데이터 베이스 프로젝트 1-1 보고서

컴퓨터공학부

2012-11258

유근국

- 핵심 모듈과 알고리즘에 대한 설명

JavaCC를 이용하여 SQL문을 파싱할 수 있는 SQL 파서를 구현하였다. 정해진 문법에 맞게 PRJ1-1_2012-11258_src.jj파일에 파싱 룰을 정의하였으며 이를 컴파일하여 자동으로 java 파일들을 얻었다. PRJ1-1_2012-11258_src.jj 에 정의된 파서(class)의 이름은 레퍼런스 코드와 동일하게 SimpleDBMSParser라 하였다.

파서의 기본적인 알고리즘은 다음과 같다.

1. 사용자로부터 쿼리를 입력을 받는다.
2. 입력된 쿼리가 정해진 Syntax에 맞는지 판단한다.
3. Syntax에 맞지 않다면 Syntax error를 출력하고, 맞다면 해당 쿼리가 request 됐음을 출력한다.
4. 위 과정을 exit; 입력이 들어올때까지 반복한다.

또한 위 알고리즘에서 다음과 같은 제약을 만족해야 한다.

- SQL 구문이 여러 줄로 이루어진 경우에도 파싱할 수 있어야 한다
- 쿼리의 끝에는 항상 세미콜론(';')이 존재한다고 가정한다.
- 쿼리의 중간에 개행 문자('\r', '\n' 등)가 나타나더라도 쿼리를 종료하지 않아야 한다.

- 구현한 내용에 대한 간략한 설명

구현은 기본적으로 주어진 Grammar를 보며 이를 javaCC 문법에 맞게 번역하는 방식으로 진행하였다. Grammar 파일의 각 토큰들 중 "<QUERY LIST>::=(<QUERY> <SEMICOLON>)+" 와 같이 또 다른 토큰에 의해 정의되는 경우에는 jj 파일의 non-terminal symbol으로 정의하였으며 "exit", "create table" 와 같이 키워드 이거나 "<COMMA> ::= ," 와 같이 문법 구조상 말단이라 볼 수 있는 경우에는 jj 파일의 TOKEN 으로 직접 정의 하였다.

토큰을 정의함에 있어 키워드로 분류되는 토큰과 그 외의 토큰들을 분리하여 정의하였으며, 키워드로 분류되는 토큰을 우선적으로 정의하였다. 이를 통해 키워드 토큰들이 <LEGAL IDENTIFIER> 로 인식되지 않도록 하였다.

전역 옵션으로 “IGNORE_CASE = true;” 옵션을 주어 키워드들이 case insensitive 하도록 만들었다. 다만 옵션을 전역적으로 주었기 때문에 키워드 뿐만 아니라 identifier (table name, column name 등) 또한 case insensitive 하게 되는데 이는 과제 스펙에 명시되어있지 않으므로 고려하지 않았다.

쿼리의 공백을 무시하도록 하기위해 SKIP을 { " " | "\r" | "\t" | "\n" } 와 같이 정의 하였다. 이 때문에 TOKEN에 SPACE (“ “)를 < SPACE : " " >와 같이 정의하게 되면 에러가 발생하게 되는데 이로인해 SPACE의 경우 따로 토큰을 정의하지않고 직접 상수 스트링 “ “을 사용하였다.

```
comparisonPredicate()
| nullPredicate()
```

JavaCC에서 위 와 같은 경우 comparisonPredicate()와 nullPredicate()의 정의에 의해 “<LEGAL_IDENTIFIER> <PERIOD>” 와 같은 입력에 대해 choice conflict가 발생하게 된다. 따라서 LOOKAHEAD(4) 옵션을 주어 앞선 토큰까지 확인할 수 있도록 하여 choice conflict를 해결하였다. 아래와 같은경우도 마찬가지이다.

```
(
  tableName()
  < PERIOD >
)?
columnName()
```

<NULL OPERATION>의 정의에서 <NULL OPERATION> ::= **is** [**not**] **null** 를 JavaCC로 옮기면

```
< IS >
  (
    < NOT >
  )?
< NULL >
```

와 같은 꼴이 이상적이지만 이 경우 이미 정이된 <NOT_NULL> 토큰에 의해 “is not null” 이라는 입력에 대해 <IS><NOT><NULL> 이 아닌 <IS><NOT_NULL>로 처리되어 에러를 발생시키게 된다. 따라서 위의 문법을 아래와같이 바꾸어 정의하였다.

```
< IS >
(
  < NOT_NULL >
| < NULL >
)
```

- 가정한 것들

  - 키워드 중 “not null” , “create table” 과 같이 중간에 공백이 있고 여러 단어들로 구성된 키워드의 경우 구성하는 단어 각각 또한 키워드로 인식하도록 구현하였다. 즉, “create”, “table”, “not”, “null” 각각을 모두 키워드로 인식하도록 하였다.

