2024-2 Database (001) Project 1-3 Report

2023-12675 박지호

이번 프로젝트에서는 이전에 만든 SQL 인터프리터에 추가로 `SELECT`, `INSERT`, `DELETE` 등 DML 기능을 완전히 추가하여, 기본적인 SQL query를 수행할 수 있는 프로그램을 완성하였다.

이전 리포트에서 설명하였던 것과 같이, 이 프로젝트는 여러 모듈로 구성되어 있는데, 이번 단계에서는 `models` 모듈의 `Relation` 클래스와 `Table` 클래스, 그리고 `process_query`에서 해당 클래스의 인터페이스를 호출하는 부분을 주로 수정하였다.

우선 `INSERT`문과 `DELETE`문의 경우 파일에서 `Table` object를 불러온 후, 이를 modify하고 다시 파일에 저장하는 과정을 거쳐 수행한다. 이때 `Table` 클래스의 `insert_record`, `delete` 메소드가 호출된다. `INSERT`의 경우 이전 단계에서도 구현이 사실상 끝난 상태였지만, 에러 메세지를 제공된 것으로 바꾸는 것과, 특정 column을 명시하는 경우의 처리 등 사소한 수정이 있었다. `DELETE`의 경우 후술할 `WHERE` clause의 구현을 사용해 삭제할 record를 선택하였다.

`SELECT`문 수행을 위해서는 `Relation` 클래스에 relational algebra의 각 연산자에 대응되는 메소드를 구현하였다. 이때 $\bowtie_\theta$에 대응되는 `join`, σ에 대응되는 `where`, Π에 대응되는 `select`를 만들었으며, 각 메소드가 이 순서대로 호출되어 `SELECT` query가 수행된다. 또한, `order_by` 메소드는 `ORDER BY` clause를 처리하며, 이는 `where`와 `select`사이에 호출된다. 추가적으로 $\times$에 해당하는 `product` 메소드 또한 구현하였는데, 이는 query 수행 과정에서 직접적으로 사용되지는 않으나, `join`의 구현에 사용된다.

이 중 `WHERE` clause와 `ON` clause의 경우 이전 단계에서 계획한 것과 같이 `WhereProcessor` 클래스를 만들어 처리하였다. 이는 조건식을 evaluate하는 `Transformer` 클래스로, 생성자에서 record 하나를 받아 해당 record가 조건을 만족하는지 여부를 계산한다. 이때 조건식은 tree 형태에서 bottom-up으로 evaluate되는데, 이는 우선 조건식의 값 부분을 해당 record의 값으로 대체한 다음, 비교 및 논리 연산을 수행해 최종 결과를 계산하는 식으로 진행된다.

또한, 추가적인 구현 사항이었던 `GROUP BY` clause 또한 구현하였다. 이를 위해서는 `select` 메소드의 인터페이스를 확장하여 추가로 `group_cols` parameter를 받게 하였다. 이때, 이 parameter를 주지 않는 경우 `GROUP BY`를 포함하지 않은 `SELECT`가, 주는 경우 `GROUP BY`를 포함한 `SELECT`가 수행된다. 또한, 이를 구현하기 위해 `Relation` 클래스에 record를 몇 개의 column을 기준으로 묶는 `group` 메소드와, record group에서 필요한 값들을 선택하여 하나의 record로 만드는 `select_from_group` 메소드 또한 구현하였다. 이들이 순서대로 호출되어 `GROUP BY`를 포함한 `SELECT`가 수행된다.

전체적으로, 이번 단계는 이전 단계에서 적절한 프로젝트 구조를 미리 설계해 둔 덕분에 별다른 어려움 없이 수행할 수 있었다. 일반적으로 규모가 있는 프로젝트를 진행할 때에는 개별 기능의 구현보다 각 부분의 연결 방식을 결정하는 것이 더 어려운 작업이다. 하지만 이번 프로젝트에서는 이러한 구조적 뼈대를 미리 마련해 두었기 때문에, 독립적인 기능을 구현하는 간단한 작업만으로도 제시된 명세를 구현할 수 있었다.

또한, 이 결과 명세에서는 일부 경우만을 지원하는 것을 요구하였지만, 특정 경우만을 구현하는 것이 오히려 더 지저분하고 복잡할 것 같아 좀 더 일반적인 경우까지 지원하도록 구현된 기능들이 몇 있다. 대표적으로는 `WHERE`, `ON` clause에서 일반적인 명시된 특정 조건뿐 아니라 모든 경우를 지원하는 것, `GROUP BY` clause에서 여러 column을 사용하는 것도 지원하는 것 등이 있다.

구현이 완료된 이후에는 `unittest` 패키지를 사용해 테스트 자동화 코드를 작성하였다. 테스트 코드는 `process_query`를 호출하였을 때 출력되는 값이 알맞은지 확인하는 식으로 구성하였다. 가능하다면 각 모듈의 내부 기능에 대한 유닛 테스트를 만들고 싶었지만, 이미 프로젝트가 최종 단계이기 때문에 이와 같이 high-level에서의 테스트만을 작성하는 것으로 결정하였다. 그럼에도 이는 잘못 구현한 부분을 확인

하는 데 큰 도움이 되었다. 또한 refactoring 과정에서 구현된 기능에 문제가 발생하지 않았는지 확인을 용이하게 하여, 구현이 깔끔하지 않은 부분을 과감히 수정할 수 있게 해 주었다.

추가적으로, 수업 시간에 데이터베이스를 저장하는 방법에 대한 low-level optimization을 배우고 나니, 이와 같이 table을 통째로 binary encoding하여 저장하는 방식이 상당히 비효율적이라는 생각이 든다. 실제로 임의의 query를 수행하기 우선 table 전체를 메인 메모리에 로딩하여야 하기 때문에, 이 프로그램의 경우 규모가 큰 데이터베이스를 다루기에는 적합하지 않을 것이다. 하지만 각 기능을 독립적으로 분리해 둔 덕분에, 이를 지원하도록 구현을 수정하는 것은 비교적 간단할 것이다. `Table` 클래스를 사용해 record를 dynamic하게 읽어오는 것을 지원하는 클래스를 만들고, 파일에 읽고 쓰는 작업을 수행하는 `db` 모듈이 이 클래스를 사용하도록 바꾸는 등의 방법이 가능할 것이라 생각한다.