

데이터베이스설계

2020 설계 프로젝트

보고서 작성 서약서

1. 나는 타학생의 보고서를 베끼거나 여러 보고서의 내용을 짜집기하지 않겠습니다.

2. 나는 보고서의 주요 내용을 인터넷사이트 등을 통해 얻지 않겠습니다.

3. 나는 보고서의 내용을 조작하지 않겠습니다.

4. 나는 보고서 작성에 참고한 문헌의 출처를 밝히겠습니다.

5. 나는 나의 보고서를 제출 전에 타학생에게 보여주지 않겠습니다.

나는 보고서 작성시 윤리에 어긋난 행동을 하지 않고 정보통신공학인으로서 나의 명예를 지킬 것을 맹세합니다.

2020년 12월 13일

학부 정보통신공학과

학년 3

성명 김지후

학번 12181758



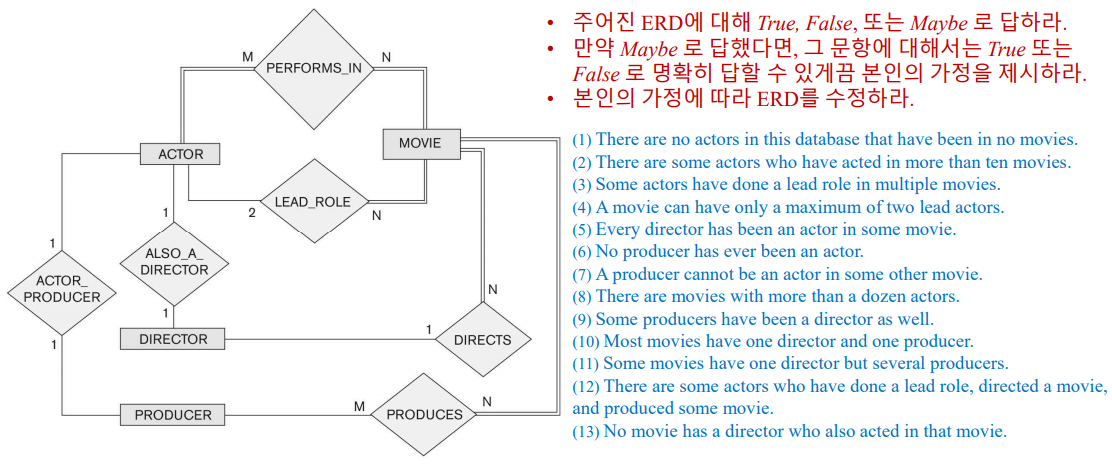
1. 개요

1) MOVIES ERD를 기반으로 MOVIES 데이터베이스를 설계한다.

2) 이를 MySQL에 데이터베이스를 구현한 후 web application을 작성한다.

1. 상세 설계 내용

1) ERD 설계



<MOVIES 데이터베이스 ERD>

(1) There are no actors in this database that have been in no movies. **TRUE**

(2) There are some actors who have acted in more than ten movies. **MAYBE->TRUE**

(3) Some actors have done a lead role in multiple movies. **TRUE**

(4) A movie can have only a maximum of two lead actors. **TRUE**

(5) Every director has been an actor in some movie. **FALSE**

(6) No producer has ever been an actor. **FALSE**

(7) A producer cannot be an actor in some other movie. **FALSE**

(8) There are movies with more than a dozen actors. **MAYBE->TRUE**

(9) Some producers have been a director as well. **TRUE**

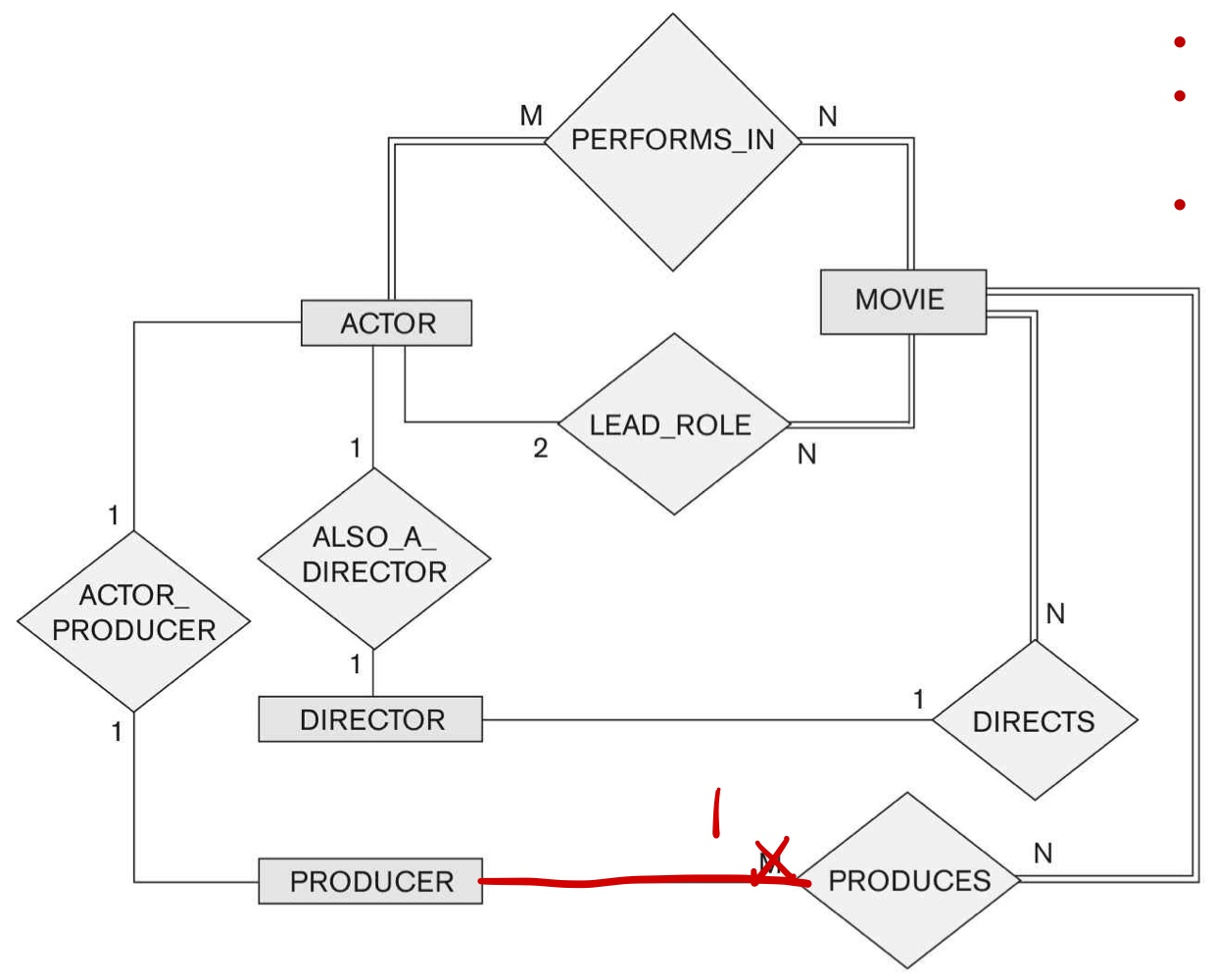
(10) Most movies have one director and one producer. **MAYBE->TRUE**

