
데이터 베이스 프로젝트 보고서

국내 영화 정보 제공 및 리뷰 공유 프로그램 구축
< 모두의 영화 >

과 목 명 : 데이터베이스(02)

교 수 명 : 김연정 교수님

팀 명 : 데베레스트

팀 원 : 박소현(1971070)

배유진(1971075)

이소울(1971080)

목 차

I. 프로젝트 개요

1. 프로젝트 소개
2. 프로젝트 목적
3. 프로젝트 개발 일정
4. 개발 방법 및 도구 소개

II. 프로젝트 내용

1. 업무 기능도
2. 개념적 데이터 모델(ER-Diagram)
3. 논리적 데이터 모델(스키마)
4. 클래스 및 메서드 설명
 - 1) DB
 - 2) GUI
 - 3) Main
5. 응용 프로그램 상세 정보
 - 1) 사용 방법
 - 2) 기능 설명
6. 강점 및 편리성
7. 역할 분담

I. 프로젝트 개요

1. 프로젝트 소개

한국 영화에 대한 정보를 제공하고 영화에 대한 리뷰를 사람들과 함께 공유할 수 있는 프로그램인 '모두의 영화'를 개발하였다. 특히, 사용자와 관리자의 역할을 구분하여 정보가 관리될 수 있도록 만들었다. 사용자는 영화에 대해 제목, 장르, 배우, 감독, 영화제별로 영화를 효과적으로 검색할 수 있고, 영화에 대한 리뷰를 등록하고 검색해볼 수 있다. 관리자는 로그인을 통하여 영화 정보를 추가하거나 수정할 수 있고 특정 키워드가 들어간 리뷰를 삭제하여 클린한 리뷰 환경을 조성할 수 있도록 하였다. 특히, GUI를 통하여 사용자가 보다 편리하게 사용할 수 있도록 하였다

.

2. 프로젝트 목적

< 사용자 >

- 한국 영화에 대하여 제목 / 장르 / 배우 / 감독 / 영화제별로 검색
- 영화의 리뷰 작성
- 영화에 대한 리뷰를 다른 사용자와 공유

< 관리자 >

- 로그인을 통한 관리자 권한 부여
- 관리자의 영화 정보 추가 및 수정 가능
- 특정 키워드를 포함하고 있는 리뷰 삭제를 통하여 깨끗한 영화 리뷰 환경 조성

3. 프로젝트 개발 일정

5/4 ~ 5/10 : 데이터 베이스 구축

5/11 ~ 6/5 : JAVA 프로그램 개발 및 테스트

6/5 ~ 6/7 : 보고서 작성, 발표 ppt 제작, 발표 비디오 제작

4. 개발 방법 및 도구 소개

- JAVA 프로그램 : Eclipse
- Java 버전 : JDK-14.0.1
- 프로그램 인코딩 : UTF-8
- MySQL 버전 : 8.0

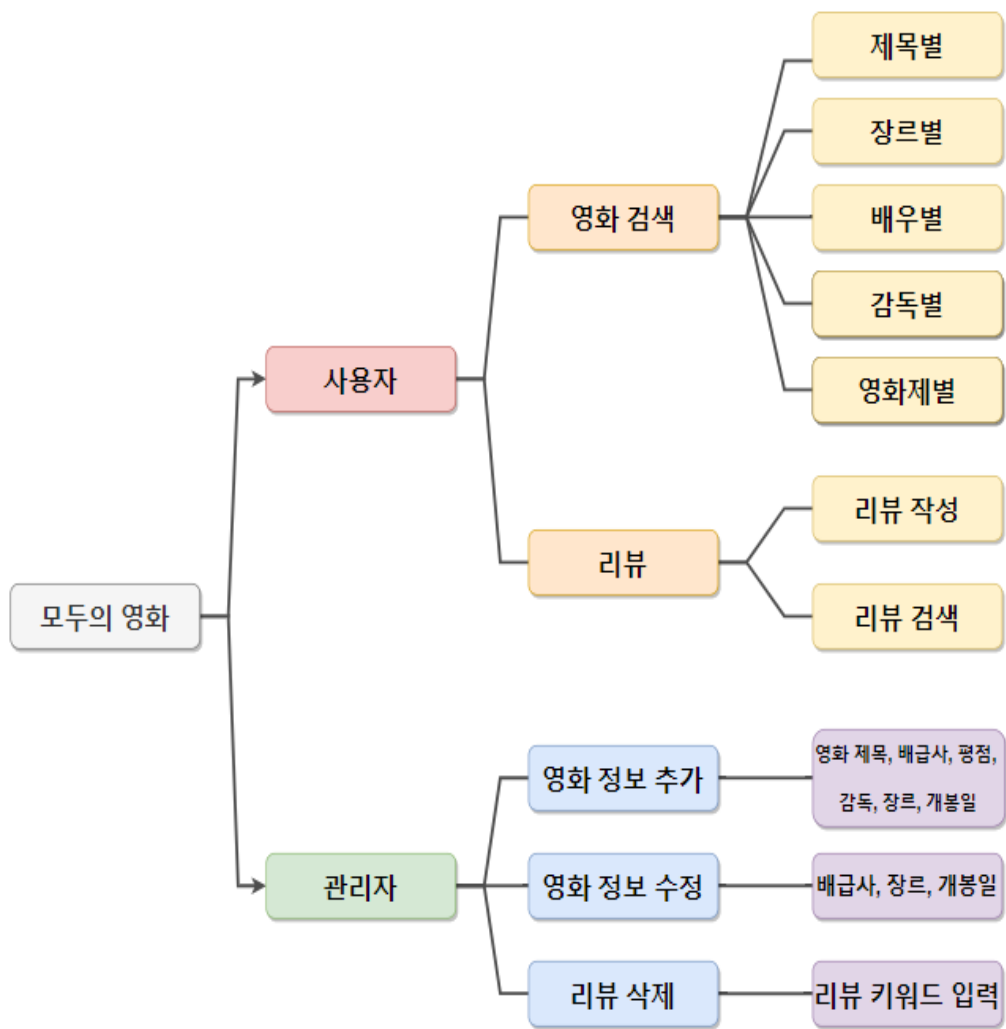
데이터 베이스의 DBMS와 자바 응용 프로그램을 연결시켜주는 응용 프로그램 인터페이스 JDBC를 이용하여 MySQL 데이터베이스로부터 정보를 삽입, 삭제, 갱신, 검색할 수 있는 자바 응용

프로그램을 개발하였다. 자바 응용 프로그램 개발 시 DB 폴더와 GUI 폴더를 따로 생성하여 역할에 맞게 클래스를 생성하였다.

