데이터베이스 1-3 보고서

2019-14355 배문성

May 14, 2023

1 Insert

Insert의 경우에는 크게 2가지로 나뉜다. column에 대한 parsing과 value에 대한 parsing이다. parsing을 하기 위해서는 column name과 data를 dictionary로 parsing하여서 전달하였다. 이때 기본 value를 null로 하여서 지정되지 않은 column에 대한 입력은 자동으로 null이 되게끔 전달하였다.

2 Where

Delete와 Select의 경우에는 Where절에 대한 연산이 선행되어야 수행이 가능하기에 먼저 구현하였다. Where절의 경우에는 lark를 통해서 각각의 비교문을 parsing하고(자세한 parsing은 Condition에서 서술하겠다. where절의 구조의 경우에는 lark를 통해서 tree구조로 parsing하는 것보다, 원문을 이용해서 postfix로 parsing하였다.

3 Condition

Condition 경우에는 실제 비교문을 저장하는 Class로 2개의 compand와 비교 연산자, null 비교문을 위한 `table_name`, `column_name`, `is_not`으로 구성되어있다. 각각의 compand는 실제 비교문에서 operation 양쪽에 위치하는 value에 해당되며, literal value, 또는 table과 column의 조합 2가지로 표현할 수 있다.
table과 column의 조합의 경우에는 table name이 비어있을 수 있는데, 이 경우에는 각각 선택된 table들의 column name을 순회하면서, 해당하는 column name을 찾아서 할당하였다. 이 방식을 통해서 nonexistcolumn이나 ambigiouscolumn과 같은 error도 추가적으로 검출하였다.

4 Record

Record의 경우에는 현재 DB에 저장되어있는 데이터를 Prasing하여 저장하는 class이다. Delete나 Select절이 호출될 경우, 먼저 from절을 통해서 불린 table들으 데이터를 cartesian product하여 origin data로서 저장한다. 이후 where절에서 parsing한 postfix 구문을 연산함을 통해서 실제로 조건에 해당하는 데이터를 추출한다.
postfix 구문을 실행하는 과정은 크게 2가지로 나뉘는데, 각각의 condition을 수행하는 과정과 논리 연산을 수행하는 과정으로 나뉜다. 먼저 condition연산을 수행하는 과정은 condition variable을 통해서 양쪽의 value를 읽어들이고, 실제로 비교하여 이에 해당하는 data들만을 origin data로 부터 추출하여서 data table을 만들고 이를 반환하여 스택에 저장한다.
이렇게 스택에 저장된 data들은 논리연산의 과정을 거치는데, not의 경우에는 origin data와 비교하여 차집합을 반환하며, and와 or의 경우에는 stack에서 2개의 data를 받아와서 각각 교집합, 합집합을 반환한다.
최종적으로 모든 연산이 진행된 후에는 self.data라는 변수에 연산의 결과(stack의 마지막 원소)를 저장한다.

5 Delete

이렇게 저장된 Record를 사용하여 delete의 경우에는 db로부터 읽어들인 하나의 column이 실제 record에 포함되는지를 검사하고 이에 해당하면 해당 column을 삭제한다.

6 Select

select는 from절을 통해서 관련 table name, select list를 통해서 원하는 column list를 입력받는다. 이 때, 비어있는 table name에 대한 처리는 condition에서 수행했던 방식과 동일한 방식으로 수행하였다. 이렇게 parsing된 column들은 record 내부에 저장되어있는 dictionary를 이용해서 tablename.columnname의 key를 idx로 변환하여 record에 저장되어있는 data 뽑아온다.

7 Problems

본 과제에서 가장 어려웠던 부분은 where절을 parsing하는 과정이었다. 지금까지 이러한 연산문을 parsing하는 과정을 전부 postfix 방식으로 해와서 인지 다른 방법이 떠오르지 않았고(정확하게는 tree구조로 parsing하는 방법을 고안하였지만, lark의 parsing방법과는 그다지 맞지 않아서 파기했다.) 그 결과, record class와 같은 space-time overhead가 굉장히 커질 수 밖에 없는 방식이 되었다고 생각한다.