# 2024-2 Database (001) Project 1-2 Report

2023-12675 박지호

이번 프로젝트에서는 이전에 만든 SQL query를 parsing하는 프로그램을 활용해, `CREATE TABLE`, `DROP TABLE`, `EXPLAIN`/`DESC`/`DESCRIBE`, `SHOW TABLES` 및, 기본적인 `INSERT`, `SELECT`문을 수행하는 프로그램을 제작하였다.

이때, 프로젝트의 규모가 커짐에 따라 프로젝트를 여러 모듈로 분리하였으며, 각 모듈이 독립적인 기능을 수행하게 구성하였다. 우선 `parse_query` 모듈에는 이전 프로젝트에서 사용하였던 `receive_queries`, `create_sql_parser` 함수가 정의되어 있다.

`process_query` 모듈에는 Lark 라이브러리에 의해 tree 형태로 parsing된 query를 수행하는 함수들이 정의되어 있다. 이는 이전 프로젝트의 `process_query` 함수와 비슷한 구조이지만, 사용하는 `Transformer` 클래스를 `QueryTreeProcessor` 클래스로 변경하였다. 이전 리포트에서 소개하였듯 `Transformer` 클래스는 tree를 bottom-up으로 순회하며 각 노드를 transform하는데, `QueryTreeProcessor` 클래스의 경우 각 노드에서 유의미한 데이터를 추출해 반환한다. 이들은 부모 노드에서 추가로 evaluate되며, 결과적으로 각 query에 해당하는 노드에서는 이렇게 구해진 값들을 사용해 서브모듈인 `query`에 정의된 함수를 호출해 query를 실제로 수행한다.

`interface` 모듈 및 `models` 모듈에는 column, table constraint, relation, table 등 데이터베이스의 여러 개념에 대응되는 클래스들이 정의되어 있다. 이때, `interface` 모듈에는 abstract 클래스가, `models` 모듈에는 concrete 클래스가 정의되어 있다. `QueryTreeProcessor`에서 primary_key_constraint, column_definition 등 복잡한 값을 반환해야 하는 노드의 경우 대응되는 `models` 클래스가 정의되어 있어 이를 반환하는데, 일부 클래스의 경우 이상이 있는 데이터가 주어지면 곧바로 exception을 raise한다.

이 클래스들은 대부분 부모 노드로 정보를 전달하기 위한 container 역할만을 수행하지만, `Relation` 클래스의 경우 `select`, `where`, `to_alias` 등의 메소드를 가지며, `Table`의 경우 `Relation`을 상속하며 `add_foreign_ref`, `insert_record`, `explain` 등의 메소드를 추가로 가진다. 이 메소드들은 query 수행 시 호출된다. 또한, `Table` 클래스의 경우 `foreign_refs` field를 통해 자신을 참조하는 table의 이름을 저장한다. 이는 `DROP TABLE` query 수행 시 자신을 참조하는 table이 있는지 확인하는 데 사용되며, `CREATE TABLE` query 수행 시 업데이트된다.

`db` 모듈의 경우 데이터베이스 table을 파일에 저장하고 불러오는 함수들을 정의한다. 이 작업을 위해서는 berkeleydb 라이브러리를 사용하였는데, 이 라이브러리는 byte array의 key-value pair를 파일에 저장하는 기능을 제공한다. 이때, `pickle.dumps`와 `pickle.loads`를 사용해 `Table` object를 통째로 byte array화하여 저장하였다. 이 모듈은 berkeleydb 관련 로직과 query 수행 로직 사이의 추가적인 abstraction layer를 두어 분리하기 위해 도입하였다.

