2024-2 Database (001) Project 2 Report

2023-12675 박지호

이번 프로젝트에서는 MySQL 데이터베이스를 활용하는 DVD 대출 애플리케이션을 제작하였다. 이 애플리케이션은 CLI로 사용할 수 있으며, dvd 추가, 삭제, user 추가, 삭제, dvd 대출, 반납 및 평점 부여, dvd 검색 등 여러 기능을 지원한다.

이 애플리케이션의 데이터베이스는 `user`, `dvd`, `director`, `rent`, `rating`의 총 5개의 테이블로 구성되어 있다. `user` 테이블의 경우 user의 id, 이름, 나이를 저장하며, (이름, 나이)에 unique constraint가 있고, `dvd` 테이블의 경우 dvd의 id, 제목, 감독명, 남은 수를 저장하며, (제목, 감독명)에 unique constraint가 있다. `rent` 테이블의 경우 대출 기록을 저장하며, 대출 id, 유저 id, dvd id, 그리고 반납 여부를 저장하고 있다. `rating` 테이블은 `rent` 테이블의 대출 id를 foreign key로 reference하며, 해당 대출에 대한 평점을 저장한다. 이때, 반납 여부를 별도로 저장하지 않고, `rent` 테이블에 바로 평점을 저장하고, 평점이 NULL이라면 반납되지 않은 것으로 취급하는 등의 방식 또한 생각하였으나, 이는 추가적인 정보 없이 이해하기 어렵다고 생각하여 이와 같은 방식을 채택하였다.

마지막으로, `director` 테이블은 감독의 이름만을 저장하는데, 이는 감독명 검색 기능이 있기 때문에 감독명에 대한 deletion anomaly가 발생하면 안 되기 때문에 추가하였다. 이때, dvd의 감독명에서 director의 감독명에 대한 foreign key가 정의되어 있는데, 일반적으로는 감독에게 id가 부여되어 있고, 이 id를 foreign key로 사용하는 것이 적합하였을 것이다. 하지만 이번 프로젝트에서는 dvd의 (제목, 감독명)이 unique해야 한다는 조건이 있어 이와 같은 방식을 채택하였다. 감독에 id를 부여하고 `director` 테이블에서는 감독명에 unique constraint를, `dvd` 테이블에서는 (제목, 감독 id)에 unique constraint를 두는 것으로도 이를 보장할 수는 있지만, 이는 충분히 직관적이지 못하다고 판단하였다.

이 5개의 table 외에도 추가로 `dvd_view`와 `director_view`를 두었다. 우선 `dvd_view`의 경우 `dvd` 테이블에 있는 정보에 추가로 누적 대출 횟수와 평균 평점을 가지고 있으며, `director_view`의 경우 각 감독의 누적 대출 횟수와 평균 평점을 가지고 있다. 이는 여러 query에서 사용되는 것이 확인되었기 때문에 제작하였다.

데이터베이스의 정확한 구조는 `db` 모듈의 `ddl.py`에 있는 `create_tables` 함수에서도 확인할 수 있다.

이번 프로젝트는 TDD를 위해 구현할 기능의 테스트를 먼저 작성하고, 구현을 이후에 하는 순서로 개발하였다. 테스트는 전부 `tests` 모듈 안에 작성되어 있다. 따라서 unit test가 용이하도록 프로젝트를 설계하는 것을 목적으로 하였다.

이를 위해 우선 애플리케이션 실행 시 선택할 수 있는 `Action`을 `Prompt`와 `Operation`으로 분리하였다. 이 때, `Prompt`는 `Operation`을 수행하기 위해 추가적인 값을 입력받는 부분이고, `Operation`은 이 값들을 사용해 필요한 동작을 수행하는 부분이다. 예를 들어, dvd 추가 `Action`의 경우 dvd 이름과 감독명을 입력받는 부분이 `Prompt`, dvd를 실제 데이터베이스에 추가하는 부분이 `Operation`이다. 이들은 `core` 모듈에 각각 클래스로 정의되어 있으며, `action` 모듈에는 `Action`과 `Operation`의 인스턴스가, `prompts` 모듈에는 `Prompt`의 인스턴스가 정의되어 있다. 이들이 서로의 구현에 의존하지 않기 때문에 개별적인 부분의 동작만 확인하면 전체 프로그램이 올바르게 동작하는 것을 보장할 수 있다.

