프로젝트 1-3

2014-15703 송화평

0. 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 1-2에서 구현한 create table, drop table, desc, show tables의 기초적인 DDL 구문에 더하여, insert into, delete from, select의 기초적인 DML 구문을 구현하였다. 이번 구현부터 실제 레코드가 디스크에 저장되게 되며, delete from 명령을 통해 조건을 걸어 레코드를 지우거나 이를 select 명령을 통해 화면에 표시할 수 있다.

2. 핵심 모듈과 알고리즘

프로젝트 1-2에서 DDL 구문 구현을 위해 Java의 클래스를 사용하였다. 프로젝트 1-3에서는 DDL 구문으로 정의한 테이블의 인스턴스를 만들고 관리하기 위해 클래스 중 Table 클래스의 인스턴스 변수로서 테이블의 인스턴스를 가지고 있게 하였다. 구체적으로, instance라는 이름의 ArrayList<ArrayList<String>> 타입의 변수를 설정하였다. 이 변수는 setInstance() 메서드로 설정하고, getInstance() 메서드로 참조할 수 있으며 테이블 인스턴스에 레코드를 추가하고자 할 때는 addInstance() 메서드를 호출할 수 있도록 하였다. 또한 checkPKConstraint() 메서드와 checkFKConstraint()는 해당 테이블에 추가하려는 레코드를 받아 그 레코드가 PK constraint와 FK constraint를 위반하지는 않는지 검사해주도록 구현하였다.

또한 SimpleDBMSParser 클래스 내에 여러 함수를 정의하였다. 예를 들어 product() 함수는 Table의 리스트를 받아 이 테이블의 인스턴스들에 대해 cartesian product한 결과를 ArrayList<ArrayList<String>>의 타입으로 돌려준다. 또한 compare() 함수는 두 operand와 operator, 그리고 type을 받아 두 operand를 비교한 값을 0이나 1의 정수형으로 반환하며, columnNameCount() 함수는 각 인스턴스의 attribute를 설명하는 header 내에 tableName.columnName 형태의 레퍼런스가 몇 번 등장하는지 반환하는 함수이며, printSelected() 함수는 select query 시 터미널에 이를 올바로 출력해주는 역할을 한다.

레코드를 삽입하고자 할 때, insert into <tableName>, <columnValuePair>의 형식으로 쿼리가 들어오게 된다. tableName은 단순히 문자열의 형태로 받아오지만, columnValuePair의 경우에는 ArrayList<ArrayList<String>>의 타입을 가진 이중 리스트로 받아온다. 가장 바깥쪽의 리스트는 언제나 길이가 2이며, 0번째 원소로는 삽입할 테이블의 column 이름의 순서가 있는 리스트를 갖고, 1번째 원소로는 "type:value"와 같은 형태의 문자열의 리스트를 갖도록 구현하였다. 이를 파싱하면 해당 column의 값 뿐만 아니라 그 타입에 대한 정보도 알 수 있게 된다.

레코드를 삽입하기 전, insert into table (attr0, attr1, ..., attr(n-1)), values (val0, val1, ..., val(n-1))의 형태로 쿼리가 주어졌는지, 아니면 단순히 insert into table values (val0, va1, ..., val(n-1))의 형태로 쿼리가 주어졌는지 판단한다. 만약 후자라면 해당 테이블에 정의된 attribute의 순서대로 값들이 요청되었다고 판단하며, 전자라면 명시된 순서와 같은 순서대로 값들이 요청되었다고 판단한다. 두 경우 모두에 대해 size mismatch나 type mismatch 등의 수많은 validity check를 수행하

며, 만약 문제가 없다면 최종적으로 Foreign Key Constraint와 Primary Key Constraint를 위반하지는 않았는지 검사한다. 만약 두 경우 모두 위반하지 않는다면 비로소 실제 insert가 수행되게 된다.

PK constraint를 점검하는 과정은 primary key가 정의되었는지 아닌지에 따라 갈리는데, 만약 정의되었다면 기존에 존재하는 각 레코드들에 대해 PK를 구성하는 attribute만 추출하여 추가하고자 하는 레코드의 PK를 구성하는 attribute와 비교하여 만약 같다면 primary key duplication이므로 레코드를 추가하지 않도록 하였다. 만약 primary key가 정의되지 않았다면, 모든 attribute가 primary key를 이룬다고 판단하여 모든 attribute에 대해 비교를 수행하게 된다. FK constraint를 점검하는 과정도 비슷하여, 각 foreign key에 대해 해당 레코드가 레퍼런스하는 다른 테이블의 레코드가 실제로 존재하는지 판단한다. 이 때도 FK를 이루는 attribute만 추출하여 비교하도록 하였다.

select로 레코드를 가져올 때에는 product() 함수를 통해 명시된 테이블들에 cartesian product를 적용한다. 이는 productHelper() 함수를 통해 두 개의 테이블을 먼저 곱하고, 그 결과 값에 또 다시 productHelper() 함수를 적용함으로써 이루어지도록 하였다. productHelper() 함수는 두 테이블의 각 인스턴스의 레코드들에 대해, 그 레코드를 concatenate한 것을 계속해서 추가함으로써 간략히 구현하였다. 이후 where 절에서 조건을 비교하도록 할 때에는, booleanList라는 리스트를 정의하여 1을 true, 0을 false, -1을 unknown으로 인식하도록 구현함으로써 product 연산의 결과인 products의 각 레코드들에 대해 where 조건을 만족시키면 1, 아니면 0, unknown이면 -1을 대응시키도록 하였다. 모든 작업이 이루어진 후에는 대응되는 booleanList 값이 1인 레코드만 추출하여 printSelected()함수에 넣어 이를 출력하도록 하였다.

Delete로 레코드를 삭제할 때에는 select와 같은 방식으로 booleanList에 조건을 만족하는 레코드들에 대한 정보를 담고, 조건을 만족하는 레코드들에 대해서만 삭제하도록 구현하였다. 이 때 어떤 레코드를 삭제할 때 이 레코드를 foreign key로서 레퍼런스하는 다른 테이블의 레코드들에 대해, non-null인 attribute가 있다면 해당 레코드에 대한 삭제를 취소하고, 그렇지 않다면 해당 attribute를 null로 설정하도록 하였다. 이는 Table 클래스의 메소드인 checkRecordDeletable() 메서드와 nullifyReferencingRecord() 메서드에서 수행한다.

## 3. 가정한 것들

이번 구현에서도 가정한 것은 크게 없지만, select attribute as A;와 같이 rename이 select 문에서 이루어졌을 경우 이를 출력할 때 attribute 이름을 A로 바꾸는 것을 가정하였다.

## 4. 컴파일과 실행 방법

Eclipse IDE에서 javacc로 컴파일한 후 실행해도 되고, 첨부한 PRJ1-3_2014-15703.jar 파일이 있는 디렉토리에 db/ 폴더를 생성한 뒤 터미널에서 java -jar PRJ1-3_2014-15703.jar 명령어를 이용하여 실행할 수도 있다.

## 5. 느낀 점

다른 과제와 취업준비에 쫓겨 과제 제출이 늦어졌지만, 고생한 만큼의 보람이 있는 프로젝트였다. 물론 쉽지 않은 과제였고, 생각보다 아주 많은 시간을 필요로 하였으며, 3000줄에 달하는 큰 분량때문에 많이 힘들었지만 결과물에 자부심을 느낄 수 있었다. 다음 과제는 늦지 않게 제출할

수 있도록 시간관리를 잘 해야 할 것 같다.