II.프로젝트 내용

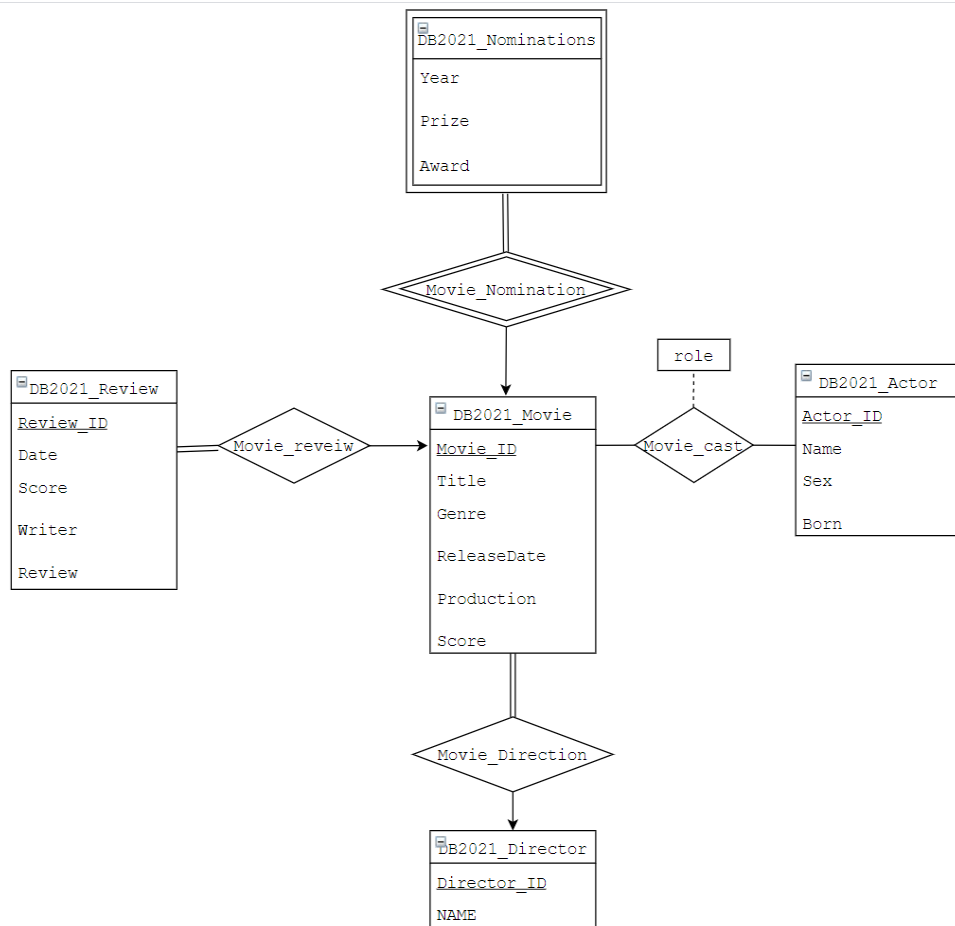
1. 업무 기능도

- 사용자와 관리자 기능 구분
- 사용자 : 영화 검색, 리뷰 작성 및 검색
- 관리자 : 로그인을 통해 영화 정보 추가 및 수정, 리뷰 삭제



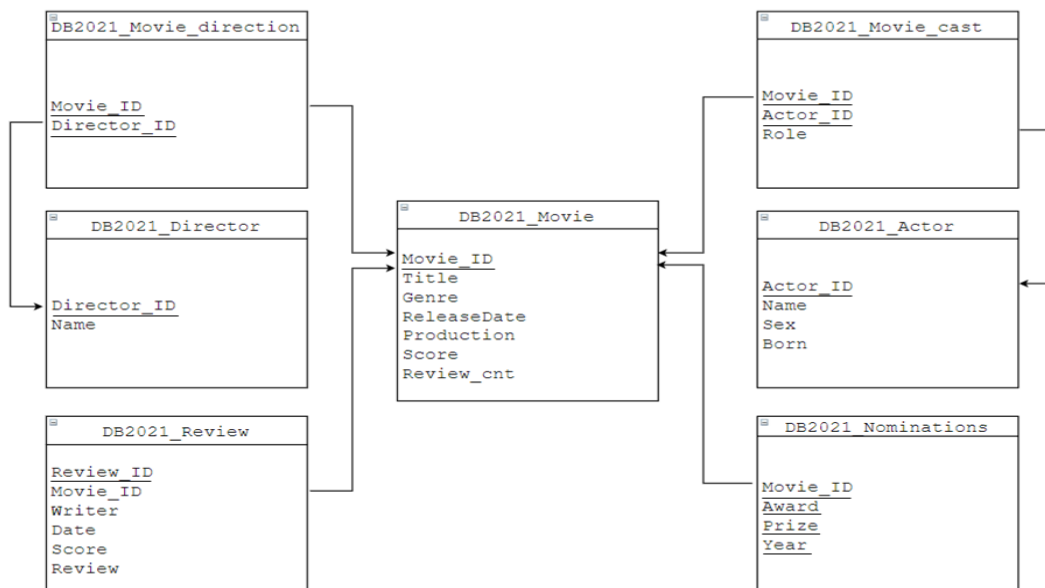
[그림 1] 업무 기능도

2. 개념적 데이터 모델 : ER-diagram



[그림 2] ER-diagram

3. 논리적 데이터 모델



[그림 3] 스키마 다이어그램

4. 클래스 및 메서드 설명

각각의 클래스들을 DB, GUI, main 패키지로 나누어 역할에 맞게 구분하여 생성하였다. 각 패키지에 대하여 영화 검색(Movie), 리뷰(Review), 관리(Admin) 세 개의 탭을 각각의 클래스로 제작하였다.

1) DB

① AdminDB : 관리자 기능을 사용할 경우 결과 데이터를 AdminGui로 보내는 클래스

메서드	설 명
find_director_id(String director)	감독의 이름을 인자로 받아 director_id를 찾아서 리턴
find_moive_id(String title)	영화 제목을 인자로 받아 movie_id를 찾아서 리턴
insert_moive(String title, String genre, String date, String production, String score, String director)	영화 제목, 장르, 개봉일, 배급사, 평점, 감독 이름을 입력 받아 영화 정보를 삽입해주는 메서드
modify_movie(String title, String genre, String date, String production)	영화 제목, 장르, 개봉일, 배급사를 입력 받아 영화 정보를 수정
delete_review(String keyword)	리뷰의 키워드를 입력 받아 해당 키워드를 포함하고 있는 리뷰 정보를 삭제

② MovieDB : 영화 검색 시 결과 데이터를 MovieGui로 보내는 클래스

메서드	설 명
awardList()	데이터 베이스에 저장된 영화제의 목록을 반환
genreList()	데이터 베이스에 저장된 장르의 목록을 반환
search_movie(String s)	영화의 제목을 인자로 받아 해당 이름을 가지고 있는 tuple list를 반환
search_genre(String s)	영화의 장르를 인자로 받아 해당 장르의 영화 정보 tuple list를 반환
search_actor(String s)	배우의 이름을 인자로 받아 해당 배우가 출연한 영화 정보 tuple_list를 반환
search_director(String s)	감독의 이름을 인자로 받아 해당 감독이 제작한 영화 정보 tuple_list를 반환
search_award(String s)	영화제의 이름을 인자로 받아 해당 영화제에서 수상한 영화 정보 tuple list를 반환

③ ReviewDB : 영화 리뷰 관련 기능 결과 데이터를 MovieGui로 보내는 클래스

클래스 및 메서드	설 명
ReviewClass{}	검색 시 보여줄 컬럼들을 class로 정의
search_review(String s)	리뷰를 검색하면 ReviewClass를 리턴
find_movie_id(String s)	영화 제목을 인자로 받아 movie_id를 찾아서 리턴
insert_update_reviewScore(String tf1, String tf2, String tf3, String tf4)	리뷰를 작성하면 리뷰를 넣고 평점을 업데이트