(11) Some movies have one director but several producers. **TRUE**

(12)There are some actors who have done a lead role, directed a movie,and produced some movie. **TRUE**

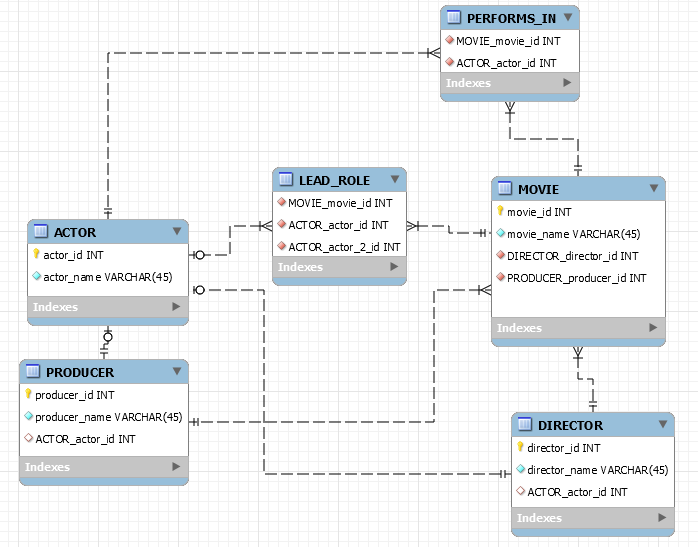
(13) No movie has a director who also acted in that movie. **FALSE**

**-위 가정을 바탕으로 수정된 ERD**



위 가정 10번에서 거의 모든 영화는 디렉터 1명, 프로듀서 1명으로 구성되어 있다고 가정하였으므로 PRODUCES 관계를 M:N이 아닌 1:N으로 수정하였다.

-MySQL에서 작성한 EER 다이어그램



**-MOVIES 데이터베이스를 MySQL에 생성하기**

mysql> create table actor(

-> actor\_id int not null,

-> actor\_name varchar(45) not null,

-> primary key(actor\_id));

Query OK, 0 rows affected (0.12 sec)

mysql> create table director(

-> director\_id int not null,

-> director\_name varchar(45) not null,

-> actor\_actor\_id int,

-> primary key(director\_id),

-> constraint fk\_director\_actor1 foreign key(actor\_actor\_id)

-> references actor(actor\_id) on delete set null on update cascade);

Query OK, 0 rows affected (0.12 sec)

mysql> create table producer(

-> producer\_id int not null,

-> producer\_name varchar(45) not null,

-> actor\_actor\_id int,

-> primary key(producer\_id),

-> constraint fk\_producer\_actor1 foreign key(actor\_actor\_id)

-> references actor(actor\_id) on delete set null on update cascade);

Query OK, 0 rows affected (0.11 sec)

mysql> create table movie(

-> movie\_id int not null,

-> movie\_name varchar(45) not null,

-> director\_director\_id int not null,

-> producer\_producer\_id int not null,

-> primary key(movie\_id),

-> constraint fk\_movie\_director1 foreign key(director\_director\_id)

-> references director(director\_id) on delete no action on update cascade,

-> constraint fk\_movie\_producer1 foreign key(producer\_producer\_id)

-> references producer(producer\_id) on delete no action on update cascade);

Query OK, 0 rows affected (0.10 sec)

mysql> create table performs\_in

-> movie\_movie\_id int not null,

-> actor\_actor\_id int not null,

-> constraint fk\_movie\_has\_actor\_movie foreign key(movie\_movie\_id)

-> references movie(movie\_id) on delete no action on update cascade,

-> constraint fk\_movie\_has\_actor\_actor foreign key(actor\_actor\_id)

-> references actor(actor\_id) on delete no action on update cascade);

Query OK, 0 rows affected (0.12 sec)

mysql> create table lead\_role(

-> movie\_movie\_id int not null,

-> actor\_actor\_id int not null,

-> actor\_actor\_2\_id int not null,

-> constraint fk\_movie\_has\_lead\_actor\_movie foreign key(movie\_movie\_id)

-> references movie(movie\_id) on delete no action on update cascade,

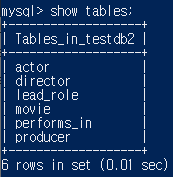
-> constraint fk\_movie\_has\_lead\_actor\_actor foreign key(actor\_actor\_id)

-> references actor(actor\_id) on delete no action on update cascade,

-> constraint fk\_movie\_has\_lead\_actor\_actor2 foreign key(actor\_actor\_2\_id)

-> references actor(actor\_id) on delete no action on update cascade);

Query OK, 0 rows affected (0.13 sec)



총 6개의 테이블이 만들어졌다.