  - “create table”, “foreign key” 등과 같이 공백을 포함한 키워드의 경우 “createtable”, “foreignkey” 와 같이 공백을 입력하지 않으면 에러로 판단하였다.

  - 쿼리의 중간에 개행문자 및 공백이 들어가도 상관없으나 키워드 중간을 끊는 개행문자 및 공백의 경우 에러로 간주된다. 특히 “primary key”, ”foreign key”, “not null” 의 경우 중간을 끊을 수 없다. 즉 다음과 같은 경우 에러로 간주한다.

  “primary \n key”, “cre \n ate table”, “not \n null”

  - 다음과 같이 키워드 뒤에 공백 없이 identifier가 나타나는 경우 에러라고 간주하였다. 즉, 다음과 같은 입력은 에러이다.

  “drop tableaccount;” , “select name asnamenew…..”

  - 위와 같이 특수한 경우가 아니면 쿼리의 다음 공백문자들 { " " | "\r" | "\t" | "\n" }은 무시된다.

  - 프롬프트(“DB_학번> ”)의 경우 입력을 받을때만 출력되도록 하였다. 즉 쿼리의 결과가 출력될때는 다음과같이 프롬프트가 출력되지 않고 결과만 출력되도록 하였다.

```
DB_2012-11258> show tables;
'SHOW TABLES' requested
```

그러나 이로 인해 한 줄에 여러 쿼리가 동시에 들어올 경우 다음과 같이 출력된다.

```
DB_2012-11258> show tables;show tables;show tables;
'SHOW TABLES' requested
DB_2012-11258> 'SHOW TABLES' requested
DB_2012-11258> 'SHOW TABLES' requested
```

- 컴파일과 실행 방법

  이클립스 상에서 프로젝트를 열고 PRJ1-1_2012-11258_src.jj 파일을 src로 등록 한 후 JavaCC를 통해 컴파일 한다. 그 후 생성된 Java파일을 통해 SimpleDBMSParser를 메인 class로 하여 실행한다.

  혹은 PRJ1-1_2012-11258.jar 파일을 바로 java –jar PRJ1-1_2012-11258.jar 명령을 통해 실행할 수 있다.

- 프로젝트를 하면서 느낀 점

  처음 과제를 받았을때는 어떻게 해야할지 전혀 감을 잡지 못했다. 태어나서 파서를 직접 만들어보는것도 처음이었고 무엇보다 JavaCC에 대해선 그 존재조차 들어본적이 없었기 때문에 많은 것이 낯설고 어렵게 느껴졌다. 그래도 주어진 뼈대코드와 그래머 문서를 보다 보니 조금씩 어떤식으로 파서를 만들어나가야 할지 길이 보이기 시작했고 막상 하다보니 처음 걱정했던 것 보다 훨씬 어렵지 않게 구현해 낼 수 있었다.

  과제를 하면서 가장 많은 시간을 지체했던 부분은 아무래도 LOOKAHEAD 가 필요한 부분이었다. JavaCC를 처음 다루다보니 우선 에러가 떠도 그게 LOOKAHEAD 옵션을 안 주어서 그런것인지 한번에 알아채기가 힘들었고 또 애초에 LOOKAHEAD 라는것이 있는지도 몰랐기 때문에 이것을 찾아내는것도 힘들었다. 그리고 아주 사소한 부분이지만 토큰으로 “\”를 정의할 때 “\\”로 정의해야 한다는 것을 알기까지도 꽤 오랜시간이 걸렸었다.

  전반적으로 처음 접해보는 과제였기 때문에 어렵기는 했지만 그래도 막상 해보니 생각보다는 간단하게 구현할 수 있었다. 또 그 과정에서 무언가 새로운 것을 알아가는 기분이 썩 나쁘지 않았다. 이번 파서를 만드는 과제를 잘 끝낸 만큼 다음 직접적인 DBMS 구현 과제들도 잘 끝낼 수 있었으면 좋겠다.