④ DBConn : 데이터베이스와 연결하는 클래스

클래스 및 메서드	설 명
getConnection()	mysql과 연결

2) GUI

① AdminGui : 관리자 기능 gui를 제공하는 클래스

클래스 및 메서드	설 명
modifyMovie()	관리자 페이지에 관여하는 메서드들을 동시에 실행시켜주고 탭을 생성
makeButtons(String str, JPanel panel)	버튼에 들어갈 내용과 panel을 입력 받아 버튼을 만들어 panel에 추가
insertMovie(JPanel p)	영화 정보 추가해주는 탭에 들어가는 내용에 관련된 메서드
modifyMovie(JPanel p)	영화 정보를 수정해주는 탭에 들어갈 내용에 대한 메서드
deleteReview(JPanel p)	리뷰를 삭제해주는 탭에 들어갈 내용에 대한 메서드

② MovieGui : 영화 검색 gui를 제공하는 클래스

클래스 및 메서드	설 명
getMoviePanel()	패널을 가져오는 메서드
moviePage()	패널을 만드는 메서드
genreComboBox()	장르별 영화 검색에서 사용하는 콤보박스 생성
awardComboBox()	영화제별 영화 검색에서 사용하는 콤보박스 생성
makeWindow(JPanel window, String str, int i)	moviePage()에서 먼저 호출되며 내용을 구성할 window 패널, 검색 종류 설명문, 검색 종류 구분번호를 인수로 받아 검색 패널을 구성

setUpdateTableData(JTable Table, int i)	검색버튼을 누를 때마다 호출되며 바뀔 Table과 검색 종류를 인수로 받아 결과를 출력하는 JTable을 업데이트
---	---

③ ReviewGui : 영화 리뷰 gui를 제공하는 클래스

클래스 및 메서드	설 명
reviewPage()	리뷰 전체 page를 완성
writeReview(JPanel p)	리뷰 작성 page를 완성
searchReview(JPanel p)	리뷰 검색 page를 완성
setUpdateTableData(JTable Table)	테이블을 업데이트

④ MainGui : 생성자를 통해서 window 창을 생성하는 클래스

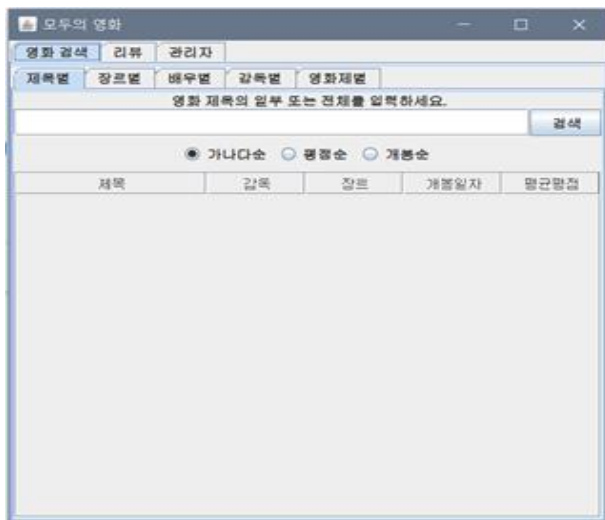
클래스 및 메서드	설 명
windowClosing(WindowEvent e)	connection을 끊고 창을 닫음

3) main

① Main : 메인 함수를 호출하는 클래스

5. 응용프로그램 상세 정보

1) 사용방법

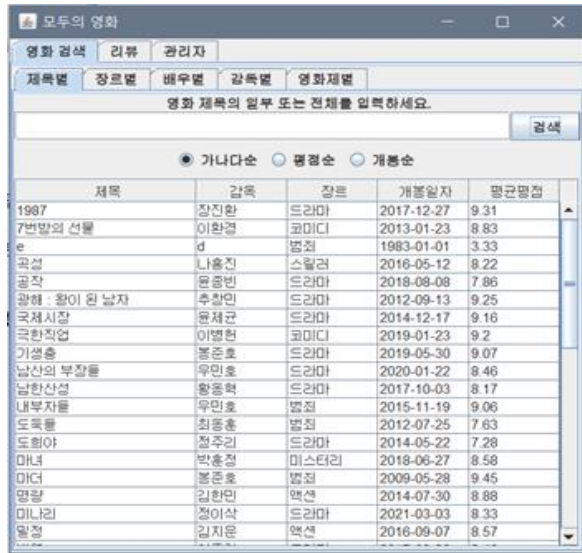


개발환경과 동일하게 프로그램 실행 환경을 구성한다.

프로그램을 실행시키면 [그림 4]와 같은 화면을 처음 마주하게 된다. 사용자는 자신이 이용하고 싶은 기능에 따라 상단의 ‘영화 검색’ 또는 ‘리뷰’ 탭을 눌러 해당 화면으로 이동할 수 있다.

기본 화면으로는 영화 검색의 제목별 검색 화면이 나타난다.

[그림 4] 프로그램 첫 실행 화면



[그림 5] 영화 전체 목록 검색

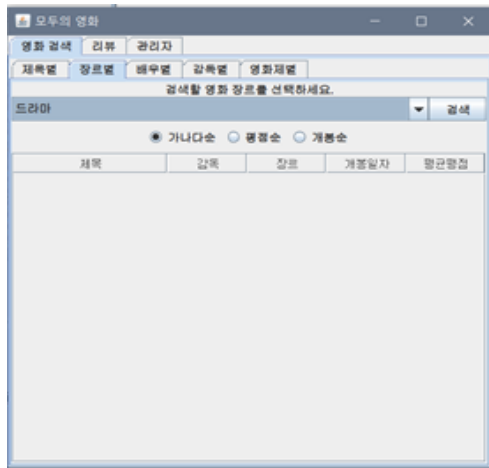
검색 방법은 총 다섯 종류로 제목별, 장르별, 배우별, 감독별, 영화제별로 자신이 원하는 정보에 따라 검색을 해볼 수 있다. 5가지 탭 중에서 원하는 메뉴를 클릭한 뒤, 상단의 textField에 키워드를 입력한 후 검색 버튼을 눌러 검색을 진행할 수 있다.

데이터 베이스에 저장되어 있는 모든 영화에 대한 정보를 알고 싶을 경우에는 textField에 공백을 입력하여 검색을 하면 된다. 결과는 [그림5]와 같다.

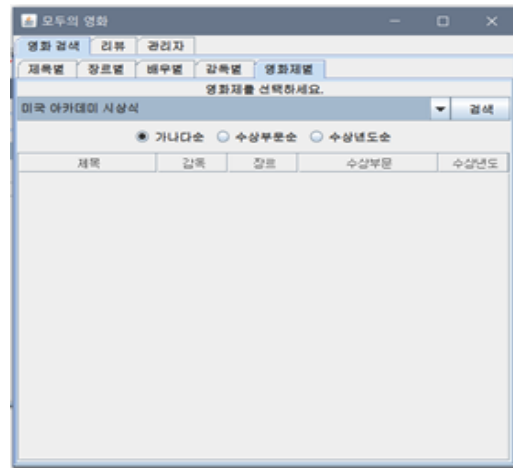
또한 검색 시, 검색 결과에 대하여

가나다순, 평점순, 개봉순으로 결과 정렬 방법 또한 선택할 수 있다.

