Proj 1-3 (Implementing DML)

핵심 모듈과 알고리즘에 대한 설명

이번 프로젝트에서는 실제 table에 record를 삽입하고 제거하고 탐색하는 기능을 추가했다. 한편, 프로젝트를 진행하면서 1-1, 1-2에서 사용했던 DB 구조나 클래스 구성을 대폭 변경했다. 따라서 새로운 DB구조와 클래스 구성에 대해서 먼저 설명하고자 한다.

1. **SimpleDBMSGrammar 내부 변경 및 데이터 구조 변경**
   기존에 쿼리를 실제로 berkelery DB와 연계하여 처리하는 부분을 jj파일 파서 내부에 정의했었는데, 그 코드가 상당히 길다. 그에 비해, jj의 코드 edit 관련된 기능이 좋지 않아서 쿼리 처리 부분을 모두 클래스로 구성하여 java 파일에서 진행되도록 했다.
   데이터 구조 관련해서 기존 방향은 berkelery DB 객체 두개를 이용하여 <key, value> = <table_name, Table object>, <table_name, record> 두개의 Database를 만들고 테이블에 대한 Catalog 정보는 모두 첫번째 Database에, 실제 record들은 두번째 Database에 넣는 것이었다. 따라서 DDL 관련된 부분은 모두 첫번째에서, DML 관련된 부분은 두번째에서 처리되게 하려는 목적이었는데, 에러처리하는 과정에서 두 Database를 모두 참조해야하는 상황이 자주 발생해서 결국 Table object 내부에 record list를 넣고 하나의 Database만을 이용하기로 했다.
   이후는 추가된 각 클래스에 대하여 그 기능을 설명한다.
2. **추가/변경된 클래스**

- **DatabaseHandler.java**
  berkelery DB를 이용하기 위해 필요한 객체를 정의하는 클래스이다. DB에서 Table 객체를 가져오거나 넣을 때 여기에 정의된 멤버를 사용한다.
  public static Environment _myDbEnvironment
  public static Database _myDatabase
  public static Database _myClassDb
  public static StoredClassCatalog _classCatalog
  public static EntryBinding _tableDataBinding

- **DatabaseDefine.java**
  DDL과 관련되어 쿼리를 처리하는 모든 함수를 담고 있다.
  기존에 jj 파일에 정의되어 있던 것을 따로 클래스를 만들어서 담았다.
  public static void desc(String table_name)
  public static void showTables()
  public static void dropTable(String table_name)
  public static void createTable(Table table)

- **DatabaseManage.java**
  DML과 관련되어 쿼리를 처리하는 모든 함수를 담고 있다.
  (이번 프로젝트에서 새롭게 추가된 기능들을 처리)
  public static void delete(String table_name, ExpressionType EXP)
  public static void insert(String table_name, ArrayList record, ArrayList record_match_attributes)
  public static void select(ArrayList SNL, ArrayList STNL, ExpressionType EXP)

- **각종 Type관련 클래스들**

  - Types.java
    Where절에서 등장하는 Expression의 true/false 값을 계산하기 위해 각 요소를 Type 클래스로 치환한다.
    아래 6클래스의 `super class` 로 type number를 가지고 두 타입의 비교 관련된 `equal` 와 `comapreTo` 메소드를 가진다. 아래 child class들은 각자 type에 맞게 관련 멤버와 메소드를 가지고 있다.
    - CharType.java
    - DateType.java
    - IntType.java
    - NullType.java
    - VarType.java
      Where절에서 기본 type이 아닌 variable이 나왔을 때 이 type을 가지며 variable name과 실제 값을 구분하여 멤버로 가지고 있다. 실제값은 처음에는 null이지만 expression을 계산하면서 resolve된 record의 부분의 한 column의 값으로 채워진다.
    - ExpressionType.java
      where절에 나오는 식 전체를 담을 수 있는 type으로 solve라는 내부 메소드를 통해 recursive하게 expression을 계산할 수 있다.

- **errorMsg.java**
  에러 메세지 출력과 관련된 클래스.

- **JoinHandler.java**
  from 절에서 여러개의 테이블을 join하여 record by record로 순회해야하는데, 이 역할을 담당해주는 클래스이다. 이 클래스를 사용하여 위와 같은 작업을 Iteratior 클래스를 사용하는 것처럼 할 수 있다.