**-CREATE INDEX문**

mysql> create unique index actor\_director on director(actor\_actor\_id asc)visible;

Query OK, 0 rows affected (0.06 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create unique index actor\_producer on producer(actor\_actor\_id asc)visible;

Query OK, 0 rows affected (0.08 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create index directs on movie(director\_director\_id asc)visible;

Query OK, 0 rows affected (0.08 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create index produces on movie(producer\_producer\_id asc)visible;

Query OK, 0 rows affected (0.12 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create index fk\_performs\_in\_actor\_idx on performs\_in(actor\_actor\_id asc)visible;

Query OK, 0 rows affected (0.07 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create index fk\_performs\_in\_movie\_idx on performs\_in(movie\_movie\_id asc)visible;

Query OK, 0 rows affected (0.07 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create index fk\_lead\_role\_actor\_idx on lead\_role(actor\_actor\_id asc)visible;

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create index fk\_lead\_role\_actor2\_idx on lead\_role(actor\_actor\_2\_id asc)visible;

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> create index fk\_lead\_role\_movie\_idx on lead\_role(movie\_movie\_id asc)visible;

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

-insert

-actor insert

mysql> insert actor values(1,'jihoo');

Query OK, 0 rows affected (0.03 sec)

mysql> insert actor values(2,'jihyun);

Query OK, 0 rows affected (0.03 sec)

mysql> insert actor values(3,'jihyuk');

Query OK, 0 rows affected (0.02 sec)

mysql> insert actor values(4,'jihoon');

Query OK, 0 rows affected (0.02 sec)

-director insert

mysql> insert director values(101,'jihoo',1);

Query OK, 0 rows affected (0.02 sec)

mysql> insert director values(102,'minji',null);

Query OK, 0 rows affected (0.02 sec)

mysql> insert director values(103,'jihyuk',3);

Query OK, 0 rows affected (0.02 sec)

-producer insert

mysql> insert producer values(201,'jihoo',1);

Query OK, 0 rows affected (0.01 sec)

mysql> insert producer values(202,'jihyun',2);

Query OK, 0 rows affected (0.02 sec)

mysql> insert producer values(203,'minju',null);

Query OK, 0 rows affected (0.01 sec)

-movie insert

mysql> insert movie values(501,'movie-1',101,201);

Query OK, 0 rows affected (0.01 sec)

mysql> insert movie values(502,'movie-2',101,202);

Query OK, 0 rows affected (0.01 sec)

mysql> insert movie values(503,'movie-3',101,203);

Query OK, 0 rows affected (0.01 sec)

mysql> insert movie values(504,'movie-4',102,202);

Query OK, 0 rows affected (0.02 sec)

mysql> insert movie values(505,'movie-5',102,201);

Query OK, 0 rows affected (0.01 sec)

-lead\_role insert

mysql> insert lead\_role values(501,1,2);

Query OK, 0 rows affected (0.01 sec)

mysql> insert lead\_rolevalues(502,1,3);

Query OK, 0 rows affected (0.01 sec)

mysql> insert lead\_role values(503,2,3);

Query OK, 0 rows affected (0.01 sec)

mysql> insert lead\_role values(504,2,4);

Query OK, 0 rows affected (0.01 sec)

mysql> insert lead\_role values(505,3,4);

Query OK, 0 rows affected (0.01 sec)

-performs\_in insert

mysql> insert performs\_in values(501,1);

Query OK, 0 rows affected (0.01 sec)

mysql> insert performs\_in values(501,2);

Query OK, 0 rows affected (0.00 sec)

mysql> insert performs\_in values(501,3);

Query OK, 0 rows affected (0.01 sec)

mysql> insert performs\_in values(502,1);

Query OK, 0 rows affected (0.01 sec)

mysql> insert performs\_in values(502,3);

Query OK, 0 rows affected (0.00 sec)

mysql> insert performs\_in values(502,2);

Query OK, 0 rows affected (0.01 sec)

mysql> insert performs\_in values(503,2);

Query OK, 0 rows affected (0.00 sec)

mysql> insert performs\_in values(503,3);

Query OK, 0 rows affected (0.00 sec)

mysql> insert performs\_in values(503,4);

Query OK, 0 rows affected (0.00 sec)

mysql> insert performs\_in values(504,2);

Query OK, 0 rows affected (0.00 sec)

mysql> insert performs\_in values(504,3);

Query OK, 0 rows affected (0.00 sec)

mysql> insert performs\_in values(504,4);

Query OK, 0 rows affected (0.00 sec)

mysql> insert performs\_in values(505,1);

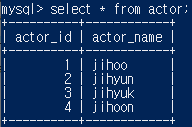
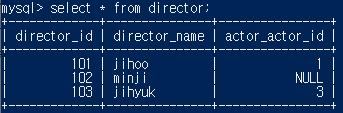
Query OK, 0 rows affected (0.00 sec)

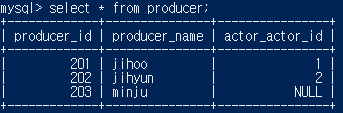
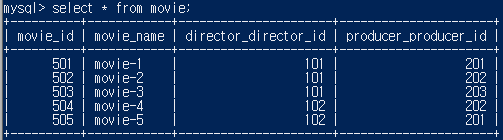
mysql> insert performs\_in values(505,3);

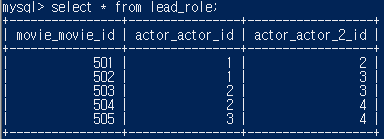
Query OK, 0 rows affected (0.00 sec)

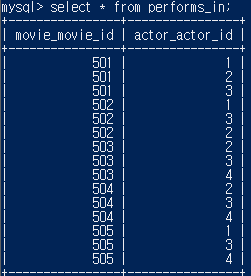
mysql> insert performs\_in values(505,4);

Query OK, 0 rows affected (0.00 sec)

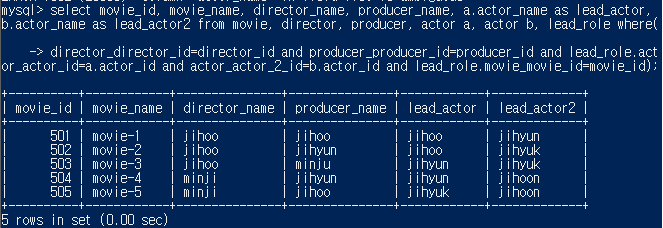
 





위와 같이 6개의 테이블에 샘플 데이터들을 insert하였다.

