PRJ1-3 레포트

2016-12299

컴퓨터공학부 박용주

1. 핵심 코드 설명

1-2와 같이 하나의 table은 하나의 DB에 대응한다. 이 DB에는 table의 schema와 record가 함께 저장되며, 이는 key에 있는 특수한 string을 이용해 구분한다.

1-2에서 create에 대한 기다란 query는 arraylist를 이용해 받아왔지만, 구현이 어려웠기 때문에 1-3에서는 insert, delete, select에 대해 query를 하나의 기다란 string으로 받아온 후, 나중에 parsing하는 방법을 사용하였다. String은 " 또는 ""을 이용하여 parsing하는데, char에 quote string이 들어오지 못한다는 점을 이용한 것이다.

1번에서는 DB를 열고 필요한 데이터를 얻어오는 과정은 최소화하고 설명이 필요한 부분만 추가적으로 설명한다. 자세한 구현은 주석을 통해 확인 가능하다.

- INSERT

쿼리는 모두 한 줄의 string으로 받아온다. Insert into t(a,b) values (1,2); 일 경우 받아오는 쿼리는 "t"a"b"values"1"2 의 형태가 되어 맨 첫번째는 `table_name`, values 왼쪽은 `column_name`, values 오른쪽은 real_data가 된다. 이 string을 " 로 split하여 파싱한다.

그 다음 `table_name` 에 해당하는 DB를 open하여 :`columns`에 저장되어 있는 해당 table의 column이름들과 각각의 type을 가져온다.

그 후 column의 schema와 real_data를 하나씩 비교하며 column 수가 맞는지, type이 맞는지 비교한다. 이 때, null일 경우 :`not_null`을 이용해 해당 column에 null이 삽입 가능한지 확인한다.

다음으로는 다시 DB(table)을 열고 cursor를 이용해 이미 존재하는 pk가 있는지 확인한다.

Foreign key가 있을 경우 참조하는 table을 바로 열고, 그 table을 cursor로 스캔하여 대응하는 pk가 있는지 확인한다.

핵심적인 구현 방법은 기본적인 error체킹이 모두 끝난 후 insert operation을 진행한다는 것이다. 즉, 먼저 기본 schema와 column 존재여부, type 체킹, primary key가 있는 지 체크, 대응하는 참조 테이블의 schema와 맞아떨어지는 지를 먼저 체크한다.

여기까지 error가 없으면 비로소 insert를 실행하며 이 과정에서 'pk의 중복'이나 '대응하는 foreign key data가 부재' 등에 대한 record별 error를 체크하며 insert를 수행한다.

- DELETE

Delete는 insert와 달리 table_name""where""([조건1])"and/or"([조건2]) 의 더 복합한 형태의 string을 넘겨받는다. where앞은 table이름이고 where뒤는 where절이며 이것을 "" 로 split하여 구분한다. Schema와 type에 대한 체크는 insert와 비슷하며, delete의 핵심 코드는 where절의 false와 true를 계산하는 것이었다.

(([a=1] and [b=2]))or ([c=3]) 이라는 구문이 있다면 먼저 [ ] 을 이용하여 조건절을 하나씩 쪼갤 수 있다. 그렇게 하여 모든 조건절의 결과를 먼저 계산한다.

((true)and(false))or(true)의 형태로 바뀌면 먼저 (true)는 true로, (false)는 false로 변환한다. 그러면 (true and false) or true 의 형태가 되고, (true and false)는 (false)로 바꾸어 false or true로 또 변환한다. 그 후 false or true는 true이므로 해당 where절의 최종 결과는 true가 된다.

이처럼 where절의 조건을 하나씩 쪼개고, 처음에 얻어진 ((true)and(false))or(true) 이라는 기다란 string을 반복문을 이용해 점차 좁혀나가면서 true라는 결과를 얻는 것이 핵심이었다.

Delete의 두 번째 핵심은 참조하는 DB에 대한 문제이다. 먼저 FK_DB(참조 관계를 저장해둔 DB)를 이용해 현재 TABLE을 참조하는 모든 TBALE을 찾는다. 그리고 참조 테이블을 돌면서, 현재 delete 하려고 하는 record_data의 pk와 대응되는 (참조)data가 있는지 찾는다.

만약 있다면, 해당 column이 nullable한 지 보고, nullable하지 않다면 해당 record의 삭제를 취소한다. 만약 모든 참조 테이블에 대해 이런식으로 진행했을 때

1. 해당 record를 참조하고 있는 table이 없다.

2. 해당 record를 참조하고 있지만 nullable 하다.

위 2조건이 모두 해당하지 않을 경우 참조에 의한 에러로 해당 record의 삭제를 취소한다.

- SELECT

Select는 where절 분석이 delete와 거의 비슷해서 상대적으로 가장 적은 시간이 소요되었다. Select의 핵심은 from에서 as를 사용했을 경우와 여러 개의 table을 불러올 경우이다.