- **SelectName.java**
  variable의 resolution 관련 하여 필요한 클래스이다. 하나의 a 테이블에 있는 b attribute를 의미하는데 b, a.b, b as T, a.b as T 등 다양한 표현이 나올 수 있으므로 그 다양한 표현을 하나의 클래스 객체로 담고 유용하게 사용할 수 있다.

- **예외 처리 관련 클래스**
  다른 에러 처리와 다르게 **DatabaseManage.java**에 정의되어 있는 함수에서 에러가 나는 것이 아니라 where절의 expression의 결과 값을 구하는 solve함수가 recursive call된 상황에서 중간에 에러를 처리하기 위하여 필요한 Exception 클래스들이다.
  WhereAmbiguousReference.java
  WhereColumnNotExistException.java
  WhereIncomparableException.java
  WhereTableNotSpecifiedException.java

구현한 내용에 대한 간략한 설명

Insert

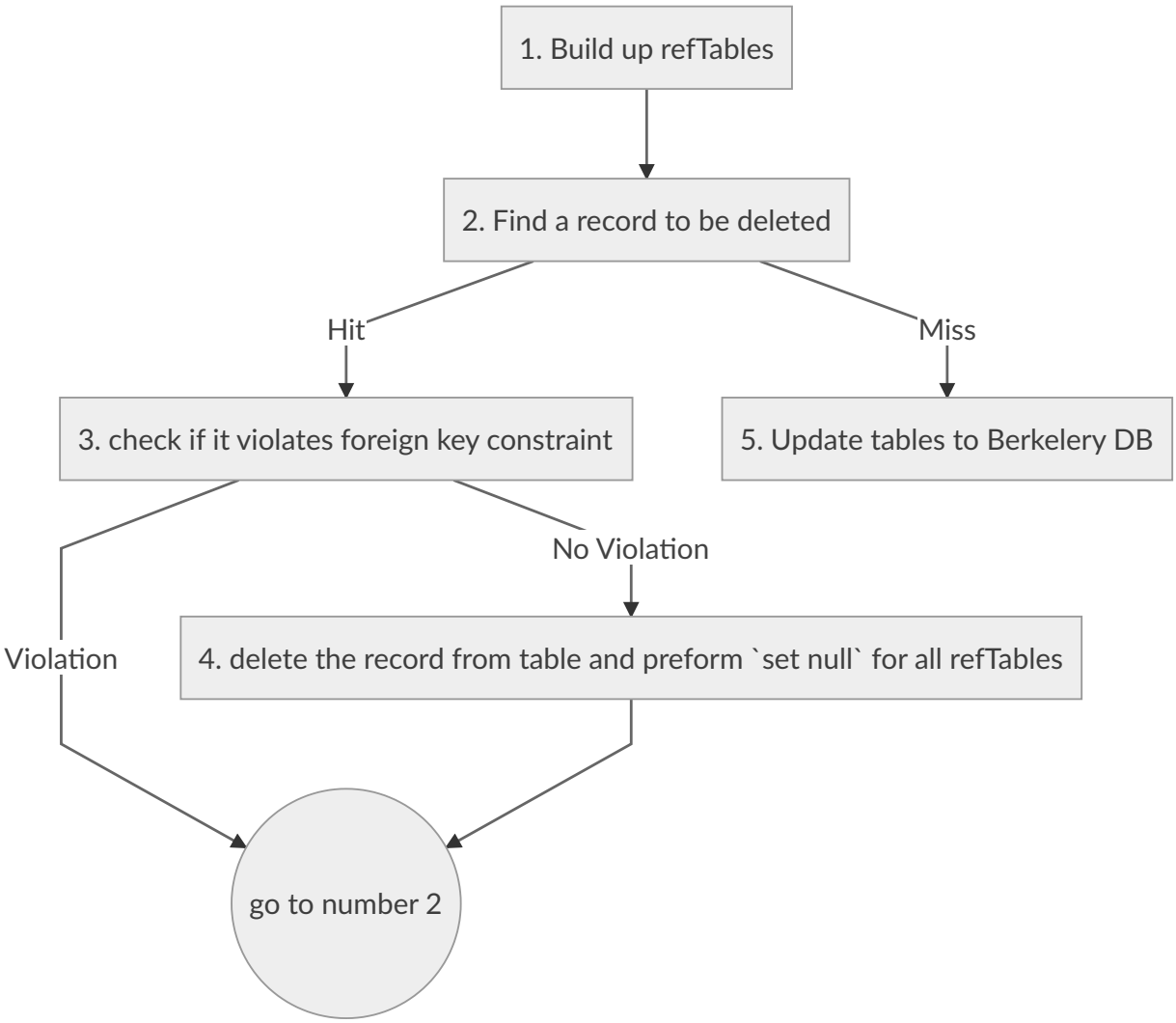
하나씩 에러처리를 해주면서 모든 에러가 없다고 판단될 때 table에 record를 넣었다. 각 에러 처리에 대해서 순서대로 간략하게 설명한다. 이해하기 쉽게 예제를 이용하도록 한다.

```
create table a(b int, c char(2));
insert into a (b, c) values(10, 'aa');
```