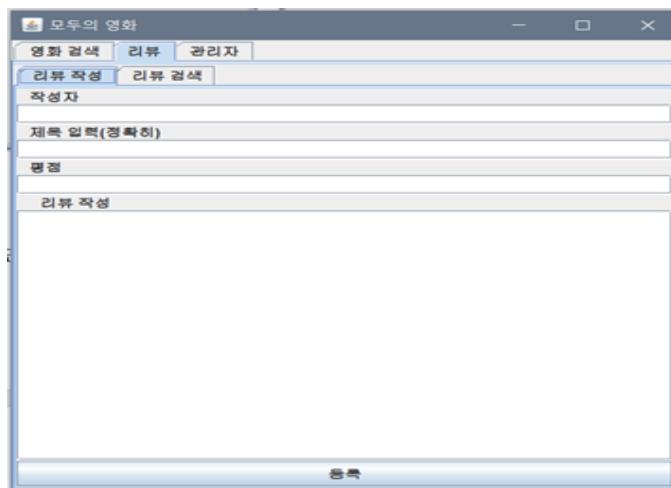
장르별 검색과 영화제별 검색의 경우 textField에 키워드 입력 대신 주어진 목록에서 선택하여 [그림 6], [그림 7]과 같이 편리하게 검색할 수 있다.



[그림 6] 영화 장르 선택 가능



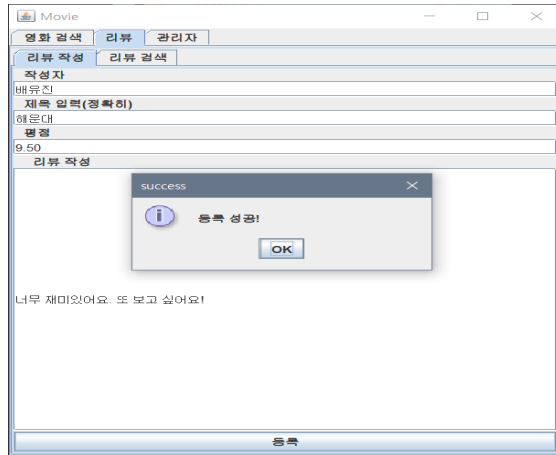
[그림 7] 영화제 선택 가능



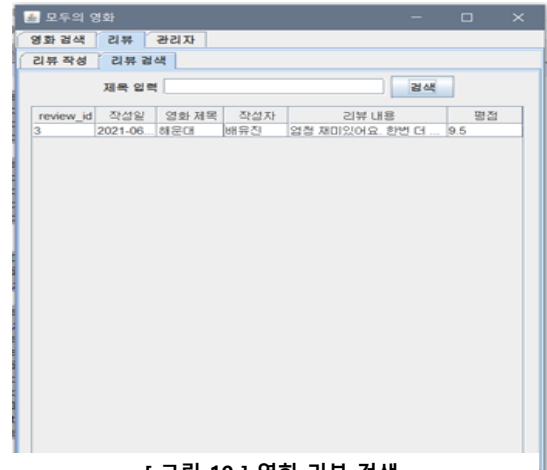
[그림 8] 리뷰 관련 기능

다음은 리뷰 탭에 대한 사용 방법 설명이다. [그림 8]과 같이 상단의 두번째 탭인 '리뷰' 탭을 눌러 영화 리뷰와 관련된 기능을 사용할 수 있다. 이 기능을 사용하여 사용자는 리뷰를 작성하거나 이미 다른 사람들에게 의해 작성되어 있는 리뷰를 확인할 수 있다.

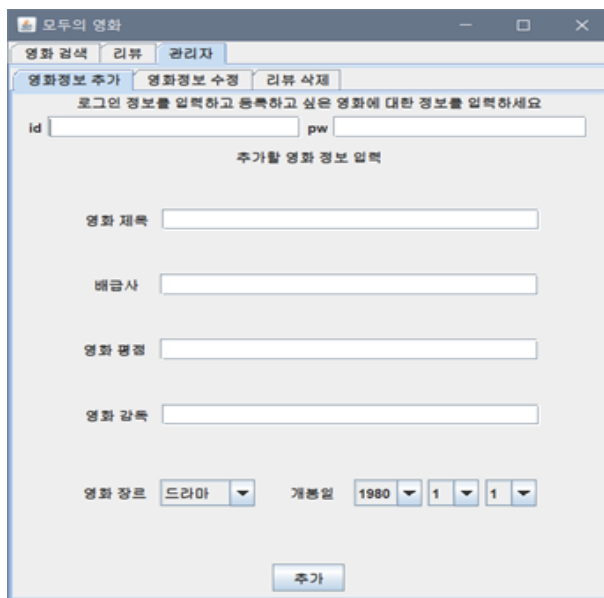
[그림 9]와 같이 작성자, 영화 제목, 평점을 입력하고 영화에 대한 리뷰를 작성하면 된다. 이때, 영화의 제목은 정확하게 입력해야 할 것을 유의해야 한다. 리뷰 등록이 잘 되었을 경우에는 다음과 같은 등록 성공 팝업창이 뜬다. 이 때 작성한 리뷰는 [그림 10]과 같이 리뷰 검색 화면에서 확인할 수 있다. 영화 제목을 입력하여 검색 시 원하는 영화의 리뷰만 확인 가능하다.



[그림 9] 영화 리뷰 작성 후 등록



[그림 10] 영화 리뷰 검색



[그림 11] 관리자 화면

마지막은 관리자 탭에 대한 사용 방법이다. 관리자 화면은 [그림 11]과 같다. 관리자 탭은 영화정보 추가, 영화정보 수정, 리뷰 삭제의 세 가지 탭으로 구성되어 있으며 각 탭을 선택하여 해당 기능을 수행할 수 있는 화면으로 넘어갈 수 있다. 관리자 탭은 앞서 설명했던 영화 검색, 리뷰 탭과 같은 사용자를 위한 기능과 구분하여 관리자만을 위한 기능이므로 아이디와 비밀번호를 입력해야 하는 로그인 화면이 항상 존재하고 관리자에 맞는 아이디와 비밀번호를 입력해야만 해당

기능들을 수행할 수 있다. 영화 정보를 추가하고 싶다면 아이디와 비밀번호를 올바르게 입력한 뒤, 영화 제목, 배급사, 영화 평점, 영화 감독을 입력하고 영화 장르와 개봉일은 목록에서 선택한 뒤 추가 버튼을 누른다면 새로운 영화 정보를 손쉽게 추가할 수 있다.

[그림 12] 영화 정보 수정

영화 정보를 수정하고 싶을 때에는 두 번째 탭인 영화정보 수정 탭을 누르면 된다. 해당 탭을 누르게 된다면 [그림 12]와 같은 화면이 나타나게 된다. 영화 정보 추가 탭과 동일하고 아이디와 패스워드를 입력한 뒤, 영화 제목과 배급사를 입력하고 영화 장르와 개봉일은 목록에서 선택한 뒤 수정 버튼을 눌러주면 된다. 이때, 영화 제목은 정확하게 입력해줘야 한다.

관리자 탭의 마지막 기능인 리뷰 삭제 탭이다. 리뷰 삭제 탭을 눌렀을 시 나오는 화면은 [그림 13]과 같다. 관리자 기능이기에 리뷰 삭제 탭 또한 아이디와 패스워드를 입력하여 로그인을 해야 한다. 키워드 입력 칸에 삭제하고 싶은 키워드를 입력한 뒤 삭제 버튼을 누르게 되면 하단의 입력 대기중이 사라지고 [그림 14]와 같이 삭제 완료가 되었다는 팝업창이 떠 성공적으로 리뷰가 삭제되었음을 보여준다. 이 기능을 통해 관리자는 특정 키워드가 포함되는 리뷰를 삭제할 수 있다. 특정 키워드를 필터링하여 부적절한 단어를 포함하거나 비속어를 사용한 리뷰를 삭제하는 효과를 가진다.