먼저 `real_table_name`과 `ref_table_name` 이름을 따로 만들어서 `ref_table_name` 에는 실제로 참조할 때 사용할 수 있는 이름만 저장해두었다. 가령 a as T 라고 사용했을 때 `real_table_name`에는 a를 `ref_table_name` 에는 T를 저장하고, 참조는 T로만 가능하게 하였다. 그 후, DATA가 필요할 때는 `real_table_name`에 있는 a를 이용하는 방식이다.

Column 배열도 1. `Column_name` 2. `ref_table_name.column_name` 으로 2가지를 저장하였다.

이 때, 어떠한 table이 ref_table에 2번 이상 나올 경우, 어떠한 column이 column배열에 2번 이

상 나올 경우 모호한 참조 에러로 처리한다.

From a, b, c와 같이 여러 개의 테이블을 참조할 때는 모든 테이블에 대해 Cartesian 곱을 해야한다. 먼저 첫 번째 a에 대한 record를 모두 arraylist에 저장한다. 그 후 b의 loop를 돌면서 arraylist의 각각의 원소마다 모든 b의 record를 append한다. 이렇게 만들어진 arraylist를 다시 c의 loop를 돌면서 모든 원소에 c의 모든 record를 append한다.

Loop가 끝나면 arraylist에는 a"b"c 순서로 append 된 커다란 배열이 만들어진다. 그 후 처음에 만든 select_schema_columns 라는 배열이 이 커다란 배열의 schema가 된다.

2. 구현하지 못한 것

기능은 모두 구현하였지만 1-2와 같이 코드의 리팩토링을 하지 못하였다. 시간에 쫓기다보니 재활용할 수 있는 코드를 생각하기보다 복사+붙여넣기를 하기에 급급하여 코드가 불필요하게 많이 길어진 것 같다.

3. 가정한 것들

- where절에서 타입검사를 할 경우 "비교 불가능한 타입에 대한 에러"는 record단위가 아니라 schema 단위로 실행할 것이다. (무결성, 안정성 보장)

- select from 절에서 a as T로 선언될 경우 a에 대한 참조는 T로만 가능할 것이다.

- select의 출력에서 column이름은 앞의 "table." 을 제외한 column 이름만 출력할 것이다.(스펙 예시 참조)

- column의 개수가 100개가 넘어가는 커다란 table은 test하지 않을 것이다. String 배열의 크기를 200으로 해두었는데, schema가 커지면 이 숫자도 따라서 늘려야한다.

4. 컴파일과 실행 방법

1) eclipse에서 SimpleDBMSParser.jj 오른쪽 클릭 -> Compile with javacc -> run

2) cmd에서 jar파일이 있는 directory로 들어감 -> java –jar PRJ1-3_2016-12299.jar 로 실행(같은 경로 안에 db폴더를 만들어야 함)

5. 프로젝트를 하면서 느낀 점

4학년까지 다니면서 지금까지 했던 개인 프로젝트 과제 중 가장 길고 커다란 과제였던 것 같다.

처음엔 DB 구현을 정말 할 수 있을까라는 생각을 했지만 총 3단계에 걸쳐 차근차근 작성하다 보니 결국 잘 끝마칠 수 있었다. 앞으로도 대형 프로젝트를 마주친다면 이번에 한 것처럼 해야 할 할 것 쪼개어 차근차근 진행할 것이며, 무턱대고 코드를 작성하는 것이 아니라 긴 시간 동안 설계를 할 것이다.

특히 이번 프로젝트의 핵심은 string 배열과 ArrayList를 잘 다루는 것이었다. 변수와 저장해야 할 데이터가 너무나 많았기 때문에 필요한 정보를 필요한 변수에 적절한 형태로 담는 것의 어려움을 알게 되었다. 또한 변수명의 중요성도 알게 되었다.

우리가 사용하는 query들에는 얼마나 많은 질서와 규칙이 있는지 알게 되었고, 수많은 query연산을 수행하는데 무결성이 깨지지 않도록 하는 것이 무척 어렵다는 것도 알게 되었다.

또한 나는 테이블 별로 DB를 만들었지만, 친구들은 대부분 하나의 TABLE에 모든 것을 저장한 후, KEY를 이용해서 구분을 하는 사람도 있었다. 구현 방법마다 장단점이 있지만 1-2에서는 create와 drop이라는 것의 의미에 맞추어 테이블 별로 DB를 생성했었다. 그런데 1-3에 와보니 이 방법이 DB를 자주 열고 닫아야 한다는 단점은 있지만 SELECT, DELETE, INSERT를 구현하기에는 매우 편했다. 결국 구현하고자 하는 것의 의미에 맞추어 자연스러운 코딩을 추구한다면 더 깔끔하고 오류 없는 코드로 발전할 수 있다는 것을 알게 되었다.

무엇보다 시간을 들이면 결국 커다란 프로젝트도 해결할 수 있다는 것을 느껴서 좋았다.