Query를 수행하는 과정에서 입력된 데이터에 이상이 있는 경우 앞서 간단히 언급하엿듯 exception을 raise한 뒤, `process_query` 함수에서 이를 catch하여 exception의 message를 출력하는 방법을 사용하였다. 일반적으로 이와 같은 식의 구현은 안티패턴이라고 알고 있지만, 이 경우 의미상으로도 exception이 해당하기에 문제가 없다고 판단하였다. `errors` 모듈은 이를 위해 필요한 custom exception들을 정의한다. 이들은 대부분 `models`에 정의된 클래스의 constructor 또는 메소드에서 raise되는데, `NoSuchTable` 등 일부 exception은 `query`에서 raise된다. 또한, 명세에 있는 에러 외에도 동일한 column이 하나의 PRIMARY KEY 또는 FOREIGN KEY에서 여러 차례 명시된 경우의 `PrimaryKeyDuplicateColumnError`, `ForeignKeyDuplicateColumnError`, `ForeignKeyDuplicateReferenceColumnError`, FOREIGN KEY로 정의된 column 수와 참조하는 column들의 수와 다른 경우의 `ForeignKeyLengthError`, INSERT 시 값의 길이가 column 수와 다른 경우의 `ValueLengthError`, 타입이 column의 타입과 다른 경우의 `ValueTypeError`, NULL을 NOT NULL column에 넣으려는 경우의 `NullValueError` 등을 정의하였다.

마지막으로, `utils` 모듈에는 출력을 위한 상수, `duplicate_exists`, `filter_by_type` 등의 helper functions, 데이터베이스 파일의 경로를 반환하는 함수 등이 정의되어 있다.

이번 프로젝트에서는 이후 필요하게 될 기능을 구현하기 수월하게 프로젝트 구조를 설계하는 것이 목표였다. 이를 위해 각 기능을 의미상 적절하게 모듈로 분할하였다. 또한, `CREATE TABLE`을 구현하는 단계에서는 `Relation` 클래스 없이 `Table` 클래스만 만들었었는데, `SELECT`문을 구현하는 과정에서 이후 cross product, where clause, select column 등을 구현하기 위해선 constraint나 foreign reference 등 없이 name, column list, record만 가진 클래스가 필요한 것을 깨달았다. 또한, `Table`이 이 클래스를 상속하는 것이 자연스러웠기에 `Table`의 base class로 `Relation` 클래스를 추가하고 해당 메소드들을 정의하였다. 이 과정에서 아직 구현이 요구되지 않은 기능 또한 일부 구현하게 되었는데, `SELECT`문에서 일부 column만을 선택하는 기능, table alias, column alias 등이 구현되었다. 앞으로는 `Relation` 클래스의 곱셈 연산을 오버로딩해 cross product를 구현하고, 하나의 record에 대해 `WHERE` clause에 해당하는 Tree를 `bool` 값으로 evaluate하는 `Transformer` 클래스를 만들어 `WHERE` clause 구현하고자 한다. `DELETE`와 `UPDATE`의 경우 아직 생각해보지 않았으나, `WHERE` clause가 구현되면 이를 사용하는 메소드를 `table`에 추가해 구현할 수 있으리라 생각된다.

또한, `QueryProcessor`의 경우 query에 대응되는 top-level 노드에서 필요한 정보를 찾는 대신, 자식 노드들을 전부 evaluate하며 올라가 각 노드에서 direct children만 처리하면 되는 방식으로 구현하였다. 이 방법은 각 노드를 evaluate하는 코드가 간단하며, column_name, table_name 등 여러 쿼리에서 사용되는 노드가 하나의 함수로 일괄적으로 evaluate되기 때문에 효율적이다. 이는 이후 다른 query를 구현할 때에도 도움이 될 것이다. 또 데이터에 이상이 있는 경우 이상이 있는 것을 확인할 수 있는 노드에서 바로 검사되기 때문에 데이터를 전부 모은 후 검사하는 것보다 직관적이다.

마지막으로, 초기 구현 시에는 파이썬 3.8에서 작동해야 하는 사실을 알지 못해 match-case문 및 3.10에서 추가된 여러 type hint 문법을 사용했는데, 이들 중 일부의 경우 3.8에서는 런타임 에러를 띄우기 때문에 문제가 있었다. 따라서 구현이 끝난 이후 이들을 파이썬 3.8의 type hint 문법으로 변경해야 했다. 하지만 주석의 경우 이해가 쉬운 현대적인 문법으로 유지하였다.