[그림 13] 리뷰 삭제 탭

[그림 14] 리뷰 삭제 완료

2) 기능 설명

본 프로젝트는 해당 프로젝트의 기본 요구사항 (1) ~ (17) 항목을 모두 만족한다. (프로젝트 설명 안내 워드 문서 참고)

A. (1), (3) 만족 - 5개 이상의 테이블을 가지고 각 테이블들의 컬럼의 수를 합하면 20개 이상이며, 기본 키, 외래 키, not null 제약 조건을 포함한다.

```
mysql> use db2021team05
Database changed
mysql> show tables:
+-----+
| Tables_in_db2021team05 |
+-----+
| db2021_actor            |
| db2021_director        |
| db2021_movie            |
| db2021_movie_cast      |
| db2021_movie_direction |
| db2021_nominations     |
| db2021_review          |
| db2021_viewmovie       |
| db2021_viewreview      |
+-----+
```

[그림 15] 테이블 목록

데이터 베이스의 테이블은 [그림 15]를 보면 알 수 있듯이 DB2021_actor, DB2021_director, DB2021_movie, DB2021_movie_cast, DB2021_movie_direction, DB2021_nominations, DB2021_review로 구성되어 있으며 총 7개의 테이블을 가지고 있다.

각 테이블을 살펴보면 [그림 16] ~ [그림 22]와 같다. 각 테이블들의 컬럼 수를 합해보면 28개의 컬럼을 가지고 있음을 알 수 있다. 또한, 각각, not null 제약 조건, 외래키, 기본키를 사용하였다.

```
mysql> desc db2021_actor;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| Actor_ID | int          | NO   | PRI | NULL    | auto_increment |
| Name     | varchar(30)  | NO   |     | NULL    |                |
| Sex      | enum('M','F') | YES  |     | NULL    |                |
| Born     | date         | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
```

[그림 16] DB2021_actor 테이블

```
mysql> desc db2021_director;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| Director_ID | int          | NO   | PRI | NULL    | auto_increment |
| Name       | varchar(30)  | YES  |     | NULL    |                |
+-----+-----+-----+-----+-----+-----+
```

[그림 17] DB2021_director 테이블

```
mysql> desc db2021_movie;
+-----+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| Movie_ID | int          | NO   | PRI | NULL    | auto_increment |
| Title    | varchar(50)  | NO   |     | NULL    |                |
| Genre    | enum('드라마', '미스터리', '스릴러', '멜로', '범죄', '액션', '코미디', '판타지') | YES  |     | NULL    |                |
| ReleaseDate | date        | YES  |     | NULL    |                |
| Production | varchar(30) | YES  |     | NULL    |                |
| Score    | float        | YES  |     | NULL    |                |
| Review_cnt | int         | NO   |     | 100     |                |
+-----+-----+-----+-----+-----+-----+
```

[그림 18] DB2021_movie 테이블

```
mysql> desc db2021_movie_cast;
```

Field	Type	Null	Key	Default	Extra
Movie_ID	int	NO	PRI	NULL	
Actor_ID	int	NO	PRI	NULL	
Role	enum('주연', '조연')	YES		NULL	

[그림 19] DB2021_movie_cast 테이블

```
mysql> desc db2021_movie_direction;
```

Field	Type	Null	Key	Default	Extra
Movie_ID	int	NO	PRI	NULL	
Director_ID	int	NO	PRI	NULL	

[그림 20] DB2021_movie_direction 테이블

```
mysql> desc db2021_nominations;
```

Field	Type	Null	Key	Default	Extra
Movie_ID	int	NO	PRI	NULL	
Year	int	NO	PRI	NULL	
Award	varchar(30)	NO	PRI	NULL	
Prize	varchar(30)	NO	PRI	NULL	

[그림 21] DB2021_nominations 테이블

```
mysql> desc db2021_review;
```

Field	Type	Null	Key	Default	Extra
Review_ID	int	NO	PRI	NULL	auto_increment
Movie_ID	int	NO	MUL	NULL	
Date	date	YES		NULL	
Score	float	YES		NULL	
Writer	varchar(20)	NO		NULL	
Review	varchar(100)	YES		NULL	

[그림 22] DB2021_review 테이블

B. (2) 만족 – 초기화를 위해 적어도 30개의 레코드(투플)을 가지고 있어야 한다. 모든 테이블의 레코드 수의 총합이 30개 이상이다.

```
mysql> select count(*) from db2021_actor;
```

count(*)
131

```
1 row in set (0.00 sec)
```

```
mysql> select count(*) from db2021_director;
```

count(*)
34

```
1 row in set (0.01 sec)
```

```
mysql> select count(*) from db2021_movie;
```

count(*)
45

[그림 23] 레코드 수 확인 (1)

[그림 23], [그림 24]를 참고하면

DB2021_actor는 131개,
 DB2021_director는 34개,
 DB2021_movie는 45개,
 DB2021_movie_cast는 190개,
 DB2021_movie_direction은 45개,

```
mysql> select count(*) from db2021_movie_cast;
+-----+
| count(*) |
+-----+
|      190 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from db2021_movie_direction;
+-----+
| count(*) |
+-----+
|       45 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from db2021_nominations;
+-----+
| count(*) |
+-----+
|       64 |
+-----+
1 row in set (0.00 sec)

mysql> select count(*) from db2021_review;
+-----+
| count(*) |
+-----+
|        0 |
+-----+
1 row in set (0.01 sec)
```

[그림 24] 레코드 수 확인 (2)

DB2021_nominations는 64개,
DB2021_review는 0개로 총 509개의
투표를 가지고 있음을 확인 할 수 있다.

C. (4), (8) 만족 - 최소 2개의 뷰를 정의하고 뷰를 사용하는 쿼리를 포함해야 한다.

```
mysql> show tables;
+-----+
| Tables_in_db2021team05 |
+-----+
| db2021_actor            |
| db2021_director         |
| db2021_movie            |
| db2021_movie_cast       |
| db2021_movie_direction  |
| db2021_nominations      |
| db2021_review           |
| db2021_viewmovie        |
| db2021_viewreview       |
+-----+
```

[그림 25] 뷰 정의 확인

[그림 25]에서 알 수 있듯이
DB2021_viewmovie, DB2021_viewreview를
사용하여 총 2개의 뷰를 정의했다.

< DB2021_viewmovie >

DB2021_movie - direction 테이블에서 movie_id에 대한 director_id를 가져오고 그것으로
director의 이름을 DB2021_director 테이블에서 가져온다.

