**Project 1 보고서**

2017-19428 서준원

1. 프로젝트 개요

이 프로젝트는 SQL의 기본적인 기능들을 수행할 수 있는 간단한 버전의 DBMS를 만들기 위한 밑작업이다. Parsing Programming Language인 JAVACC를 이용하여 SQL 쿼리들을 파하는 파서를 만들었다.

SQL의 기본적인 기능들을 구현하기 위해서 쿼리를 정확히 파싱할 필요가 있다. 정규 표현식으로 정의된 쿼리들을 파싱하도록 구현하였다. 대부분의 작업은 JAVA CC를 이용하여 Grammar를 그대로 옮기는 작업으로 이루어졌다. Grammar에서 Ambiguous한 부분들을 없애기 위한 크고작은 이슈들이 존재했다. Java CC의 구동 원리를 이해하고 이 언어에 익숙해지며, SQL 문법에 익숙해질 수 있었다.

1. 구현 내용

기본적으로 레퍼런스 코드 위에 구현하였다.

* 상수를 지정하여 쿼리마다 알맞은 출력을 할 수 있도록 구현하였다. Query 함수에서 해당 상수를 리턴값으로 지정했다.
* 토큰을 설정하였다. 토큰은 우선순위를 가지게 하기 위해 두 가지로 구분하여 만들었다. 키워드(예약어)로 지정된 토큰을 먼저 정의하여, table name, column name 등으로 인식되지 않도록 했다. 또한, 레퍼런스 코드 위의 not null을 분리하여 추후 오류가 없도록 했다. 다음 토큰들은 숫자, 특수 캐릭터 등을 정의하기 위해 사용했다.
* 코드는 크게 command -> queryList -> query 순으로 이어지며, exit;을 통해 프로그램이 종료되도록 설정했다. Syntax error의 경우에는 에러 핸들링을 하도록 main에서 설정했는데, 이 경우 sql 프로그램이 재시작되게 된다.
* 구현한 query는 다음과 같다.
  + CREATE TABLE / DROP TABLE / DESC / SHOW TABLES / SELECT / INSERT/ DELETE

각각 알맞은 키워드와 문법을 사용하여 정규식의 형태로 Javacc를 사용해 구현하였다.

* 구분들은 세미콜론으로 구분된다.
* 탭, 줄바꿈, 공백 등은 무시된다. 다만 줄바꿈 시 Syntax가 잘못된 것이 있으면 돌아가지 못하고 Syntax error로 처리된다. Input을 처리하는 과정 때문이다.
* Case Insensitive하다. 이는 맨 위의 options를 통해 구현됐다. 또한 여기서 디버깅 옵션을 사용하여 어떤 값이 들어갔는지 확인해볼 수 있다.

1. 추가적 가정

프롬포트 관련된 이슈가 존재했다. 이 프로그램은 엔터를 입력해도 command가 끝나는 것이 아닌 querylist 함수가 새로운 query를 기다리도록 설계되어있다. 하지만 프롬포트는 command가 끝나고 새로운 input을 기다릴 때 출력되어야 한다. 또한 결과를 출력할 때도 나와야 한다. 이를 구분할 수 없기 때문에 스펙 대로 출력하는 것이 불가능했다. 따라서 프롬포트를 만들지 않았다.

설계된 대로 한다면 첫 출력 시에만 프롬포트가 출력이 안된다. 통일성을 위해 모든 출력을 제거했다. 다만 쉬운 사용을 위해는 출력이 필요할 것이다.

1. 특별한 케이스

파서가 정규식을 올바로 이해하기 위해 LOOKAHEAD 설정을 바꿔줄 필요가 있었다. Default로는 LOOKAHEAD는 1로 설정이 되어 있다. 몇 개의 토큰을 보고 여러 개 중 선택을 할 지를 지정해주는 것이다. 다른 토큰이지만, 중복되는 정의가 있을 때 Lookahead를 설정해줘서 올바른 토큰으로 할당되도록 해주었다.

아래는 Lookahed 설정을 바꿔준 특별한 경우이다.

* Selectcolumn

<SELECTEDCOLUMN>::=[<TABLENAME><PERIOD>] <COLUMNNAME>[as <COLUMNNAME>]

Lookahead가 1일 경우는 tablename이 옵션으로 들어왔을 때, 이것이 column name인지 table name인지 알 수 없다. 따라서 Lookahead를 2로 설정했다. 필수적인 설명을 위해 코드의 일정 부분을 추가했다.

**void** selectedColumn() :

**{}**

**{**

**(**

**LOOKAHEAD(**2**)**

tableName()

< PERIOD >

**)?**

columnName()

**(**

< AS >

columnName()

**)?**

**}**

* Predicate와 null predicate

<PREDICATE>::=<COMPARISONPREDICATE>|<NULLPREDICTE>

<COMPARISONPREDICATE>::=<COMPOPERAND><COMPOP><COMPOPERAND>

<COMPOPERAND>::=<COMPARABLEVALUE> **|[<TABLENAME><PERIOD>] <COLUMNNAME>**

<NULLPREDICATE>::=[**<TABLENAME><PERIOD>] <COLUMNNAME**><NULLOPERATION>

여기서 comparison predicate와 null predicate는 강조한 부분과 같이 둘 다 <LEGAL\_IDENTIFIER > "." < LEGAL\_IDENTIFIER >의 형태로 시작이 가능하다. 따라서 최소 4개의 토큰을 봐주어야 한다. 따라서 아래와 같이 4개의 토큰을 보도록 구현했다.

**void** predicate() :

**{}**

**{**

**LOOKAHEAD(**4**)**

comparisonPredicate()

**|**

nullPredicate()

**}**

* Comp operand 와 null predicate도 selectcolumn과 마찬가지로 ambiguity가 존재한다. 같은 이유로 2개의 토큰을 보도록 지정했다. 위와 같은 형태로 구현되어 있어 소스코드는 첨부하지 않았다.