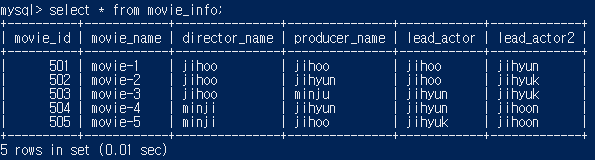
-CREATE VIEW문



우선 view문을 만들기 전에 원하는 결과가 나오도록 select문을 위 사진처럼 작성하였다. 이 select문을 토대로 view문을 만든다.

mysql> create view movie\_info as select movie\_id, movie\_name, director\_name, producer\_name, a.actor\_name as lead\_actor, b.actor\_name as lead\_actor2 from movie, director, producer, actor a, actor b, lead\_role where(director\_director\_id=director\_id and producer\_producer\_id=producer\_id and lead\_role.actor\_actor\_id=a.actor\_id and actor\_actor\_2\_id=b.actor\_id and lead\_role.movie\_movie\_id=movie\_id);

Query OK, 0 rows affected (0.04 sec)

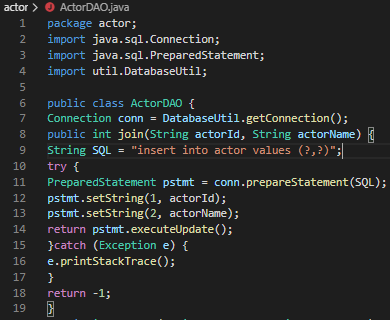
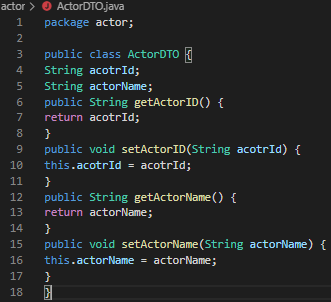


위 사진과 같이 movie\_info 뷰에 정보들이 정상적으로 출력된다.

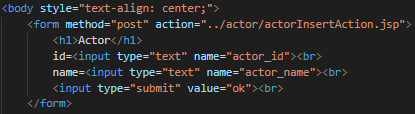
-MOVIES에 대한 4개의 페이지 작성하기

(작성방법이 테이블별로 모두 동일하므로 보고서에는 actor 테이블의 삽입,수정,삭제만 첨부하였다.)

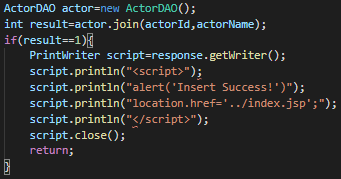
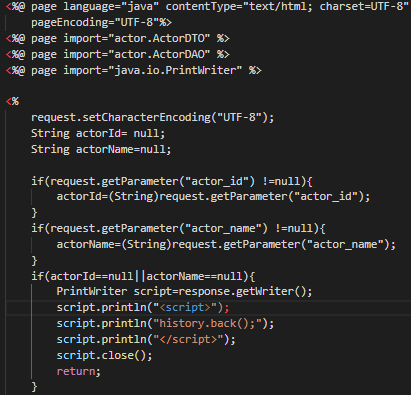
가. 삽입



ActorDAO에서 join 함수를 이용하여 actor 테이블의 값을 삽입해주었다.



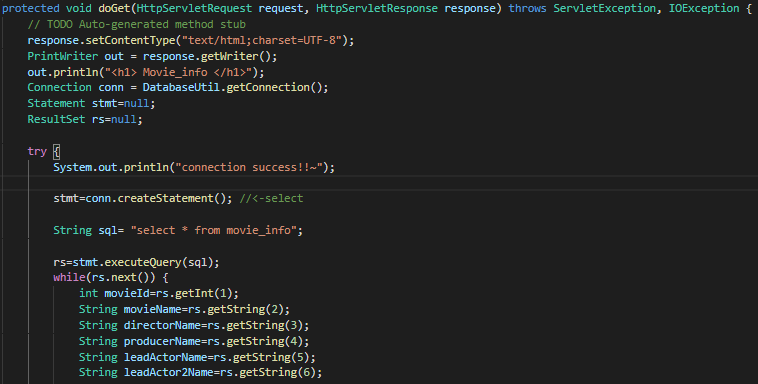
그런 다음 html 파일에서 actor을 삽입하는 부분을 form으로 감싸고 이를 actorInsertAction.jsp로 넘겨준다.

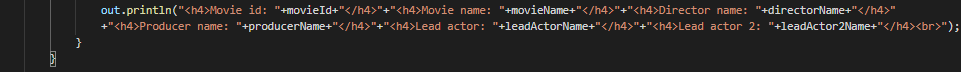


actorInsertAction.jsp에서는 html로부터 받은 값들을 모두 저장해주고, ActorDAO의 join 함수를 이용하여 테이블에 반영한다.

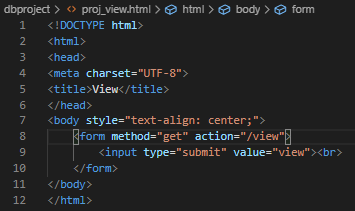
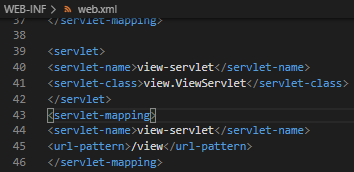
이와 똑같은 방법으로 director, producer, movie 테이블의 삽입도 구현한다.

나. 조회 (view를 통한 조회)



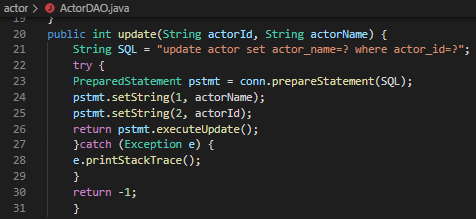


view 폴더 안에 viewServlet.java 파일을 생성하여 movie\_info view의 내용을 출력하는 내용을 작성한다.

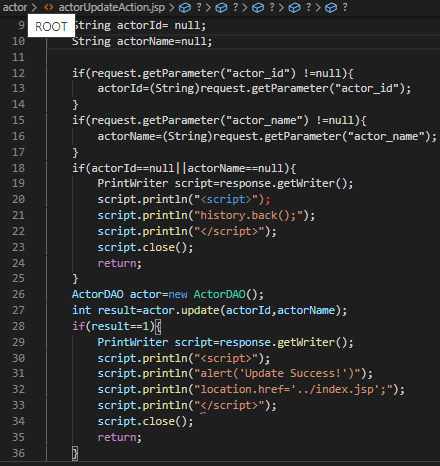


그런 다음 web.xml 파일에서 servlet 이름과 url을 매핑시켜준 다음, html 파일에서 form action으로 /view를 넘겨준다.

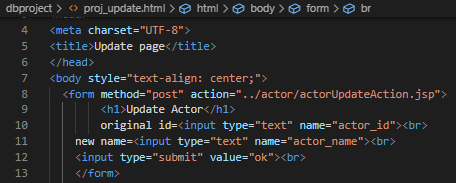
다. 수정



ActorDAO에서 update 함수를 작성하여 actor\_id값을 바탕으로 actor\_name을 수정한다.