최종적으로 영화 감독과 영화 정보를 사용자에게 뷰를 통해서 제공할 수 있다. 이 부분에서는
[그림 26] 과 같이 복잡한 코드를 view로 간단히 정의하기 위해서 view를 사용하였다.

```
CREATE VIEW DB2021_ViewMovie AS                                #영화 검색 결과 출력을 위한 view 정의
SELECT Title, Director, Genre, ReleaseDate, Score
FROM DB2021_Movie JOIN (SELECT Movie_ID, Director_ID, DB2021_Director.Name AS Director
                        FROM DB2021_Movie_direction JOIN DB2021_Director
                        USING(Director_ID)) T
USING(Movie_ID);
```

[그림 26] 영화 검색 결과 출력을 위한 view 정의

코드에서는 [그림 27] 과 같이 사용하였고 총 9개의 쿼리에서 사용하였다.

```
sql = "select * " + " from DB2021_ViewMovie" + " where Title like ?"
      + " order by Title ASC;";
String sql1 = "SELECT * FROM DB2021_ViewMovie WHERE Director LIKE ?";
```

[그림 27] view를 사용한 SQL 코드

< DB2021_viewreview >

리뷰와 영화 테이블에서 영화별 리뷰를 쓴 사람의 수(평균을 구하기 위해 개발자가 사용), 영화별 사용하지 않는 정보 등 불필요한 attribute를 사용자에게 제공하지 않고 필요한 정보만 제공하기 위해 [그림 28] 과 같이 view를 사용하였다.

```
CREATE VIEW DB2021_ViewReview AS                                #리뷰 출력을 위한 view 정의
SELECT Review_ID, Title, Writer, Date, DB2021_Review.Score AS sc, Review
FROM DB2021_Review JOIN DB2021_Movie
USING (Movie_ID);
```

[그림 28] 리뷰 출력을 위한 view 정의

코드에서는 [그림 28] 과 같이 사용하였다.

```
String sql="select Review_ID,DATE,Title,Writer,Review,sc"
          + " from DB2021_ViewReview"
          + " where Title like ?"
          + " order by DATE;";
```

[그림 29] view를 사용한 SQL 코드

D. (5) 만족 – 모든 테이블과 뷰의 이름들은 ‘DB2021_’라는 접두어를 가지고 있어야 한다.

따라서 [그림 30] 과 같이 테이블 이름과 뷰 이름을 정의하였다.

```
mysql> show tables;
+-----+
| Tables_in_db2021team05 |
+-----+
| db2021_actor            |
| db2021_director        |
| db2021_movie            |
| db2021_movie_cast      |
| db2021_movie_direction |
| db2021_nominations      |
| db2021_review           |
| db2021_viewmovie       |
| db2021_viewreview      |
+-----+
```

[그림 10] 접두어 사용

E. (6), (7) 만족 – 적어도 4개의 인덱스를 정의하고 이를 사용하는 쿼리들을 포함해야 한다.

```
CREATE INDEX idx_movie ON DB2021_Movie ( Movie_ID );
CREATE INDEX idx_movie ON DB2021_Movie ( Title );
CREATE INDEX idx_director ON DB2021_Director ( director_ID );
CREATE INDEX idx_director ON DB2021_Director ( name );
CREATE INDEX idx_actor ON DB2021_actor ( Actor_ID );
```

[그림 31] 인덱스 설정

투플을 탐색할 때 많이 사용하는 primary key(ID)와 검색이 되는 attribute(name, Title)를 index로 설정하였다. 여기서 name은 감독의 이름을 의미한다.

```
CREATE VIEW DB2021_ViewMovie AS                                #영화 검색 결과 출력을 위한 view 정의
SELECT Title, Director, Genre, ReleaseDate, Score
FROM DB2021_Movie JOIN (SELECT Movie_ID, Director_ID, DB2021_Director.Name AS Director
                        FROM DB2021_Movie_direction JOIN DB2021_Director
                        USING(Director_ID)) T
                        USING(Movie_ID);

String sql1 = "SELECT * FROM DB2021_ViewMovie WHERE Director LIKE ?";
sql = "select * " + " from DB2021_ViewMovie" + " where Title like ?"
      + " order by Title ASC;";
"(SELECT Title, Award, Prize, YEAR "
"FROM DB2021_Movie JOIN DB2021_Nominations "
"USING(Movie_ID))"
"SELECT * "
"FROM DB2021_ViewMovie JOIN A "
"USING(Title) "
"WHERE Award= ?";
String sql1 = "WITH A AS (SELECT Movie_ID, DB2021_Actor.Name AS Actor, Role FROM DB2021_Movie_cast JOIN DB2021_Actor USING(Actor_ID) )"+
" SELECT * FROM DB2021_ViewMovie JOIN (SELECT Title, Actor, Role FROM A JOIN DB2021_Movie USING(Movie_ID)) S"+
" USING(Title) WHERE Actor LIKE ?";
```

[그림 32] 인덱스 사용 쿼리

F. (9) 만족 – 트랜잭션을 포함해야 한다.

```
public int insert_movie(String title, String genre, String date, String production, String score, String director) {
    Connection conn = null;
    PreparedStatement pstmt1 = null;
    PreparedStatement pstmt2 = null;
    PreparedStatement pstmt3 = null;
    int f=0;

    try {
        conn = DBConn.getConnection();
        String sql1 = "insert into DB2021_Movie (title, genre, Releasedate, production, score, review_cnt) values(?,?,?,?,?,100)";
        String sql2 = "insert into DB2021_Director (name) values(?)";
        String sql3 = "insert into DB2021_Movie_direction (movie_id, director_id) values(?, ?)";
        pstmt1 = conn.prepareStatement(sql1);
        pstmt2 = conn.prepareStatement(sql2);
        pstmt3 = conn.prepareStatement(sql3);

        conn.setAutoCommit(false);
        pstmt1.setString(1, title);
        pstmt1.setString(2, genre);
        pstmt1.setString(3, date);
        pstmt1.setString(4, production);
        pstmt1.setString(5, score);
        pstmt1.executeUpdate();
        if (find_director_id(director)==-1) {
            pstmt2.setString(1, director);
            pstmt2.executeUpdate();
        }
        pstmt3.setInt(1, find_movie_id(title));
        pstmt3.setInt(2, find_director_id(director));
        pstmt3.executeUpdate();
        conn.commit();
    } catch (SQLException e) {
        e.printStackTrace();
        try {
            if(conn!=null) //오류가 생기고 연결되어있으면 rollback() 적용
                conn.rollback();
        } catch (SQLException se2) {
            se2.printStackTrace();
        }
    } finally { //계속 같은 DB에 연결함으로 conn은 시스템이 끝날 경우 닫아준다.
        try {
            pstmt1.close();
            pstmt2.close();
            pstmt3.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    return 1;
}
```

[그림 33] 첫번째 트랜잭션

```

public int delete_review(String keyword) {
    if (keyword.equals(""))
        return 2;
    //////////////////////////////////////
    Connection conn = null; //connection 객체 생성
    PreparedStatement pstmt = null; //SQL 등록, 실행
    PreparedStatement pstmt0 = null;
    try {
        conn=DBConn.getConnection();
        System.out.println(keyword);
        String sql0="update DB2021_movie as m ,DB2021_Review as r "
            + "set review_cnt=review_cnt-1 ,m.score=(m.score*review_cnt-r.score)/(review_cnt-1) "
            + "where m.movie_id=r.movie_id and review like ?";
        String sql ="delete from DB2021_Review review where review like ?";
        pstmt0=conn.prepareStatement(sql0);
        pstmt=conn.prepareStatement(sql);

        conn.setAutoCommit(false);
        pstmt0.setString(1, "%" + keyword + "%");
        pstmt.setString(1, "%" + keyword + "%");
        pstmt0.executeUpdate();
        pstmt.executeUpdate();
        conn.commit();

    } catch (SQLException e) {
        e.printStackTrace();
        try {
            if(conn!=null) //오류가 생기고 연결되어있으면 rollback()적용
                conn.rollback();
        } catch (SQLException se2) {
            se2.printStackTrace();
        }
    }
    } finally { //계속 같은 DB에 연결함으로 conn은 시스템이 끝날 경우 알아준다.
        try {
            conn.setAutoCommit(true);
            pstmt.close();
            pstmt0.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}

return 1;
}

