프로젝트 1-2

2014-15703 송화평

1. 개발 환경

본 프로젝트는 MacOS Catalina 10.15.4 운영체제가 탑재된 컴퓨터에서 Eclipse IDE 4.15.0, JRE 1.8.0\_241-b07 버전을 사용하여 개발되었다.

1. 구현한 내용

프로젝트 1-1에서 파서를 구현하였으므로, 이번 프로젝트에서는 SQL의 가장 기초적인 DDL 관련 구문인 create table, drop table, desc, show tables가 작동하도록 하였다. 특히 이 구현이 메모리 위에서만 머무르지 않고 실제 디스크에 저장되거나 디스크에서 불러올 수 있도록, 즉 기초적이나마 하나의 완전한 DDL 구문으로서 작동할 수 있도록 하는 것이 목표였다. 이를 위하여 Oracle 사의 Berkeley DB API를 사용하였다.

1. 핵심 모듈과 알고리즘

2.1. 핵심 모듈

DDL 구문 구현을 위하여 Java의 클래스를 사용하였다. Table, Attribute, ForeignKey, TableElementList, TableColumn, TableConstraint 등 총 여섯 개의 클래스를 구현하였고, storeTableToDB(), loadTableFromDB(), makeTable(), existTable() 등 총 네 개의 함수를 구현하여 사용하였다.

각 클래스는 그 멤버 변수로서 여러 가지 데이터를 가지는데, 이 때 동일한 자료형의 데이터가 여러 개 필요한 경우에는 java의 내장 자료구조 ArrayList를 사용하였다. 예를 들어 Table 클래스는 멤버 변수로 다음과 같은 데이터를 가진다.

String tableName;

ArrayList<Attribute> attributes;

ArrayList<ForeignKey> foreignKey;

ArrayList<String> primaryKey;

int referenceCount;

즉 테이블의 이름으로서 tableName이라는 변수를, 각 attribute를 나타내는 클래스의 집합으로서 attributes를, 각 foreign key를 나타내는 클래스의 집합으로서 foreignKey를, primary key의 집합을 나타내기 위해 primaryKey를, 그리고 이 테이블을 foreign key로서 가리키고 있는 테이블의 숫자를 나타내기 위해 referenceCount를 가지고 있도록 구현하였다. Table 클래스는 그 메소드로서 단일 attribute를 추가하는 addAttribute(), primary key의 리스트에 있는 모든 primary key를 가져오는 addPrimaryKeys(), 단일 foreign key를 추가하는 addForeignKey(), referenceCount를 증가시키거나 감소시키는 incrementReferenceCount()와 decrementReferenceCount()를 가지고 있으며, 그 외에도 각 변수들에 대한 getter와 setter를 가지고 있다. 다른 클래스도 이와 유사한 형태로 구성되어 있다.

storeTableToDB() 함수는 Table 인스턴스를 받아 이를 DB에 저장한다. loadTableFromDB() 함수는 이와 정 반대인데, 원하는 table의 이름을 받아 DB에서 해당하는 Table 인스턴스를 불러와 반환해주는 역할을 한다. makeTable() 함수는 table의 이름과, 그 table이 가져야 할 attribute와 constraint의 목록을 파싱 로직으로부터 가져와 Table로서 구현해낸 후, 이를 반환한다. 마지막으로 existTable() 함수는 단순히 loadTableFromDB() 함수를 감싸 인자로 받아온 table의 이름에 해당하는 Table 인스턴스가 실제 DB에 존재하는지 여부를 반환한다.