1. (b, c)와 같이 column을 명시한 경우 그곳에 나오는 column 수와 values 뒤에 나오는 column 수가 같은지 검사.
2. a라는 table이 있는지 검사
3. a라는 table의 attribute의 수와 values 뒤에 나오는 column 수가 같은지 검사.
4. 만약에 (b, c)와 같이 column 명시가 되어있지 않으면 단순히 insert하려는 record의 각 column 값이 table의 각 attribute의 type과 같은지 혹은 record의 column값이 null인데 정말 null이 될 수 있는 column인지 검사하면 되고, 그렇지 않고 명시가 되어있는 경우에는 이것 이외에도 각 column이 실제 table에 정의 되어있는 attribute인지도 검사한다.
5. insert할 record의 column 값중에서 char type은 이 단계에서 실제 attribute의 길이에 맞게 truncate 시킨다.
6. 이 record의 primary key 부분이 중복되지 않았는지 검사한다.
7. record의 foreign key 부분의 경우 참조하는 table에 존재하는지 검사한다. 단, foreign key 부분 중에 null이 하나라도 있으면 검사는 통과한다.
   위의 모든 검사를 통과하고 나면 해당 record는 insert가 가능하므로 table에 record를 추가하고 변경된 테이블을 berkelery DB에 반영한다.

delete

delete는 단순히 where절을 만족하는 record를 지우는 query이지만 여러가지 에러처리를 하다보면 단순하지 않다. 다른 모든 DML query들도 마찬가지이다. 그래서 delete의 경우도 다양한 에러처리를 해줘야 하지만 foreign key constraint와 관련된 에러가 가장 중요하므로 이 에러를 기준으로 설명한다.