```

[그림 34] 두번째 트랜잭션

```

public int insert_update_reviewScore(String tf1,String tf2,String tf3,String tf4) { //평점,작성자,영화제목,리뷰
    if(tf4.equals("") || tf3.equals("") || tf2.equals("")||tf1.equals("")) return -2;
    Connection conn = null; //connection 객체 생성
    PreparedStatement pstmt1 = null; //SQL 등록, 실행
    PreparedStatement pstmt2 = null;
    PreparedStatement pstmt3 = null;
    PreparedStatement pstmt4 = null;
    ResultSet rs = null;
    int mv_id;
    try {
        conn=DBConn.getConnection();
        String sql1="insert into DB2021_Review (movie_id,date,score,writer,review) values(?,?,?,?);"; //영화id,date,평점, 작성자, 리뷰
        String sql2="select score,review_cnt from db2021_movie where movie_id=?";
        String sql3="update db2021_movie set score=? where movie_id=?";
        String sql4="update db2021_movie set review_cnt=? where movie_id=?";
        mv_id=find_movie_id(tf3);
        if(mv_id==-1) return -1;
        pstmt1=conn.prepareStatement(sql1);
        pstmt2=conn.prepareStatement(sql2);
        pstmt3=conn.prepareStatement(sql3);
        pstmt4=conn.prepareStatement(sql4);
        SimpleDateFormat f=new SimpleDateFormat("yyyy-MM-dd");

        pstmt2.setInt(1,mv_id );
        System.out.println(pstmt2);
        rs=pstmt2.executeQuery();
        rs.next();
        float score=rs.getFloat("Score");
        int review_cnt=rs.getInt("review_cnt");
        score=(Float.parseFloat(tf1)+score*review_cnt)/(review_cnt+1);
        //트랜잭션 시작
        conn.setAutoCommit(false);
        pstmt1.setInt(1, mv_id);
        pstmt1.setString(2,f.format(new Date()));
        pstmt1.setFloat(3,Float.parseFloat(tf1));
        pstmt1.setString(4,tf2);
        pstmt1.setString(5, tf4);
        pstmt3.setFloat(1,score);
        pstmt3.setInt(2,mv_id);
        pstmt4.setInt(1, review_cnt+1);
        pstmt4.setInt(2,mv_id);

        pstmt1.executeUpdate();
        pstmt3.executeUpdate();
        pstmt4.executeUpdate();
        conn.commit();
    } catch (SQLException e) {
        e.printStackTrace();
        try {
            if(conn!=null) //오류가 생기고 연결되어있으면 rollback()적용
                conn.rollback();
        } catch (SQLException se2) {
            se2.printStackTrace();
        }
    } finally {
        try {
            conn.setAutoCommit(true);
            rs.close();
            pstmt1.close();
            pstmt2.close();
            pstmt3.close();
            pstmt4.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    return 1;
}

```

[그림 35] 세번째 트랜잭션

G. (10), (11) 만족 – 중첩된 쿼리들과 조인쿼리들을 가지는 쿼리들을 포함해야 한다.

```

String sql = "";
String sql1 = "WITH A AS (SELECT Movie_ID, DB2021_Actor.Name AS Actor, Role FROM DB2021_Movie_cast JOIN DB2021_Actor USING(Actor_ID) )"+
    " SELECT * FROM DB2021_ViewMovie JOIN (SELECT Title, Actor, Role FROM A JOIN DB2021_Movie USING(Movie_ID)) S"+
    " USING(Title) WHERE Actor LIKE ?";

if (order == "Title")
    sql = sql1 + " order by Title ASC;";
else if (order == "Score")
    sql = sql1 + " order by Score DESC;";
else
    sql = sql1 + " order by releaseDate DESC;";

```

```

CREATE VIEW DB2021_ViewMovie AS                                     #영화 검색 결과 출력을 위한 view 정의
SELECT Title, Director, Genre, ReleaseDate, Score
FROM DB2021_Movie JOIN (SELECT Movie_ID, Director_ID, DB2021_Director.Name AS Director
                        FROM DB2021_Movie_direction JOIN DB2021_Director
                        USING(Director_ID)) T
                        USING(Movie_ID);

```

[그림 36] 중첩 쿼리와 조인 쿼리

H. (12) 만족 – 매개 변수를 가지면서 동적으로 만드는 쿼리를 포함해야 한다.

대부분의 쿼리들이 사용자로부터 입력 값을 받고 사용자가 입력한 값으로 쿼리를 생성하도록 만들었다.

```

public ArrayList<ReviewClass> search_review(String s) { //제목을 받아서 리뷰를 보여준다.
    Connection conn = null; //connection 객체 생성
    PreparedStatement pstmt = null; //SQL 등록, 실행
    ResultSet rs = null; //DB 결과값 받을 공간
    mlist=null;
    try {
        conn=DBConn.getConnection();
        String sql="select Review_ID,DATE,Title,Writer,Review,sc"
            + " from DB2021_ViewReview"
            + " where Title like ?"
            + " order by DATE;";
        pstmt=conn.prepareStatement(sql);
        pstmt.setString(1,"%"+s+"%"); //해당 단어가 들어가는 영화의 모든 리뷰를 보여준다.
        rs=pstmt.executeQuery();
        mlist=new ArrayList<ReviewClass>();
        while(rs.next()) {
            ReviewClass rc = new ReviewClass();
            System.out.println(rs.getInt("Review_ID"));
            rc.review_id=rs.getInt("Review_ID");
            rc.date=rs.getDate("DATE");
            rc.title=rs.getString("Title");
            rc.writer=rs.getString("Writer");
            rc.review=rs.getString("Review");
            rc.score=rs.getFloat("sc");
            mlist.add(rc);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally { //계속 같은 DB에 연결하므로 conn은 시스템이 끝날 경우 닫아준다.
        try {
            rs.close();
            pstmt.close();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    return mlist;
}

```

[그림 37] 매개변수 갖는 동적 쿼리

I. (13) 만족 – 그래픽 또는 문자 기반의 사용자 인터페이스를 사용해야 하며 사용하기 쉽고 사용하기에 도움이 되는 정보를 가지고 있는 메뉴를 제공해야 한다.

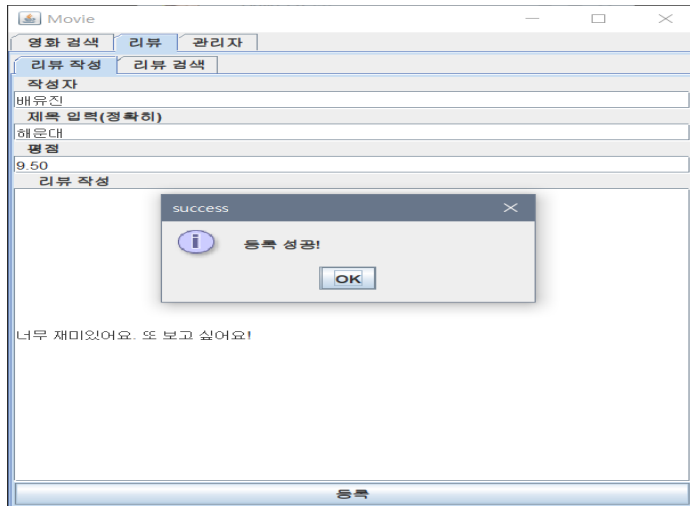
그래픽 기반의 인터페이스를 사용해서 응용프로그램을 만들었으며 이에 대한 구체적인 설명은 5-1) 사용방법에서 충분한 정보를 제공하여 생각한다.

