2015 fall 4190.310-001 Database

Project 1-3: Implementing DML

2013-11395 김희훈

1. Introduction

이번 프로젝트에서는 프로젝트 1-1, 1-2에 이어 INSERT, DELETE, SELECT 구문에 대해 실제 동작을 하도록 SimpleDBMS를 개선하였다.

2. New & Modified Classes

DML 구문들의 동작을 위한 새로운 클래스들을 작성하게 되었다.

• Value

DML 구문들은 데이터를 직접 다루기 때문에, 값의 타입과 실제 값을 저장하는 Value 클래스를 작성하였다. NULL, INT, CHAR, DATE 타입을 지정할 수 있고, INT인 경우 int, CHAR, DATE의 경우 String으로 실제 값을 저장한다. 비교를 위해 equals, 출력을 위해 toString도 구현되어 있다. 스키마를 알고 있다면, List<Value> 형태로 레코드를 다룰 수 있다.

• RecordBinding

1-2에서 스키마의 저장을 위해 ColumnBinding을 작성했듯이, 레코드의 저장을 위해 RecordBinding을 작성하였다. 구조는 (타입 [실제값])+ 이며 데이터의 끝을 -1로 표시하였다.

• Records

레코드 여러개, 즉 테이블 인스턴스를 나타내는 클래스이다. Table은 테이블 스키마를 표현하기 때문에 별도로 구현하였다. List<List<Value>> 형태로 레코드를 저장한다. addRecord, remove, getRecord, size 등의 기본 함수가 구현되어 있고, integrity 체크를 위해 checkPrimary, checkForeign도 구현하였다.

• ExpTree

Where절의 Boolean Expression을 연산하기 위해 구현된 클래스이다. 내부에 각 문법 노드를 나타내는 서브클래스들을 가지고 있고, 이 서브클래스들은 interface로 모두 eval 함수를 가지고 있다. 루트 노드의 eval에 레코드를 넘겨주면 재귀적으로 연산하여 Value3 오브젝트를 리턴하는데, Value3는 true, false, unknown의 값을 나타내는 클래스다.

• Column & ColumnBinding

Column의 순서를 칼럼명의 알파벳 순서로 유지하고 있었는데, INSERT에서 칼럼을 명시하지 않고 튜플이 주어지거나, SELECT 결과를 출력할 때 등의 경우에서, CREATE TABLE 당시 스키마에서 나타난 칼럼의 순서가 필요하다는 것을 알게 되었다. 그래서 내부적으로 ord 변수와 getSortedColumns()를 구현하고, Berkeley DB에도 ord 값을 저장하였다.

• Foreign

이 클래스는 foreign key constraint를 지정할때, 다른 테이블의 특정 칼럼을 지칭하는 용도로 사용했었다. 그러나 DML 구문들에서 (테이블명, 칼럼명) 꼴의 레퍼런스가 빈번하게 사용되어 용도를 확장하게 되었다. 구현에는 거의 변함이 없다.

- **SchemaManager**

레코드를 저장하기 위해 records라는 이름의 Berkeley DB를 추가하고 insert, delete, get, update 기능을 구현하였다. 테이블 삭제시 레코드도 함께 삭제되어야 하므로 dropTable[s]의 구현도 수정되었고, open 및 close 하는 루틴도 추가되었다.

3. Implementation

- **INSERT**

insert는 Table.insert()에서 담당한다. 먼저 입력으로 들어온 대상 칼럼과 값 튜플을 보아서 존재하는 칼럼인지, 타입이 맞는지, not null인 칼럼에 null을 넣으려 하는지, char의 길이는 맞는지를 체크하고 삽입할 레코드를 구성한다. 그리고 대상 테이블과 이 테이블이 레퍼런싱하고 있는 테이블들의 레코드를 불러와 primary, foreign key constraint를 체크한다. 이상이 없다면 삽입한다.

- **DELETE**

delete는 Table.delete()에서 담당한다. 먼저 WHERE절 안에서 대상 테이블의 존재하는 칼럼만 사용하는지 확인한다. 그리고 ExpTree.eval()을 이용하여 조건을 만족하는 삭제될 후보 레코드들을 체크한다. 그리고 각 레코드마다 foreign key constraint를 체크하여 삭제가능한 레코드만 삭제한다.