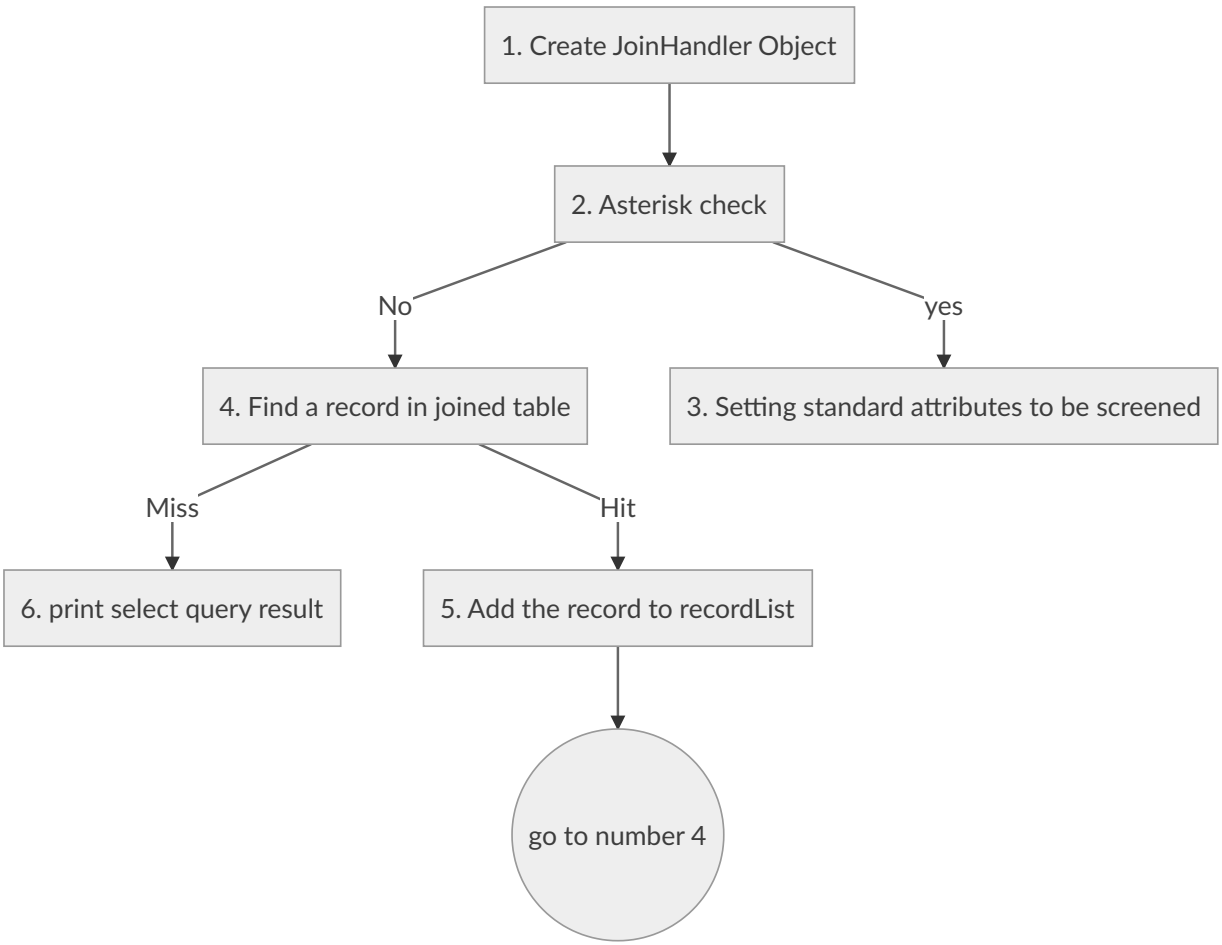


1. record를 지우고자 하는 table을 참조하는 모든 테이블의 이름을 refTables라는 arraylist에 넣는다.
2. table에 들어있는 모든 record에 대하여 where절의 expression을 solve 함수를 통해 값을 계산한다. 만약 값이 Type.T이면 where절을 통과한 record라는 것이고 지울 수 있는 record 후보가 된다.
3. 1번에서 구한 모든 refTable에 대하여 record의 primary key부분과 동일한 foreign key 부분을 가진 record를 가졌는지와 그 부분이 null 값으로 변경될 수 있는지 확인한다. 만약에 하나의 refTable 이라도 전자는 만족하는데 후자(nullable)가 만족하지 않는다면 foreign key constraint를 위반한 것이다.
4. 만약 foreign key constraint를 위반하지 않았다면 해당 record를 table에서 지우고 모든 refTable에 대하여 해당 record의 primary key 부분을 참조하는 foreign key를 null로 변경한다. 단, 이때는 berkelery DB에 반영하는 것이 아니라 Table 객체의 record list에 변경을 하는 것이다.

5. 더이상 지울 record가 없다면 변경된 table과 모든 refTable을 berkelery DB에 반영한다.

Select

select 함수 자체는 복잡한 로직이 없지만 helper class들이 매우 복잡하다. 모두 다 설명하기에는 너무 복잡해서 대략적인 select work flow와 ExpressionType 객체에서 true, false, unknown 이 세 가지 value를 어떻게 다루는 지만 언급하려고 한다.



1. 위에서 짧게 설명한 것처럼 JoinHandler 클래스는 from 절에서 여러개의 테이블을 join하여 record by record로 순회해야할 때 사용한다. JoinHandler Object를 만들어서 생성자를 실행시키면 joined table로 만들 수 있는 첫번째 record를 Object내에 가지고 있게 된다. 이후 `getNext()` 메서드를 이용하여 계속해서 다음 record를 가지고 있게 할 수 있다.
2. select query에서 표시할 attribute list에 별표를 입력한 경우 result를 출력할 때 attribute name으로 출력해야하므로 그 작업을 3번에서 해준다. 그게 아니라면 query를 parsing할 때 이미 무엇을 출력해야하는 지 받았으므로 그것을 출력하면 된다.
3. JoinHandler에서 받은 각 record에 대하여 where 절을 통해 받은 ExpressionType의 `solve` 메소드를 이용해 결과 값을 구한다. `solve` 메소드는 and, or, not일 경우 recursive하게 `solve` 메소드를 호출하여 전체 predicate의 값을 계산한다. 여기서 계산하는 값은 true, false, unknown 세가지 값인데, 계산을 편리하기 위하여 각 값에 1, -1, 0을 대응시켰