또한, 데이터베이스 관련 동작을 테스트할 때는 sqlite3로 in-memory 데이터베이스에 접속하여 테스트한다. 테스트 시에도 제공된 production 데이터베이스에 접속하면 데이터를 전부 삭제하지 않는 한 예상 가능한 결과를 얻기 힘들기 때문이다. 이 때문에 데이터베이스 관련 동작을 실행하는 함수는 전부 `connection`이라는 파라미터를 받아, MySQL과 sqlite connection을 둘 다 지원한다. 하지만 sqlite에서는 사용할 수 없거나, 사용 방법이 다른 SQL 문법이 있어 이들의 경우 별도로 처리해주고 production 데이터베이스에서도 작동하는지 수동으로 확인해 주는 과정이 필요하였다.

또한 sqlite3와 mysql-connector가 일관된 인터페이스를 제공하지 않아 이들을 통일하기 위한 wrapper 클래스를 제작하여야 했다. 이때 이 wrapper 클래스에 몇 가지 편의성 메소드를 정의하였는데, SQL query를 수행하고 그 결과를 반환하는 `execute_and_fetch`, 결과의 row 수를 반환하는 `execute_and_count`, 결과가 존재하는지 여부를 반환하는 `execute_and_exists` 등이 있다. 또한, 데이터베이스 접속 시 close와 commit 등을 명시적으로 작성해야 한다면 이를 작성하지 않아 문제가 발생할 확률이 높다고 생각하였기 때문에, 여기에서 context manager를 사용해 with 블럭을 나갈 때 자동으로 commit 및 close가 되게 하였다. 이는 DDL과 함께 `db` 모듈에 정의되어 있다.

또한, 프로젝트 1에서 사용한 것과 같이 Exception을 throw하고 top-level에서 catch하는 것은 훌륭한 설계가 아니라고 판단하여, rust의 `Result`를 구현하여 사용하였다. `Result`는 subclass로 `Ok`와 `Err`을 가지고 있으며, 동작이 성공한 경우 그 결과값을 가지고 있는 `Ok`를, 실패한 경우 에러 메세지를 가지고 있는 `Err`를 반환하는 식으로 사용된다. `Result`는 `utils` 모듈에 정의되어 있으며, `utils`에는 그 외에도 출력 문자열 및 helper function들이 정의되어 있다.

또한, package manager로 poetry를 사용하였으며, `poetry run reset-db`를 통해 데이터베이스 초기화, `poetry run test`를 통해 테스트, `poetry run start`를 통해 프로그램을 실행할 수 있다. 물론 `python run.py` 역시 정상적으로 작동한다. 또한, requirements.txt는 poetry export를 통해 작성하였는데, 실제 사용한 모듈은 `mysql-connector-python`, `numpy`, `pandas`밖에 없지만, poetry export의 결과 이들이 사용하는 패키지 또한 requirements.txt에 포함된 것으로 보인다. 명시적으로 추가한 dependency는 `pyproject.toml`에서 확인할 수 있다.

프로젝트 수행 과정에서 파이썬의 본질적인 결함 탓에 상당한 불편함을 느꼈다. 특히 추가 구현 과제인 collaborative filtering은 구현하는 과정에서 numpy와 pandas가 제대로 typing되어 있지 않아 type: ignore를 많이 작성해 주어야 했다. 또한, 추가 과제의 경우에는 시간이 부족하여 상당히 아름답지 못하게 구현하였고, 테스트 코드 또한 작성하지 못하여 아쉽다.

마지막으로, 명세에 나와 명확하게 나와 있지 않은 부분은 임의로 구현하였다. 실행할 동작을 고르는 부분에서 올바르지 않은 입력을 받는 경우 “Please choose a valid action”이 출력된다. 또한, 데이터베이스 리셋 시 y와 n 외의 입력을 받는 경우 y나 n이 입력될 때까지 다시 입력을 받는다.