데이터 베이스 Poject1-1 보고서

2019-14355 배문성

1. 구현

1) SQL grammar : grammar.lark file

본 과제에서는 grammar.lark 파일에 EBNF 형식을 이용해서 create table 과 select 를 제외한 나머지 구문을 정의해야했다. 새로 정의해야하는 query 를 각각의 format 에 따라서 분류하면 다음과 같이 분류할 수 있다.

1. <query_symbol> := <query_name>

해당 format 은 어떤 argument 도 없이 독자적으로 기능하는 query 로 본 과제에서는 show tables 가 유일하게 해당된다.

2. <query_symbol> := <query_name> <table_name>

table_name argument 만을 필요로 하는 query 이다. 해당되는 query 에는 drop, explain, describe, desc 가 있다. 각각의 query_name 의 경우에는 새로운 token 을 선언해 주었고, table_name 의 경우에는 CREATE TABLE 에서 정의되어있기 때문에 해당 token 을 이용하였다.

3. <query_symbol> := <query_name> FROM <table_name> [where_clause]

Delete query 에 해당되는 format 으로, table_name 외에도 where_clause 가 선택사항으로서 존재하는 query 이다. 마찬가지로 where_clause 에 대한 정의는 SELECT 문에서 정의된 token 을 그대로 사용하였다.

4. <query_symbol> := <query_name> INTO <table_name> [column_list] VALUES values_list

Insert 에 해당하는 query 로서 column_list 와 삽입할 value 에 대한 list 를 추가로 요구하는 format 이다. column_list 는 CREATE TABLE 의 token 을 사용하였으나, value_list 의 경우에는 마땅한 token 이 없어서 column_list 의 문법을 참조하여 comparable_value 에 대한 list 를 작성하였다. 이때, 주어진 조건에 따라 value 는 INT, STR, DATE 만을 받을 수 있으므로, comparable_value 는 사용할 수 있는 token 이다.

5. <query_symbol> := <query_name> <table_name> SET <column_name> ‘=’i comparable_value [where_clause]

업데이트에 해당하는 query 이다. definition 에 의거하여 다음과 같은 format 을 사용하였는데, 동등연산에 해당하는 '='를 comp_op 와 구분할 필요가 있었다. lark 의 경우에는 token 을 이용하여 분류를 하는 과정에서 적은 sub-token 으로 이루어지는 token 에 대해서 우선적으로 대입을 하기 때문에 '='만을 담당하는 새로운 token 인 EQUAL 을 선언하여 사용하였다.

2) Python code

Python code 의 경우에는 크게 3 부분으로 나눠져있다. 먼저 첫번째 section 은 MyTransformer 로, 해당 부분에서는 입력받은 query 에 대한 각각의 tree 상의 node 에 대해서 해당 부분을 거칠때, 적용할 수 있는 함수를 구현하는 부분이다. 1-1 에서는 어떤 definition 에 부합하는 input 은 받아들이므로, 이 부분에서는 어떤 query type 이 입력되었는지 출력하는 부분만을 구현하였다. 이를 위해서 각각의 query 에 대해서 함수를 선언하고 해당 query 가 입력되었을 때, self.query_type 에 출력할 query 의 이름을 저장하는 방식으로 구현하였다.

두번째 section 은 input 을 받는 get_input 부분이다. input 의 경우에는 문장의 마지막이 세미콜론으로 끝날때 까지 입력을 받아야 하며, 첫번째 열에만 PROMPT PREFIX 를 출력해야한다. 또한 입력을 받은 이후에는 받은 입력의 whitespace('\n', '\r', '\t', ' ')를 전부 제거해 주었다. 그렇게 저장된 input 은 ';'를 기준으로 split 하여 list 로서 저장하였다.

마지막 section 은 이렇게 받은 input_list 를 lark 와 MyTransformer 를 이용하여 parsing 하는 과정이다. 이 과정에서는 program_end 라는 trigger 를 사용하여 해당 trigger 가 false 가 될때까지 loop 를 돌린다. 해당 trigger 는 주어진 input 에서 exit command 가 parsing 되었을 때 발동되며, main loop 를 종료시킨다. loop 안에서는 주어진 input_list 에 대해서 parsing 을 수행한다. 이때, 만일 parsing 과정에서 문제가 발생하면 Syntax error 를 출력하고 이후의 input 에 대한 parsing 을 생략한다. 정상적으로 parsing 이 수행되었다면 transfomer 로 부터 받은 query_type 과 REQUESTED_SURFIX 를 출력하고 다음 parsing 을 수행한다.

2. 미구현 사항

문서에 지칭된 SQL 쿼리에 대해서는 모두 구현을 했지만, 실제 SQL 에는 존재하지만 문서상의 definition 으로는 불가능한 부분에 대해서는 구현하지 못한 부분이 있다.

1) Sub query

실재 SQL 에서는 다양한 subquery 구문을 이용하여 유연한 쿼리를 할 수 있지만 본 과제에서는 해당 부분을 구현하지 못했다. 다만 select 구문에 대한 문법이 존재하기 때문에 다른 query 에 대해서 []와 같은 방식으로 명시해주면 구현할 수 있는 부분이라고 생각한다.

2) 복잡한 insert, update 문

SQL 에서는 상당히 유연한 방식으로 insert 구문을 수행할 수 있지만 본 과제에서는 직접적인 데이터 입력에 대한 insert 문에 대한 문법만을 적용해야했다.

그 외에도 HAVING 이나 GROUPBY 와 같은 다양한 구문과 statistic 함수들에 대한 구현 또한 되어있지 않다.