J. (14) ~ (17) 만족 - 데이터베이스에 삽입, 갱신, 삭제, 검색을 하기 위한 인터페이스와 쿼리를 가지고 있어야 한다.

다음은 많은 쿼리를 중 삽입, 갱신, 삭제, 검색 기능을 1개씩 보여주고 있다.

```
String sql1="insert into DB2021_Review (movie_id,date,score,writer,review) values(?,?,?,?);";
String sql2="select score,review_cnt from db2021_movie where movie_id=?";
String sql3="update db2021_movie set score=? where movie_id=?";
String sql ="delete from DB2021_Review review where review like ?";
```

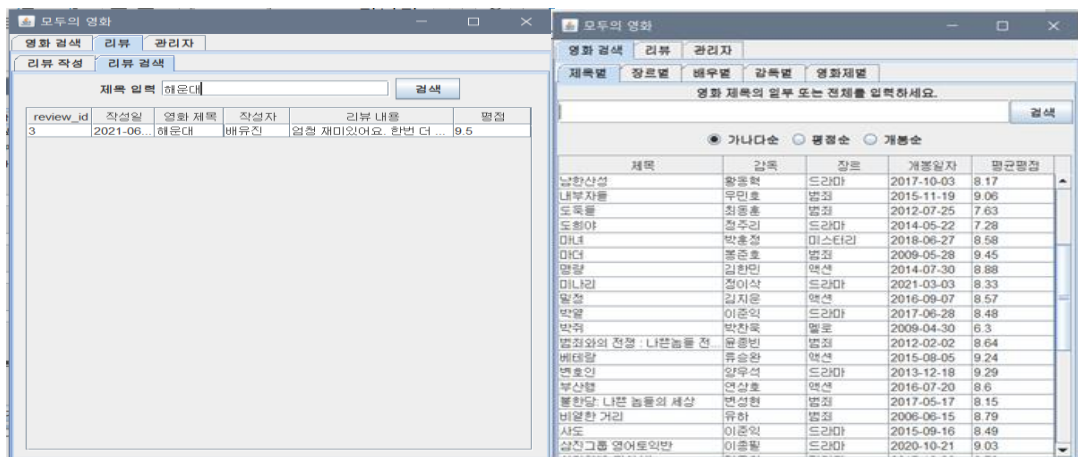
[그림 38] 삽입, 갱신, 삭제, 검색 쿼리



[그림 39] 리뷰 작성 인터페이스

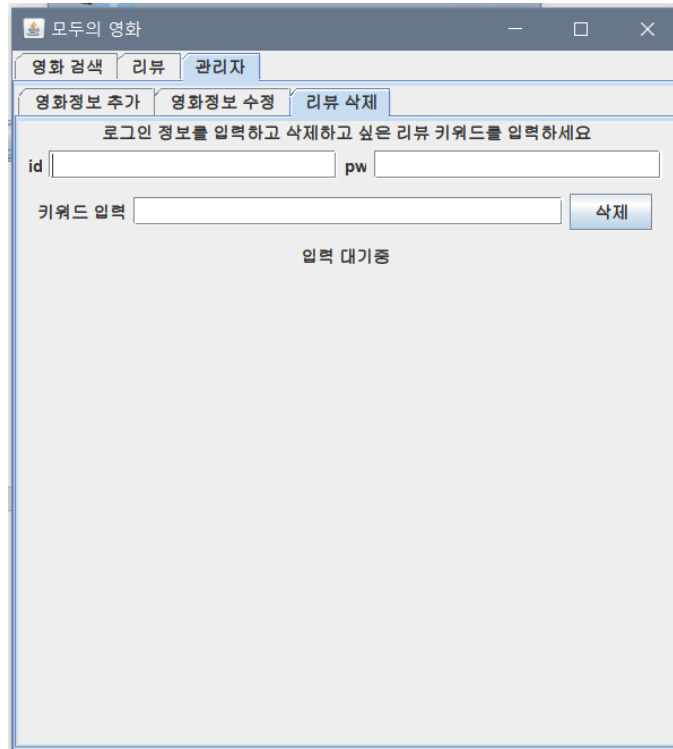
다음은 리뷰를 쓸 수 있는 인터페이스이다. 리뷰를 새로 추가할 때 insert문을 사용했고 영화의 평점을 검색할 때 select문을 사용했고 평점을 새로 업데이트 할 때 update문을 사용하였다.

새로운 리뷰 등록을 통하여 업데이트된 평점을 확인할 수 있다.



[그림 40] 업데이트 된 평점 확인

다음은 키워드에 따라 리뷰를 삭제할 수 있는 인터페이스이다. 여기서 delete문을 사용했다.



[그림 41] 리뷰 삭제 인터페이스

6. 강점 및 편리성

1) 다양한 기능을 제공하는 그래픽 인터페이스를 통해 사용자와 관리자 모두가 편리하게 프로그램을 사용할 수 있게 만들었다.

- A. 다양한 검색 방법과 정렬 방법이 존재하여 사용자가 원하는 방식으로 영화를 검색해 볼 수 있다.
- B. 관리자가 영화 정보를 추가하거나 수정할 때 장르와 날짜 선택이 텍스트 기반이 아닌 버튼을 누른 방식이기 때문에 오류 가능성이 적다.
- C. 영화를 검색할 때 전체 키워드가 아니라 일부만 작성해도 해당 키워드를 포함하는 결과를 자동으로 보여줘서 사용자의 검색을 더욱 용이하게 한다.
- D. 사용자가 리뷰를 작성하며 매긴 평점은 영화 정보의 평균 평점에 즉시 반영된다. 즉, 실시간으로 평점 정보를 반영하여 보다 정확한 영화에 대한 정보를 제공한다.

2) 단순히 영화 정보 제공에서 끝나지 않고 리뷰를 공유할 수 있는 플랫폼까지 제공한다.

- A. 영화에 대한 자신의 소감을 응용 프로그램을 통해 다른 사람들과 공유할 수 있다.
- B. 다른 사람들의 리뷰 또한 검색이 가능하므로 영화 선택에 있어서 더욱 도움이 된다.

3) 사용자와 관리자로 기능이 구분되어 있어 보다 효율적인 프로그램의 관리가 이루어질 수 있다.

A. 영화 정보 추가 및 수정, 리뷰 삭제는 관리자 권한으로만 가능하기 때문에 프로그램의 데이터를 효율적으로 관리할 수 있다.

B. 리뷰 필터링 기능을 제공하여 관리자는 특정 키워드가 포함되는 리뷰를 검색하고 삭제할 수 있어 더욱더 편리하게 불건전한 리뷰를 필터링 할 수 있다.

7. 역할 분담

팀원	SQL	Java Code	Report	발표	데모 비디오
박소현	Create table 스크립트 작성 Insert table 스크립트 작성 ER diagram 작성	AdminDB AdminGui	프로젝트 개요, 업무 기능도, 개념적 데이터 모델, 클래스 및 메서드 설명, 레포트 최종정리	발표	
배유진	Create table 스크립트 작성 Insert table 스크립트 작성 schema diagram 작성	MovieDB MovidGui	논리적 데이터 모델, 응용프로그램 사용방법, 강점, 클래스 및 메서드 설명	피피티 제작, 발표	
이소울	Create table 스크립트 작성 Insert table 스크립트 작성	ReviewDB ReviewGui	응용 프로그램 기능 설명, 클래스 및 메서드 설명	발표	환경 구축, 녹화