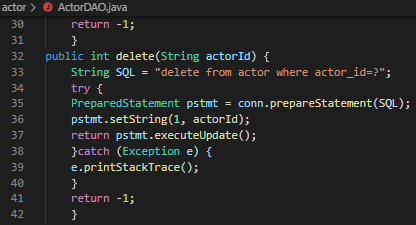
그 다음 actorUpdateAction.jsp를 작성하여, ActorDAO의 update 함수를 이용하여 actor 튜플을 업데이트한다.



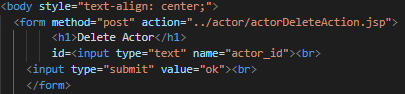
그 다음 html 파일에서 actor을 수정하는 부분을 form으로 감싸고 actorUpdateAction.jsp로 넘겨준다.

이와 똑같은 방법으로 director, producer, movie 테이블의 수정도 구현한다.

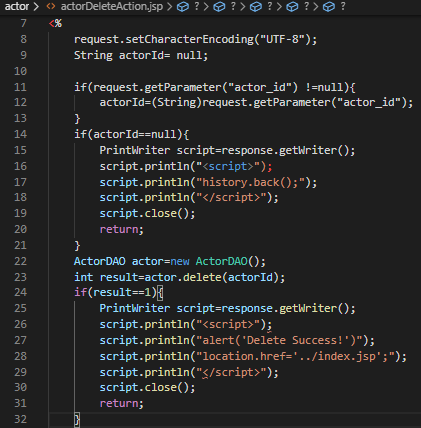
라. 삭제



ActorDAO에서 delete 함수를 작성하여 actor\_id 값을 입력으로 받아 해당 튜플을 삭제하도록 한다.



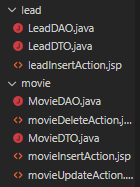
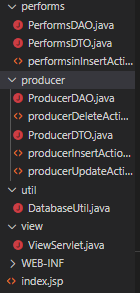
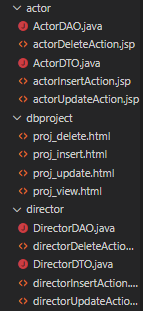
그 다음 html 파일에서 삭제할 튜플의 id를 받은 후 form을 actorDeleteAction.jsp로 넘겨준다.



그 다음 actorDeleteAction.jsp에서 html에서 받은 actor\_id를 받아서 저장하고 ActorDAO의 delete 함수를 이용하여 튜플을 삭제한다.

**-최종 결과 확인**

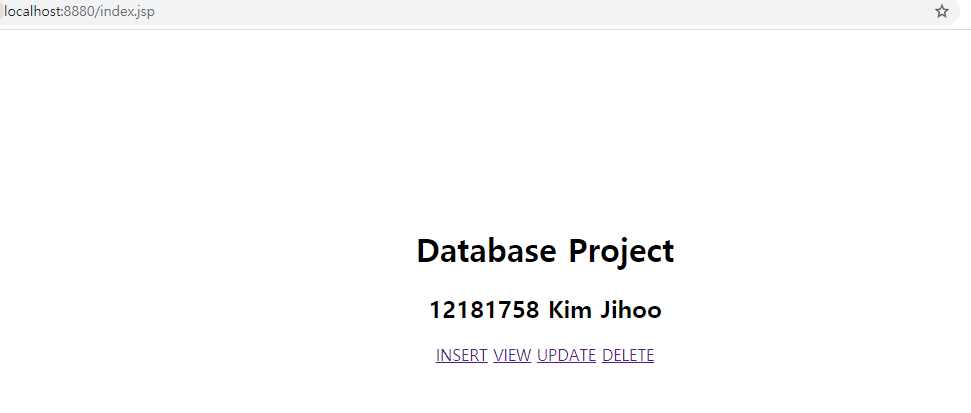
-코드 파일 설명



- ”~~DAO.java”, “~~DTO.java” 해당 클래스의 DAO, DTO를 작성하였다.

- “~~InsertAction.jsp”,“~~UpdateAction.jsp”,“~~DeleteAction.jsp” : 위 html파일에서 각 테이블별로 form으로 연결되는 코드이다.

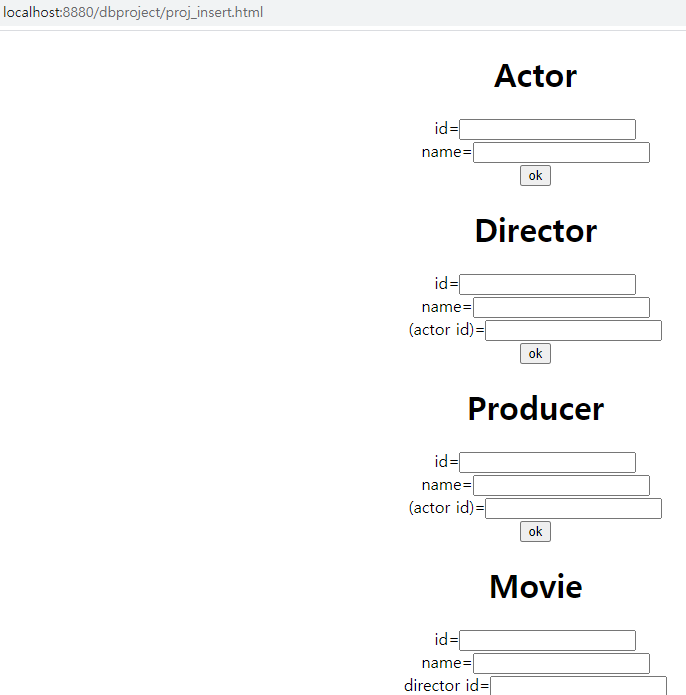
-index.jsp : 맨 처음 출력되는 메인 페이지이다. 여기서 insert, view, update, delete 네 가지 메뉴로 갈 수 있다.



