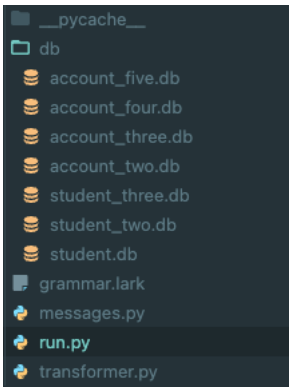
22-f Database Project 1-2: Implementing DDL

조경·지역시스템공학부
2017-19651 이혜민

1. 핵심 모듈과 알고리즘

- 알고리즘

Schema 를 저장하기 위해서 각 table 마다 db 파일을 하나 만들고, 각 attribute 의 정보를 파일에 저장하도록 했다. db 파일에 저장되는 정보는 총 column 수, primary key 리스트, 그리고 각 column 의 정보이다. 각 column 의 정보는 'column_[index]_name', 'column_[index]_type', 'column_[index]_is_null', 'column_[index]_is_primary_key', 'column_[index]_reference_table', 'column_[index]_reference_column' 이다. 총 column 수를 통해 각 column 의 index 의 range 를 파악하고 접근할 수 있다.



- **꼭 프로젝트 내 run.py 와 같은 디렉토리에 db 폴더가 있어야 동작합니다. 제출시에 폴더를 그대로 두었으니 따로 삭제하지 마시고 채점해주시길 부탁드립니다.**

Table 의 구분을 db 파일로 하기 때문에 create 쿼리에서 db 파일을 생성하고 drop, desc, show 쿼리의 경우 db 폴더 내에서 db 파일명들을 탐색한 후 이후 로직을 실행한다.

- 핵심 모듈

transformer.py: parse 된 결과 tree 를 처리하는 transformer 클래스로, 각 쿼리명에 해당하는 멤버 메서드들이 구현되어 있다.
messages.py: enum 클래스를 상속받아 각 에러 메세지를 관리한다.
run.py: 메인 로직이 구현되어 있다. Proj 1-1 에서 구현한 sql parser 를 포함한다.

2. 구현 내용

Sql parser 의 경우 proj 1-1 에서 구현하였으므로, 이번 프로젝트의 주 구현 내용은 transformer 를 상속받아 각 쿼리를 처리할 멤버 메서드를 구현하는 것이었다. 따라서 QueryTransformer 라는 클래스를 아래와 같이 선언하여 사용하였다.

```
class QueryTransformer(Transformer):

  def __init__(self):
    super()
    self.query_cols = 0
    self.column_names = []
    self.is_primary_key_set = False
```

해당 클래스는 query 에 포함된 column 개수, 각 column 의 이름들, primary key 명령어를 처리했는지 여부를 멤버 변수로 가지고 있으며 이는 create_table_qeury 메서드에서 db 에 저장하기 전 사용된다.

각 메서드의 구현은 아래와 같다. 전체 코드가 길고 복잡하여 구현한 로직에 대해 작성하였다.

2-1. create_table_query

- db 폴더를 순회하며 같은 이름의 테이블이 있는지 확인
- 겹치지 않으면 해당 이름의 새로운 db 파일을 생성
- Table element 들을 뽑아서 각 column 마다 column_definition 과 table_constraint_definition 의 경우를 나누어 처리
    - Column_definiion: 중복된 column 체크, char type 의 경우 Length 체크 후 해당 Column 의 name, type, nullable 정보를 db 에 저장
    - Table__constraint_definition:
        - Referential constraint: 참조받는 table 과 column 의 존재 체크, 참조하는 column 의 존재 체크, 참조하는 column 과 참조받는 Column 의 개수와 타입 체크, 참조받는 column 의 primary key 여부 체크, composite primary key 참조 경우 체크 후 모두 패스하면 reference table 과 reference column 을 db 에 저장
        - Primary key constraint: 존재 여부 확인 후 is_primary_key 라는 이름으로 column 정보를 db 에 저장, 이 column 의 is_nullable 정보도 “N”으로 다시 저장하여 업데이트 전체 primary key 의 리스트를 primary_keys 라는 이름으로 Db 에 저장하여 추후 composite primary key 처리를 할 수 있게 함
- 각 column 을 처리했으면 self.query_cols 를 column_num 이라는 key 로 db 에 저장

2-2. drop_table_query

- Table 이름이 db 폴더 내에 파일로 저장되어 있는지 확인 후 저장되어 있지 않으면 nosuchtable 에러 처리
- 저장되어 있으면 전체 table 파일을 순회하며 각 파일의 모든 Column 의 reference_table 을 확인. Reference table 의 이름과 drop 하려는 table 이름이 같은 경우 dropreferencetableerror 처리
- 외에는 db 폴더에서 해당 파일을 Remove 하여 Drop 처리

2-3. desc_query

- Table 이름이 db 폴더 내에 파일로 저장되어 있는지 확인 후 저장되어 있지 않으면 nosuchtable 에러 처리
- 저장되어 있으면 해당 파일 내에 저장된 모든 column 정보를 조회하여 프린트

2-4. show_tables_query

- db 폴더 내에 저장된 모든 파일명을 프린트

3. 가정한 것

- 자기 자신 table 을 참조하면 안 된다는 내용만 있고 어떠한 에러 메세지를 프린트할지 정해지지 않아서 존재하지 않는 테이블을 참조한다는 뜻의 ReferenceTableExistenceError 를 리턴하도록 구현하였다.
- Composite primary key 를 참조할 수 있는 경우는 composite primary key 를 한꺼번에 참조하는 경우만 정상적으로 스키마를 저장하도록 구현하였다. 즉 composite primary key 가 3 개일 경우 'foreign key (col1) references other_table(composite1)' 과 같은 element 가 3 개 연속 나타나는 경우가 아닌, `foreign key (col1, col2, col3) references other_table(composite1, composite2, composite3)' 와 같이 한꺼번에 참조하는 경우에만 가능하다.
- 여러 규칙을 동시에 위반하는 경우 최상위 에러 하나만 리턴하도록 구현하였다. 우선순위는 존재성이 가장 높고, 중복, type 에러 순이다.
- Column 선언의 Nullable 정보보다 primary key 선언이 우선시된다. 즉, Not null 지정하지 않은 Column 이 primary key 로 지정될 경우 not null 설정을 업데이트하도록 구현하였다.

4. 느낀 점

- Parsed 된 결과를 어떻게 처리해야 할 지 몰라 시간 소요가 있었는데, lark 의 docs 를 읽고 Transformer 에 대해 숙지하는 것이 구현의 핵심이었다고 느꼈다.
- 여러 에러 처리를 하는 과정에서 깔끔하지 못한 코드를 작성하여 리팩토링이 필요해 보인다.
- 구현에 급급하여 최적화를 하지 못한 부분이 있다. 특히 foreign key 참조 조건을 체크하는 경우 모든 table 의 모든 column 을 직접 db 에 접근해서 확인하고 있는데, 실제 DBMS 의 경우 이와 같은 경우도 최적화하여 구현해야 하겠다고 느꼈다.