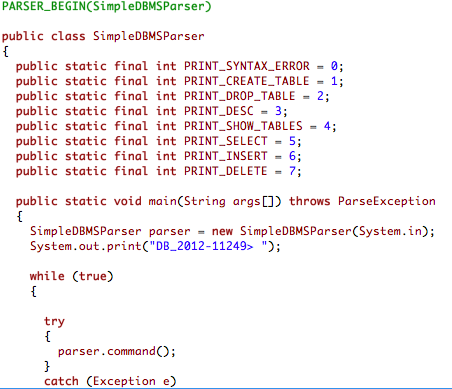
PARSER\_BEGIN~PARSER\_END에는 parser의 main 함수가 정의되어 있다. 우리의 Parser에서는 Parser의 시작과 명령프롬프트, 콘솔출력을 담당한다.

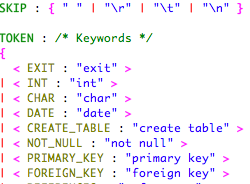
SKIP에서는 Parsing할 때 무시할 것을 설정해준다.

TOKEN에서 의미있는 문자열을 TOKEN으로 지정한다. 후에 TOKEN을 통하여 PARSER의 Grammar를 정의한다.

Parser functions(Grammar)에서 function의 Grammar와 action을 정의한다. type fncname(parameters) : { function body } { grammar } 의 형식으로 이루어져 있으며, 우리의 parser에서는 grammar부분을 주로 수정한다. 다음은 각 부분의 예시이다.







1. 구현한 내용에 대한 간략한 설명

기본 제공된 코드와 Grammar.docx를 참고하여 작성하였다. 제공된 코드 및 Grammar와 다른 부분을 위주로 설명하겠다. SimpleDBMSParser class에 각 Query에 대한 번호를 지정하였고, 그에 알맞게 콘솔에 결과를 출력하게 하였다. TOKEN에 필요하다고 생각되는 유의미한 문자열을 추가하였고 특히 <NON\_QUOTE\_SPECIAL\_CHARACTERS>에는 직접 ASCII코드표를 보고 특수문자를 추가하였다. (추후에 느낀 점에서 추가 설명) Grammar Definition을 보고 대부분의 Grammar를 작성하였다.

<TABLE NAME>과 <COLUMN NAME>의 정의가 둘다 <LEGAL IDENTIFIER>이기 때문에 우리의 Parser는 <LEGAL IDENTIFIER>의 입력이 들어올 때 이것이 <TABLE NAME>인지 <COLUMN NAME>인지 정확히 판단할 수 없다. 특히 <SELECTED COLUMN>에서는 처음 나오는 TOKEN이 <TABLE NAME>인지 <COLUMN NAME>인지 확정되지 않았기 때문에(둘 다 가능) LOOKAHEAD option을 통해서 해결해 주었다. LOOKAHEAD(2)로 통해 2개의 TOKEN을 확인하여 주어진 <LEGAL IDENTIFIER>에 대하여 <TABLE NAME>인지 <COLUMN NAME>인지 판단하게 하였다.

동일한 issue가 <PREDICATE>에도 있는데 역시나 [<TABLE NAME> <PERIOD>] <COLUMN NAME>에 대하여 이것이 <COMP OPERAND>에 들어가는지 <NULL PREDICATE>에 들어가는지 확정할 수 없다. 특히 이번에는 최대 3개의 TOKEN을 판단할 수 없으므로 LOOKAHEAD(4)로 해결하였다. <COMP OPERAND>와 <NULL PREDICATE>에서도 각각 <SELECTED COLUMN>과 같은 문제를 LOOKAHEAD(2)로 해결하였다.

<NULL OPERATION>에서 [not]을 (<NOT>)?로 처리할 경우 is + not + null로 인식하지 못하고 is + not null로 인식하여 오류가 생겼다. 의미상 is null <OR> is not null과 equivalent하므로 해당 Grammar로 대체하였다.

1. 가정한 것들

최소한 TOKEN은 개행문자 없이 제대로 들어온다고 가정하였다. 즉 create table토큰은 create + ‘\n’ + table로 입력하는 일 등이 없다고 가정하였다.