<- 처음 로드되는 메인 페이지

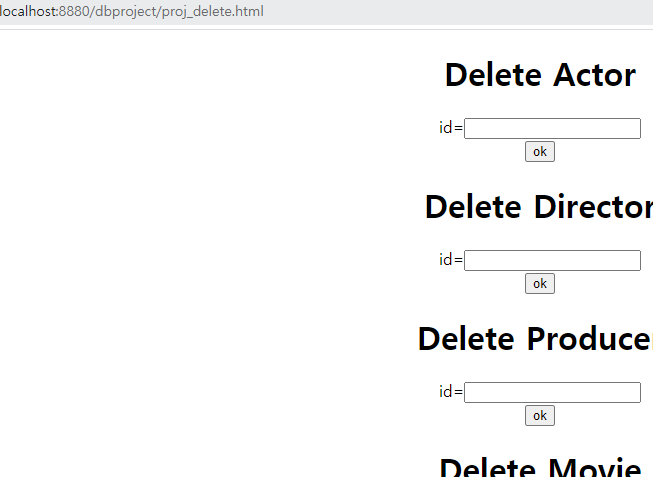
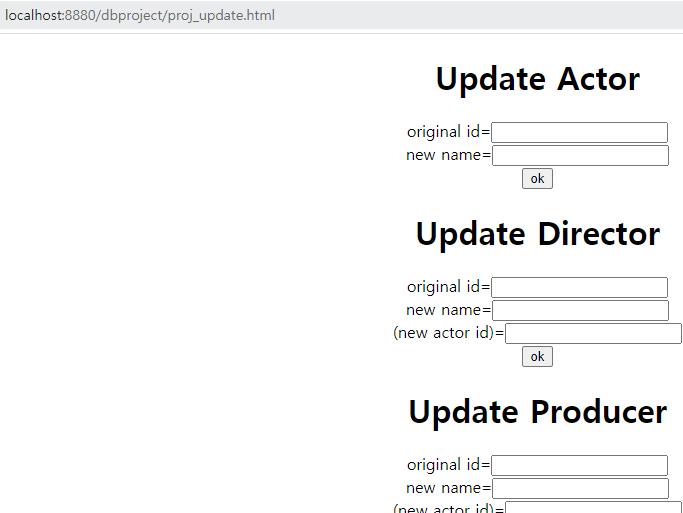
처음 index.jsp 페이지에 들어가면 위와 같이 이름과 함께 insert, view, update, delete 네 가지 메뉴가 뜬다. 각 메뉴마다 다른 html 파일로 연결되어있다.

- “proj\_delete.html”,“proj\_insert.html”,“proj\_updatehtml”,“proj\_view.html” : 각각 delete, insert, update, view 네 가지 메뉴 클릭에 연결되는 html 파일이다.



insert 페이지에서는 한 html 파일 안에서

각 테이블별로 insert 할 수 있다.



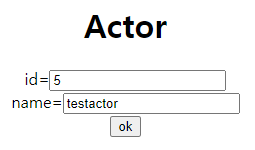
(왼: proj\_update.html, 오 : proj\_delete.html)

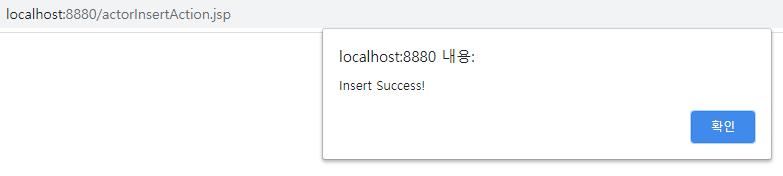


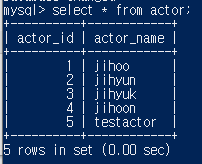
(proj\_view.html)

가. 삽입

-actor 삽입

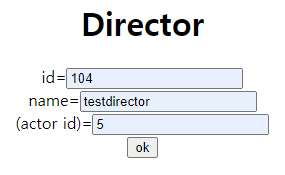


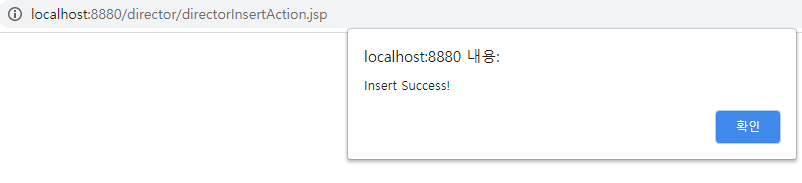




insert 페이지에서 actor 부분에 위와 같이 값을 넣어주고 ok 버튼을 누르면 actorInsertAction.jsp로 이동하며 insert success 메시지가 뜬다. 이후 mysql에서 확인해보면 위와 같이 튜플이 들어간 것을 알 수 있다.

-director 삽입

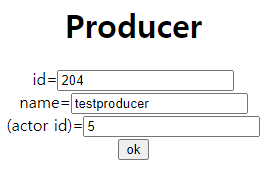
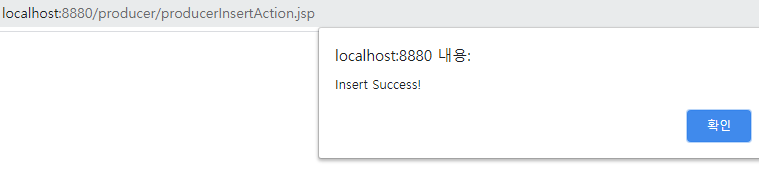


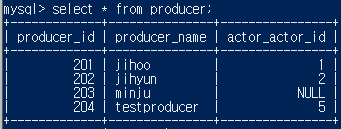




director도 마찬가지로 insert 페이지의 director 부분에 해당 값을 넣고 ok버튼을 누르면 성공 메시지가 뜬다. 실제 데이터베이스에도 새 튜플이 들어간 것을 볼 수 있다.

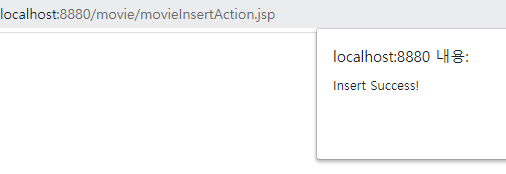
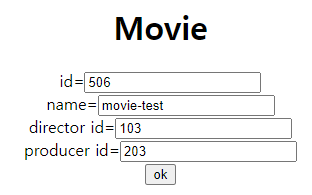
-producer 삽입

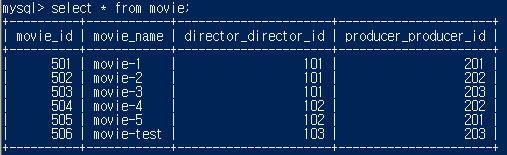
 



producer 테이블도 마찬가지로 insert 페이지에서 값을 입력하고 제출하면 성공 메시지가 뜨고 테이블에도 들어간 것을 볼 수 있다.(testproducer)

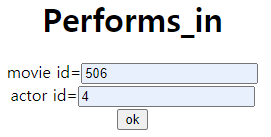
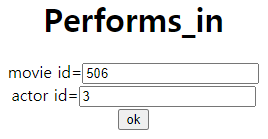
-movie 삽입

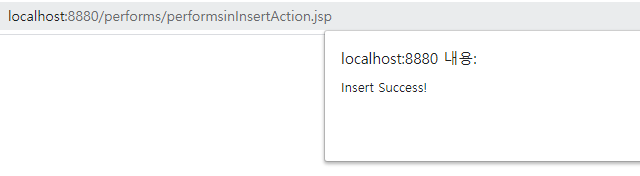




insert 결과 movie-test라는 새 튜플이 추가되었다.