또한 Berkeley DB API를 통해 db/ 디렉토리에 테이블을 key-value 형태로 저장하였다. 이를 serialized하게 저장하고 불러올 수 있도록 하기 위해 모든 클래스는 java.io.Serializable을 구현하도록 하였고, Berkeley DB의 SerialBinding을 통해 objectToEntry() 메소드를 사용하여 객체에서 DB 엔트리로 serialize하고, 다시 entryToObject() 메소드를 사용하여 DB 엔트리에서 객체로 deserialize할 수 있게 하였다. 이 때 새 테이블을 정의하여 DB에 저장하고자 할 때는 Cursor 객체를 사용하여 DB에 저장하도록 하였고, 기존에 있는 테이블을 업데이트하고자 할 때는 Database 클래스의 delete() 메소드를 호출하여 DB에서 지우고 업데이트 된 테이블을 serialize하여 put() 메소드를 통해 DB에 다시 넣는 방식을 사용하였다. 이 때 DB에 테이블은 key-value 형식으로 저장하는데, key는 테이블의 이름을 getBytes() 메소드로 serialize한 것, value는 테이블을 serialize한 것을 DatabaseEntry 클래스로 변환하여 저장한다.

* 1. 알고리즘

가장 기본적으로 파싱 로직에서 토큰을 받아오는 부분부터 본격적인 구현이 시작된다. 토큰을 받아 이를 문자열로서 변환해 오는 것은 어렵지 않게 구현하였으나, table element list, 즉 table의 attribute와 constraint를 받아 오는 로직을 구현하기가 쉽지 않았다. 이는 tableElementList()에서 새로 TableElementList 객체를 생성하여, 이 인스턴스를 모든 하위 파싱 분기마다 집어넣어, 그 토큰들을 받아오는 방식으로 구현하였다. 이렇게 파싱받아온 TableElementList 객체와 tableName 문자열을 makeTable() 함수에 패러미터로 투입하여 반환되는 테이블을 DB에 넣는 작업을 하는 것이 createTableQuery()이다. makeTable() 함수는 이번 프로젝트에서 구현하기 가장 어려웠고, 가장 방대한 부분이었으므로 이 장의 마지막에 설명하기로 한다.

dropTableQuery()로 SQL 구문이 파싱되면, 파싱받아온 테이블 이름을 loadTableFromDB() 함수에 인자로 넣어 DB에서 테이블을 꺼내온다. 만약 이 테이블의 reference count, 즉 이 테이블을 foreign key로서 참조하는 테이블의 수가 0보다 크다면 아직 이 테이블을 참조하는 다른 테이블이 있다는 의미이므로 그 즉시 drop table을 중단한다. 만약 이 테이블의 reference count가 0이라면, 반대로 이 테이블이 참조하는 다른 테이블을 DB에서 불러와 decrementReferenceCount() 메소드를 호출하여 reference count를 감소시킨다. 이후 불러온 다른 테이블을 DB에서 잠시 지우고, 다시 serialize하여 DB에 저장하는 방법으로 이 값을 업데이트한다.

descQuery()로 SQL 구문이 파싱된 경우, 테이블 이름을 loadTableFromDB() 함수를 통해 테이블 객체를 받아온다. 이 테이블 객체의 멤버 변수를 얻어와 그 자료형과 nullable 여부, PK/FK 여부를 검사하여, 문자열 포맷팅을 이용해 터미널에 뿌려준다.

showTablesQuery()는 가장 간단하게 구현하여, DB에 존재하는 모든 테이블을 순차적으로 불러와 단순히 그 이름을 나열해준다. 만약 DB에 저장된 테이블이 없다면, There is no table문자열만 터미널에 띄운다.

makeTable() 함수는 전술했듯 이번 구현에서 가장 까다로운 부분이었다. 모든 경우의 수를 고려하여 테이블 객체에 attribute를 넣어주어야 했고, 그것이 constraint를 위반하지 않는 올바른 객체인지 아닌지도 따졌어야 했기 때문이다. 함수에 진입하면 먼저 DB에 있는 key-value pair의 key값을 조사하여 테이블의 이름과 같은 것이 있는지 확인한다. 만약 그렇다면 중복된 테이블을 허용해서는 안되기 때문에 예외를 발생시킨다. 이어 파싱 로직으로부터 파싱받아온 attribute를 검사하여, char length 조건에 위배되지는 않는지, 중복된 attribute가 있는지 등을 검사하고 테이블에 추가한다.

