데이터베이스 1-2 보고서

2019-14355 배문성

1 utils.py

BerkeleyDB의 경우에는 기본적으로 NoSQL database API이기 때문에 API 자체만으로는 본 과제에서 요구하는 바와 같이 Schema를 별도로 저장하는 기능은 존재하지 않는다. 따라서 schema를 저장하기 위한 별도의 class들이 필요하였고, 이를 utils.py에 작성하였다.

1.1 Schema

Schema class는 전체 Table들의 data를 담는 최상위 class이다. Schema의 경우에는 현재 존재하는 Table의 Table instances, 이름에 대한 정보를 담고 있고, Table을 추가하거나 삭제할때 생기는 여러 과정을 처리한다. 특히, 삭제시에 Table간의 reference관계를 조사하여 각각의 Table instance를 update하는 역할을 schema instance에서 수행한다. program이 시작될 때 pickle API를 이용하여 deserialized되며 schema에 변형이 생기는 query(create table, drop table)가 실행될 때마다 update한다.

1.2 Table

Table class의 경우에는 하나의 table에 대한 data를 보관하는 class로서 table name, column names, column instances, reference 관계, primary key, 그리고 현재 테이블 내부의 data tuple의 개수에 대한 정보를 담고 있다. 특히 reference 관계는 현재 table이 어떤 table의 어떤 column으로 부터 어떤 column을 reference되고 있는지를 저장하는 데이터로, table을 삭제할 수 있는지 판단하는 근거가 되기 때문에 schema class의 remove_table을 통해서 지속적으로 update되도록 관리하고 있다.

1.3 Column

Table의 각각의 column의 data를 담고있는 class로 이름, 타입, 길이(CHAR일 경우), nullable, primary key 여부, foreign key 여부, reference table, reference column name에 대한 정보를 가지고 있다.

1.4 Error, ParsedData

해당 2개의 class의 경우는 transformer에서 parsing한 data를 run.py에 전달하기 위해서 사용한다. Error는 error type, error가 일어났는지 여부에 대한 data를 전달하기 위해서 사용하고, ParsedData의 경우에는 query작업에 필요한 데이터를 run.py에 전달하는데 사용된다.

2 transformer.py

1-1에서는 query type에 대한 parsing만을 진행하지만, 이번 과제의 경우에는 parsing data를 검사하고, 또 그렇게 생성된 data를 전송해야 하므로 해당 db의 schema를 instance로 받아서 사용한다. 데이터 전송에는 utils.py에서 작성한 Error와 ParsedData class를 이용한다.

2.1 create table

create table의 경우에는 각각 column definition과 key constraints 2개를 담당하는 하위 parsing으로 나뉜다. 따라서 최상위 query인 create_table_query에서는 table_name에 해당하는 data만을 parsing하고 세부적은 parsing작업은 하위 함수에서 처리한다. table_name의 경우에는 현재 해당 schema에 같은 이름을 가진 table이 있는지를 검사한다.

column_definition
column_definition의 경우에는 column의 이름, type, 길이, nullable에 대한 data를 parsing하는 작업으로, column 이름이 겹치는지 여부, type에 따라서 valid한 길이 data의 parsing여부, 마지막으로는 not null query의 존재 여부를 검사하여 parsing을 진행한다.

primary_key_constraint
primary_key_constraint의 경우에는 DuplicatePrimaryKeyDefError만이 발생가능한 오류이므로, 현재 transfomer의 ParsedData에 primary key가 이미 입력되었는지 여부를 통해서 중복되었는지를 검사한다.