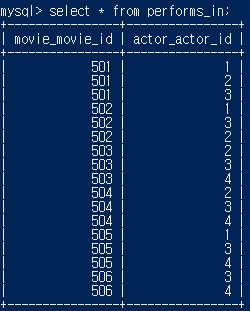
-performs\_in 삽입





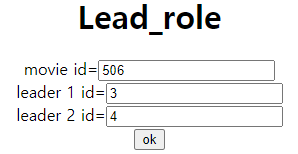
insert 페이지의 performs\_in 부분에 값을 넣고 두 개의 튜플을 insert하였다.

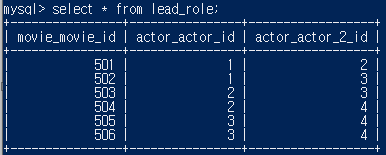
성공 메시지가 나타난 부분을 보면 performsinInsertAction.jsp로 form이 넘어간 것을 볼 수 있다.



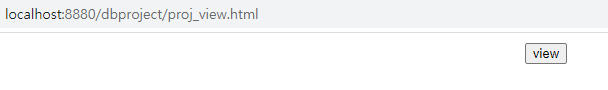
그 결과 새로 웹에서 추가한 튜플이 데이터베이스에 반영되었다.

-lead\_role 삽입



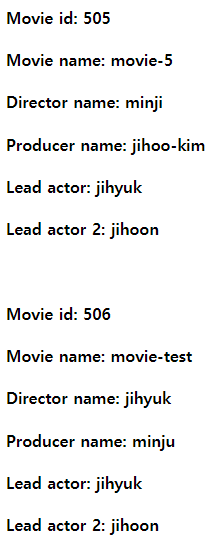
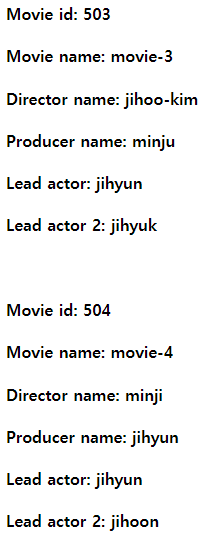
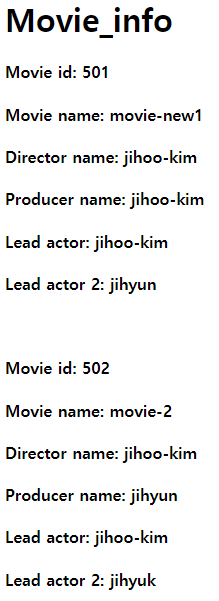


나. 조회(view를 통한 조회)



우선 view 메뉴에 들어가게 되면 위와 같이 view 버튼 하나가 뜨게 된다.

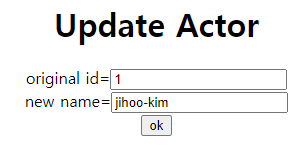


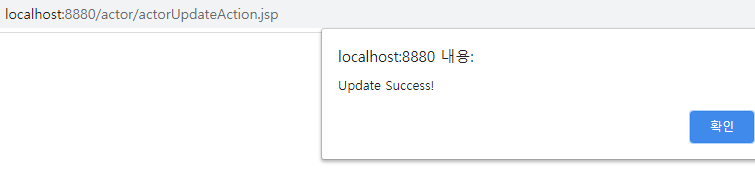


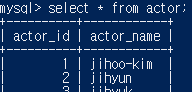
view 버튼을 누른 결과는 위와 같다. 임의로 만든 view인 movie\_info의 모든 튜플들이 차례로 출력된다. (view 조회를 가장 늦게 구현하였기 때문에 1번 actor인 'jihoo'가 'jihoo-kim'으로 이름이 바뀌었다. 이는 바로 뒤에 나오는 수정 부분의 실행결과이다. 또한 insert의 결과인 506번 movie 부분 또한 정상적으로 나오는 것을 볼 수 있다.)

다. 수정

-actor 수정

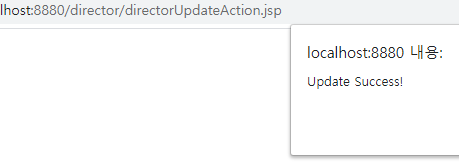
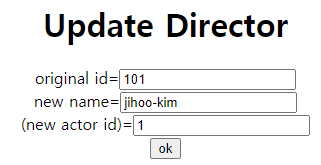


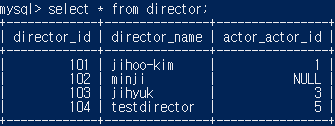




update 메뉴의 actor 부분에서 수정할 튜플의 id와 새롭게 바꿀 이름을 입력하고 제출하면 성공 메시지가 뜨게 된다. 결과를 확인한 결과 1번 actor의 이름이 jihoo에서 jihoo-kim으로 바뀌었다.

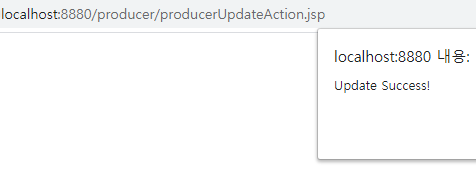
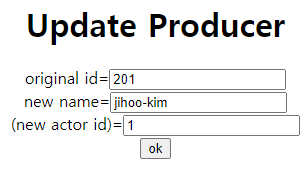
-director 수정

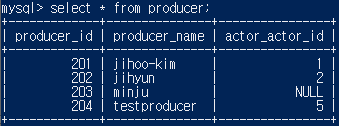




마찬가지로 101번 director의 이름이 jihoo에서 jihoo-kim으로 바뀌었다.