다. 그리고 A and B를 min(A, B), A or B를 max(A, B), not A를 -A 연산에 대응시켰다. 이렇게 하면 불필요한 nested switch문을 없앨 수 있어서 매우 효과적이다.
4. 출력할 record를 찾았으면 나중에 한번에 출력하기 위해 recordList에 넣어 놓는다.
5. 더이상 출력할 record가 없으면 recordList에 넣어놓은 모든 record를 출력한다.

가정한 것들

1. 입력으로 **들어올 수 있는 문자**는 다음과 같다.
    - Alphabet(a~Z)
    - number(0~9)
    - special characters on the keyboard
    - new line, carrige return, space
2. Where 절 관련
    - null 값은 다른 모든 타입의 값과 비교할 수 있다.
    - char 타입의 값은 길이와 상관 없이 다른 char 타입의 값과 비교할 수 있다.
3. Insert 관련
    - char컬럼 타입에 명시된 최대 길이보다 긴 문자열을 삽입하려 할 때는, 에러를 발생하지 않고 길이에 맞게 자른(truncate) 문자열을 삽입한다.
    - 테이블의 컬럼 이름은 중복되지 않는다.
    - column list가 주어졌을 때 거기에 table schema에 정의된 attribute가 없는 경우 그 값은 null로 초기화 한다. 해당 attribute가 `not null` 인 경우 **InsertColumnNonNullableError(#colName)** 에러를 낸다. 한편, column list가 주어지지 않고 values로만 값이 주어지면 table schema의 atrribute 수와 맞아야 한다. (없는 부분을 null로 초기화 하지 않는다)
4. foreign key constraint 관련
    - 어떤 테이블도 자기 자신을 참조할 수는 없다.
    - 같은 foreign key가 여러 테이블의 칼럼들을 참조할 수 없다.
5. column name과 키워드들은 case insensitive하고 나머지는 case sensitive 하다.

6. 그 외 애매한 경우는 MySQL 문법을 따른다.

컴파일과 실행 방법

compile

*eclipse*에 **javacc plug-in**을 설치하게 되면 매번 저장할 때마다 자동으로 'jj'파일을 컴파일 해준다.
만약 자동으로 컴파일 되지 않는다고 해도 'jj'파일을 우클릭한 뒤 **compile with javaCC**를 선택하면 수동으로 컴파일 할 수 있다.

실행방법

실행도 마찬가지로 *eclipse*에서 **run**버튼을 눌러서 프로젝트를 바로 실행할 수 있다.
혹은, 실행파일로 만들어서 실행하고 싶다면 해당 프로젝트를 우클릭 하고 **Export -> Runnable JAR File**을 선택한 뒤 **launch configuration**에 해당 프로젝트를 넣고 **finish**를 누르게 되면 Jar파일을 생성한다.
생성된 Jar 파일을 실행하기 위해서는 **terminal**에서 해당 jar 파일 위치로 간 뒤에 아래 shell 명령어를 실행시키면 된다.

```
java -jar <file name>.jar
```

프로젝트를 하면서 느낀 점

1-2의 연장선이라 느껴서 1-3은 금방 끝낼 수있을 거라고 생각했지만, 오산이었다! select에서 다량의 테이블을 join하는 것과 where 절을 계산하는 것과 true, false, unknown 값의 연산을 어떻게 다룰 것인지 등 생각해야할 게 많았다. 한편, 이전 프로젝트에서 만들었던 것을 허물고 새로운 데이터 구조를 만들던가 새로운 클래스를 만드는 경우도 있었고 잘쓰던 메소드를 나중에 보니 다른 메소드로 대체할 수 있다는 것을 깨닫고 파기하기도 했다. 그리고 이 모든 것은 에러처리와 관련이 있었다. 즉, DB를 구현하기 전에 무엇이 정상적인 query이고 비정상적인 query인지 정의를 확실히 하고 어떻게 구분할 것인지를 생각해야 나중에 다시 갈아 엎을 일이 줄어든다는 것을 깨달았다.