- **SELECT**

SELECT 동작은 복잡하여 SelectHelper 클래스를 별도로 구현했다.

A. FROM절로부터 가능한 (테이블명, 칼럼명)을 모두 기록해둔다. 예를 들어, 테이블 T가 칼럼 x, y를 가지고 있는데 T as S로 입력 되었다면, 가능한 경우는 (null, x), (null, y), (T, x), (T, y), (S, x), (S, y)가 되겠다. 중복이 있을 경우 카운트하여 유일한지 아닌지 판단 할 수 있도록 한다.

B. SELECT절과 WHERE절에서 접근한 Foreign을 모두 찾아서, 위에서 기록해둔 것과 유일하게 일치하는지 확인한다. 일치하는게 없거나, 유일하지 않은 경우 잘못된 접근이다. 유일하게 일치한다면 (FROM절에서 몇번째 테이블인지, 테이블 안에서 몇번째 칼럼인지)의 정수쌍으로 기록해준다.

C. Nested Loop 방식으로 FROM절 테이블들의 Cartesian Product 레코드들을 순회한다. 테이블이 n개라면 n중 루프가 되므로 for문으로는 구현이 곤란해서, n-tuple로 각 테이블의 몇번째 레코드를 돌고 있는지 기록하여 순회하였다. 예를 들어, 레코드가 각각 n, m개인 테이블 2개가 FROM절에 있다면, (0, 0) ~ … ~ (0, m - 1) ~ (1, 0) ~ … ~ (n - 1, m - 1)순으로 순회한다. 순회하면서 ExpTree.eval()을 이용하여 조건을 만족하는 출력할 레코드의 n-tuple을 기록한다.

D. 출력한다.

4. Assumptions

- foreign key의 경우 1-2에서의 질답을 오해한 결과로 독립적으로 구현되였다. 즉, 이 구현에서는
  foreign key(x) references a(x), foreign key(y) references a(y)
  foreign key(x, y) references a(x, y)

이 두 가지가 완전히 동치다.

예를 들어, 레퍼런싱 하는 테이블에 (a, 2), (b, 1)이 있다면 일반 SQL 구현에서는 (a, 1)이 삽입되지 않지만, 이 구현에서는 foreign key를 독립적으로 체크하기 때문에 삽입 가능하다.

- INSERT에서 칼럼이 명시적으로 주어지지 않는 경우, 앞 칼럼부터 값을 삽입하며 부족한 값은 null을 넣는다. 값이 초과로 들어오면 InsertTypeMismatchError를 발생시켰다.
- INSERT에서 칼럼이 명시적으로 주어졌는데 값과 갯수가 일치하지 않는 경우 InsertTypeMismatchError를 발생시켰다.
- DELETE에서 삭제된 row가 없어도 '0 row(s) are deleted' 메시지는 출력된다.
- SELECT 출력에서 한 칼럼의 폭은 20으로 고정하였다. 초과하면 truncate 된다.
- WHERE 절의 비교에서 타입은 static하지 않고 dynamic하게 체크된다. 즉, x, y의 타입이 다르고 x = y 구문이 있더라도, x, y 중 하나가 계속 null이라면 오류는 발생하지 않는다.

5. 실행 방법

java -jar 옵션으로 runnable jar 파일을 실행하면 된다. 단, db 폴더가 jar 파일과 같은 폴더에 존재해야 한다.

6. 느낀 점

WHERE절의 식을 어떻게 계산할지 까다로웠다. interface와 class를 번갈아가면서 사용하여 트리형식으로 구현하였는데 깔끔하고 구현하기 재미있었다. 가장 잘 풀리지 않았던 구현은 Foreign을 통해 지칭한 테이블, 칼럼이 존재하고 ambiguous하지 않은 지 체크하는 것이었다. 클래스를 새로 만들 것을, 원래 다른 용도로 쓰이던 클래스를 끌어다가 강제로 써서 그런것 같다. 위 두가지 부분이 모두 필요했던 SELECT 동작을 구현하는게 가장 챌린징했다.