foreign_key_constraint
foreign_definition의 경우에는 가장 많은 예외 처리를 해야하는 query였다. reference table, column의 존재 여부, primary key 여부, composite primary의 경우 처리, type 비교등 수많은 조건을 확인하여 처리하였다. 검사하는 항목이 많았기 때문에 검사 순서에 따라서 Error 출력 메세지가 여러가지 경우의 수를 띌수 있었다. 이에 대해서 보다 자세히 설명하자면, 먼저 테이블의 존재 여부를 검사한다. 이때, 자기 자신에 해당하는 table의 경우에는 query처리가 되지 않았으므로 schmea에 등록이 되어있지 않기 때문에 지정할 수 없다. 그후, 해당 테이블의 primary key의 수와 입력받은 column의 수를 비교한다. 따라서 primary key가 없는 table을 참조하려고 하는 경우에는 ReferenceNonPrimaryKeyError가 발생한다.
그 이후 개별적인 column에 대해서 세부 spec을 검사하는 과정을 거치는데, 먼저 reference column의 존재 여부를 검사하고, 해당 column이 primary key인지, primary key가 composite이라면 그 순서에 부합하는지를 검사한다.(여기서 발생하는 Error의 경우 ReferenceNonPrimaryKeyError로 정의했다. 이는 Q&A 게시판을 확인했을 때, 본 과제에서는 definition에서 선언한 순서와 동일한 경우만을 허용한다는 답변을 얻었기 때문이다.) 마지막으로 foreign key가 되고자 하는 column의 존재 여부, 타입 체크, 그리고 중복 검사 순서로 error를 검사하는 과정이 실행된다.

2.2 drop table

drop table에서 발생 가능한 오류는 NoSuchTable과 DropReferencedTableError이 있으며, 각각의 Error의 경우에는 schema에서 이름이 존재하는지, table의 referenced data가 비어있는지를 검사함을 통해서 해결할 수 있다.

2.3 explain, describe, desc

해당 query들의 경우에는 NoSuchTable오류만이 발생 가능하므로, schema에서 이름을 검사하는 방식을 통해서 방지할 수 있다.

3 run.py

run.py의 경우에는 main loop의 경우에는 크게 바뀌지 않았으나, 기존의 경우에는 error가 오직 Syntax Error밖에 존재하지 않았기 때문에 별도의 error handler를 작성하지 않았다. 하지만 이번 과제에는 다양한 error type이 존재하므로 해당 error type에 따라서 알맞는 error message를 출력해주는 별도의 error handler를 통해서 출력하였다. 또한 실재 query를 처리하는 함수 역시 별도로 제작해야 했는데, 이는 DBMS의 atomicity를 위함이다. Transformer에서 즉각적으로 query를 처리하게되면, 처리된 query 내용을 discard함에 있어서 문제점이 발달할 것이라고 판단하여 run.py에서 완벽히 data parsing이 된 이후, error가 없음을 확인하고 진행하는 방식으로 구현하였다.

query를 처리하는 작업 중 가장 까다로웠던 부분은 create table 부분이었다. 이는 단순히 db파일을 생성하는 것 뿐만아니라 column이나 table간의 관계 역시 고려해야하기 때문에 다소 복잡한 작업이 필요했다. 이를 보다 유연하게 처리하기 위해서 $Column \rightarrow Table \rightarrow Schema$ 순서로 포함관계를 가지게 하여서 상위 부분에서 add, remove와 같은 query를 처리함으로서 다른 instance와의 연관 관계를 효율적으로 처리할 수 있도록 구조를 잡았다.

4 쿼리 과정 요약

1. Program 시작시 pickle package를 이용하여 schema instance를 deserialize한다.
2. main loop에서 input을 받고 이를 ';'단위로 split하여 준비한다.
3. Transformer에서 query 처리에 필요한 개별 data를 파싱함과 동시에 schema instance를 이용하여 각종 조건을 확인하고 error에 기록한다.
4. Transformer에서 parsing된 ParsedData와 Error data를 기반으로 error, query handling을 진행한다.
5. 1-4의 단계를 exit; query가 입력될 때까지 반복한다.
6. schema instance를 serialzied하여 저장한다.

5 Error 출력 관련 정리

1. Primary Key가 존재하지 않는 Table : ReferenceNonPrimaryKeyError
2. foreign key tuple과 primary key tuple의 길이가 일치하지 않는 경우 : ReferenceNonPrimaryKeyError
3. primary key의 definition에서 사용한 key 순서와 다른 순서를 사용한 경우 : ReferenceNonPrimaryKeyError