아울러서 TOKEN과 TOKEN사이에는 적어도 1개 이상의 공백이 있다고 가정하였다. 즉 drop tableaaa;와 drop table aaa;는 equivalent하게 처리하도록 하였다. 만약 drop tableaaa;와 같은 Query가 들어왔을 때 에러로 처리하려고 한다면 dropTable()의 grammar에 <SPACE>를 추가하는 방식 등으로 해결할 수 있다.

쿼리의 끝에는 반드시 세미콜론(;)이 존재한다고 가정하였다.(요구사항과 동일)

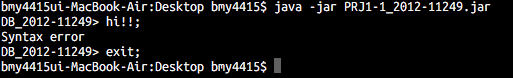
여러개의 Query가 들어오는 경우 순차적으로 처리하게 하였고, k번째 Query에서 오류가 있을 경우 (k-1)번째 Query까지만 처리하게 하였다.(요구사항과 동일)

Query에 대해서 결과를 출력할 때, 한번에 1개의 Query를 수행하는 경우는 프롬프트창이 나오지 않게 하였으며, 여러개의 Query를 수행할 경우, 첫번째 Query에는 프롬프트창을 출력하지 않고 두번째 Query부터 프롬프트창과 수행 결과가 나오게 하였다. (Presentation issue)

1. 컴파일과 실행방법

SimpleDBMSGrammar-2.jj 파일을 수정하면 같이 수정되는 파일 중 SimpleDBMSParser.java파일이 있다. 이곳에 main함수가 있으며, 기존의 java프로젝트와 마찬가지로 RUN버튼을 통해 컴파일 + 실행을 할 수 있다.

Executable jar파일로 실행하는 경우, 커맨드라인에서 java –jar PRJ1-1\_2012-11249.jar 명령어로 실행할 수 있다.



1. 프로젝트를 하면서 느낀 점

컴파일러와 Parser에 대한 지식이 전무한 상태에서 Grammar를 정의하고 Parser를 만드는것이 쉽지 않았다. 특히 문제상황에 대한 용어를 알지 못해서 해결방법을 찾기 어려웠다. 또한 javacc에 대한 지식도 전무하여 작동원리를 정확히 몰라서 추측으로 작동원리를 예측하는 것이 다소 어려웠다. 역설적으로 합리적인 추측을 한 경우(그것이 확실히 옳은지 모름에도 불구하고) 희열을 느꼈다.

Parser function의 경우 첫번째 중괄호에 java로 function의 body를 정의하고 두번째 중괄호에 Grammar를 정의하는것으로 보이는데(확실치 않음), 그것을 깨달은 것이 흥미로웠다. 2.에서 언급한 내용 중, <NON QUOTE SPECIAL CHARACTERS>에는 정규식을 이용하면 간단하게 할 수 있을 것이지만, 정규식에 대하여 정확히 알지 못하여 직접 ASCII코드표를 보고 작성하였다. 하지만 키보드만을 보고 작성하는 경우 빼먹을 수 있는 경우를 ASCII코드표를 통해 방지하였다.

Gramamr Definition에는 <SPACE>도 TOKEN으로 지정하라고 되어있는데(강요는 아니지만) 실제 SKIP에 “ ”가 있어서 충돌이 일어나는 경우가 있었다. 이런 경우 a)문서대로 하면 오류가 생긴다. b)문서대로 안할 수는 없다. 등의 이유로 당황하였다. 특히 javacc의 정확한 동작원리를 모르기 때문에 더욱 당황스러웠다.(결국에는 <SPACE>를 토큰으로 정의하지 않았음)

그럼에도 불구하고 Grammar Definition은 몹시도 소중했다! 또한 기본 제공된 코드도 javacc의 작동에 대해 추측할 때 아주 큰 도움이 됐다. 즉 앞서 말한 당황스러운 경우도 있었지만 아주 일시적인 당황일 뿐 굉장히 큰 도움이 됐다.(감사합니다 조교님!)