또한 constraint의 경우에도, PK와 FK의 경우를 구분하여 PK의 경우에는 PK가 두 개 이상 정의되어 있는 지, PK로 사용하겠다고 서술된 attribute가 실제 테이블에 존재하는 지 등을 따져본 후 테이블에 PK로서 추가한다. 이 때 PK로 선언된 attribute는 nullable하지 않으므로 이 역시 처리해준다. FK로 선언된 constraint를 만났다면, 레퍼런스하고 있는 attribute가 실제 테이블에 있는지, 레퍼런스되는 테이블이 DB에 있는지, 레퍼런스되고 있는 attribute가 레퍼런스되는 테이블에 있는지, 그리고 그것들이 레퍼런스되는 테이블의 PK인지 등을 철저하게 조사하여 검증한다. 또한 레퍼런스하는 attribute는 레퍼런스되는 테이블의 PK를 부족함 없이 완전하게 나타낼 수 있어야 하고, attribute의 자료형도 맞아야 하며, 그 숫자도 맞아야 한다. 이 검증을 끝내면 이제야 실제로 ForeignKey 객체의 ArrayList를 정의하며, 이를 테이블에 foreign key로서 추가해준다. 마지막으로 레퍼런스되는 테이블의 reference count를 증가시켜주고, 이 변화를 DB에 반영해준 후 테이블을 반환해줌으로써 makeTable() 함수를 종료한다.

1. 가정한 것들

제시된 상세를 충실히 따라 구현한 만큼 다른 사항을 딱히 가정하지는 않았다. 다만 제공받은 프로젝트 1-1 레퍼런스 코드를 토대로 작성하였는데, 이 코드에서는 파싱한 후의 작업을 세미콜론을 파싱한 후에 하는 것이 아니라 세미콜론 직전까지의 구문을 파싱하고 작업을 진행한 후에야 세미콜론을 파싱하도록 되어 있었다. 따라서, 예를 들어 show tables;라고 명령했을 경우 파서는 첫 번째로 show를 파싱하고, 두 번째로 tables를 파싱한 후 showTablesQuery()를 통해 이에 해당하는 동작을 실행하고 나서야 세미콜론 ;를 파싱하게 된다. 다만 이것이 의도된 점인지 아닌지 알지 못하고, 설령 의도되지 않았더라도 구현 상세에 적혀있지 않아 수정해야 할 당위를 느끼지 못하여 그대로 두었다.

1. 컴파일과 실행 방법

Eclipse IDE에서 javacc로 컴파일한 후 실행해도 되고, 첨부한 PRJ1-2\_2014-15703.jar 파일이 있는 디렉토리에 db/ 폴더를 생성한 뒤 터미널에서 java -jar PRJ1-2\_2014-15703.jar 명령어를 이용하여 실행할 수도 있다.

1. 느낀 점

프로젝트 1-1이 DBMS가 아니라 컴파일러를 작성하는 것 같았다면 (실제로도 반쯤 맞는 말이기는 하지만), 이번 프로젝트 1-2는 좀 더 java스러운 프로그래밍을 할 수 있었다. 구현 예시가 주어졌던 프로젝트 1-1과 달리 처음부터 아무 것도 주어지지 않아 처음에는 많이 힘들었지만, 모멘텀을 얻고 난 뒤에는 힘들다기보다는 재미있다는 느낌을 받았다. 메모리 참조 오류없이 한 번에 바로 실행되는 것을 보아 (여러 semantic 버그가 산적하긴 했지만) 프로그래밍 실력이 늘었다는 생각을 하기도 했고, 반대로 학부 수준의 쉬운 과제 구현일텐데도 이렇게 만만치 않은 것을 보아, 아직 멀었다는 생각 또한 들었다. 전부 다 완성하고 나서야 드는 생각이지만 같은 일을 해주는 코드를 모아 함수로 정의하거나, 전체 작업을 총괄하는 클래스를 정의하는 등 여러 리팩토링 방법이 있을 텐데 시간에 쫓기고 귀찮음에 시달려 끝내 하지 못한 것이 아쉽다. 좋은 프로그래밍에 대해서도 공부해보아야 할 것 같다.