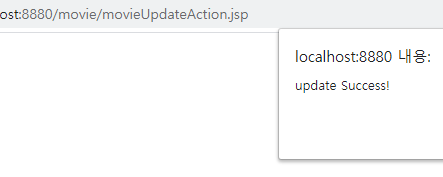
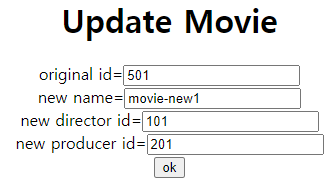
-producer 수정

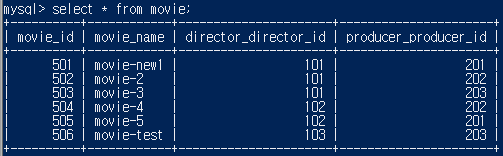




마찬가지로 201번 producer의 이름이 jihoo에서 jihoo-kim으로 바뀌었다.

-movie 수정

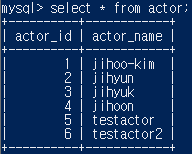




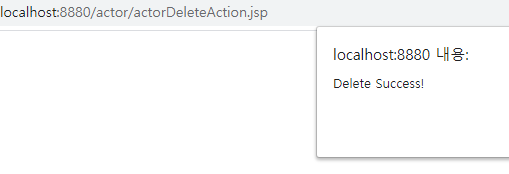
501번 movie의 이름을 movie-1에서 movie-new1로 변경하였다.

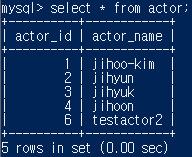
라. 삭제

-actor 삭제



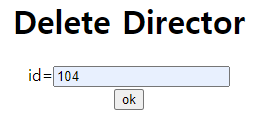
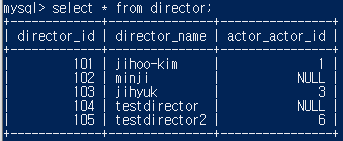
(6번 actor를 새로 삽입한 상태에서 시작, 5번 actor를 delete)



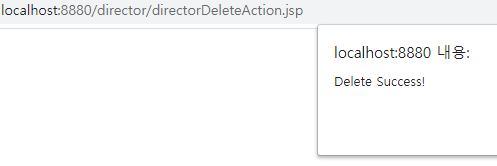


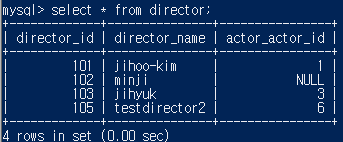
성공 메시지와 함께 5번 actor가 삭제되었음을 확인할 수 있다.

-director 삭제



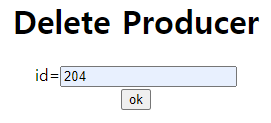
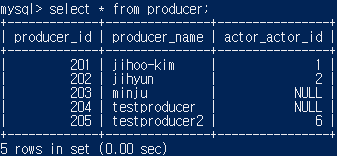
(여기서 104번 director의 actor\_id가 5에서 null로 바뀐 이유는 바로 이전에 5번 actor를 삭제하였기 떄문이다.)

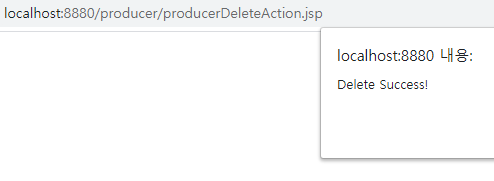


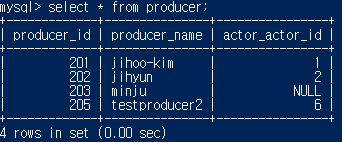


director 104번이 삭제되었다.

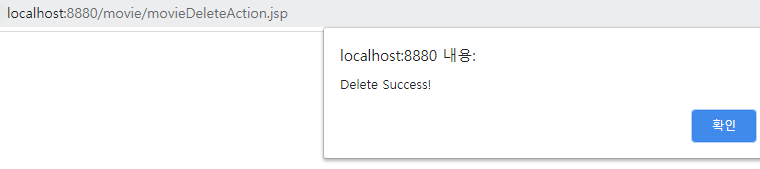
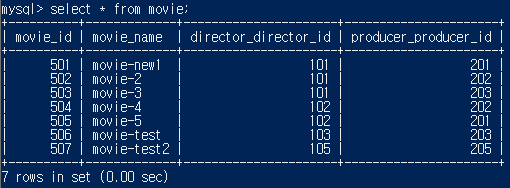
-producer 삭제

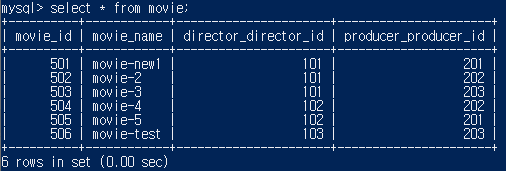






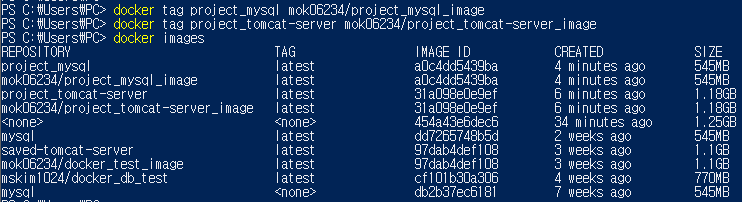
-movie 삭제

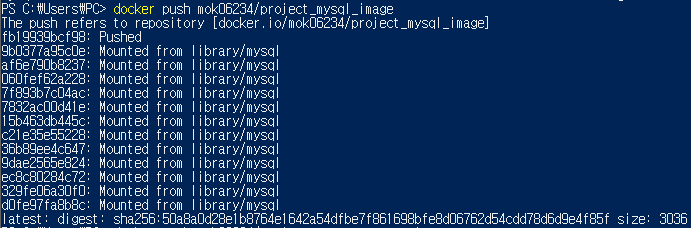


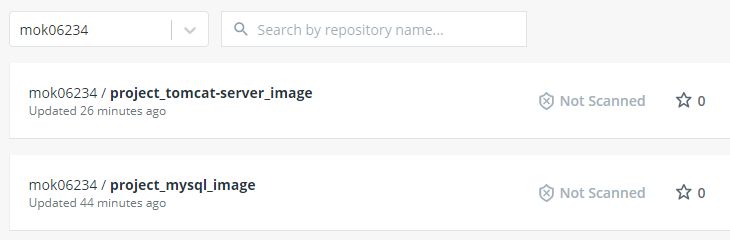


507번 영화가 삭제되었다.

-Docker 이미지







**tomcat server 이미지 : mok06234/project\_tomcat-server\_image**

**mysql 이미지 : mok06234/project\_mysql\_image**