1. **개요**

데이터베이스 실습과 이론을 통해 얻은 설계 방법으로, MOVIES 데이터베이스를 설계하고, MySQL에 데이터베이스를 구현, web에서 작동하는 것 까지를 설계하는 것이 목표였다. Visual Studio Code를 이용하여 tomcat server에 movie의 dao, dto를 만들고, 이를 이용하여 jsp또는 html파일에서 MySQL의 movie DB에 각 엔티티를 삽입, 수정, 조회, 삭제할 수 있는 기능을 구현하였다.

8880 서버가 작동하지 않아 tomcat-server2(port:8860)를 새로 만들어 진행하였으며, MySQL과 movie에 대한 이미지를 도커 허브에 올리는 것으로 마무리했다.

추가적으로 제출하라셨던 insert 문의 경우에, trigger를 사용하지 않아서 미리 넣어줄 필요가 없었습니다. 직접 insert 하여 만들지 않고, 모두 jsp를 이용하여 데이터베이스를 만들었습니다. 따라서 데이터베이스 생성문에 insert문은 첨부하지 않았습니다. mysql 연결 ip는 172.17.0.2 입니다.

1. **상세 설계내용 및 결과**

**ㅇMOVIES ERD를 기반으로 MOVIES데이타베이스를 설계하고,**

**ㅇ이를 MySQL에 데이타베이스를 구현한 후, web application을 작성하는 것을 목표로 함(총 300점)**

**1. 다음 페이지의 MOVIES 데이타베이스의 ERD에 대한 답을 제시할 것 (65점)**

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

: 어떤 영화도 출연하지 않은 배우는 없다.

PERFORMS\_IN에 ACTOR가 전체참여 이므로

-> True

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

: 10개가 넘는 영화에 출연한 배우가 있다.

ACTOR와 MOVIE의 PERFORMS\_IN RELATION이 M:N이므로 있을 수도 있고 아닐 수도 있다.

-> Maybe, \* 영화의 수는 10개를 넘을 수 있지만 지금은 없다고 가정한다. -> False

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

: 몇몇 배우는 여러 영화에서 lead role을 맡았다.

ACTOR와 MOVIE의 LEAD\_ROLE RELATION이 2:N이고, MOVIE가 전체 참여이므로

-> True

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

: 영화는 최대 2명의 lead actor를 가질 수 있다.

Actor에서 참여 수가 최대 2라고 적혀있으므로

-> True

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

: 모든 감독은 몇몇 영화에서 배우였다(현재완료형).

ACTOR와 DIRECTOR가 ALSO\_A\_DIRECTOR REALTION에서 부분참여이다.

has been이므로 과거에 어땠는지는 ERD상에서 알 수 없다.

-> Maybe, \* 배우 경력이 있을 수 있다고 가정한다.(단 DB에서는 나타내지 않는다) -> True

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

: 어떤 프로듀서도 배우였던 적이 없다(현재완료형).

ACTOR\_PRODUCER에 부분참여이므로 배우일 수 있다.

하지만 과거에 어땠는지는 알 수 없다.

-> Maybe, \* 있다고 가정한다(단 DB에서는 나타내지 않는다) -> False

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

: 프로듀서는 다른 영화의 배우가 될 수 없다.

PRODUCER가 ACTOR일 수 있고, ACTOR는 여러개의 MOVIE에 출연가능하다.

-> False

**(8) There are movies with more than a dozen actors.**

: 12명이 넘는 배우가 있는 영화가 있다.

ACTOR와 MOVIE가 PERFROMS\_IN에서 M:N 관계이다.

이때 이 DB에 12명이 넘는 배우가 있을 순 있지만, 없을 가능성도 있다.

->Maybe, \* DB상에 없다고 가정한다 -> False

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

: 프로듀서는 감독이었을 수 있다(현재완료형).

PRODUCER와 DIRECTOR 사이의 관계가 없다.

-> False

**(10) Most movies have one director and one producer.**

: 대부분의 영화는 한명의 디렉터와 한명의 프로듀서를 갖는다.

MOVIE와 DIRECTOR가 N:1이고, PRODUCER와 N:M 관계다. 즉 한명의 감독은 맞지만, 여러명의 프로듀서일 수 있다.

대다수인지 아닌지는 나와있지 않다. 따라서

-> Maybe, \* 한 명의 프로듀서만 갖는다고 가정한다, ERD 상에서 PRODUCES 관계를 1:N으로 변경한다.

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

: 몇몇 영화는 한명의 감독과 여러명의 프로듀서를 갖는다

MOVIE와 DIRECTOR는 N:1관계, PRODUCER와는 N:M 관계이고 전체 참여이므로

-> True, 하지만 (10)의 가정에 의해서 1:N으로 변경했으므로 오직 한 명의 감독과 프로듀서만 가질 수 있다. -> False

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

: lead role, director, producer였던 배우가 있다.

ACTOR와 DIRECTOR, PRODUCER 사이에 관계가 존재하므로 (ALSO\_A\_DIRECTOR, ACTOR\_PRODUCER)

-> True

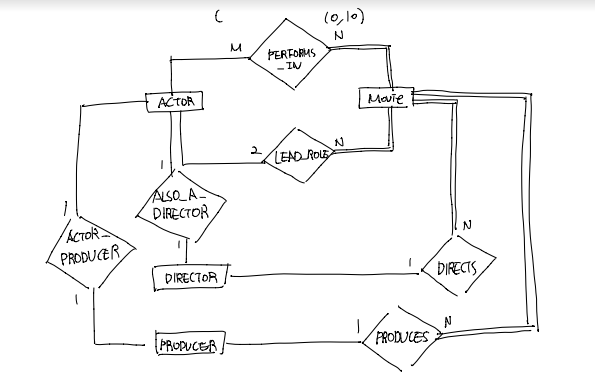
**(13) No movie has a director who also acted in that movie.**

: 한 영화에 배우이면서 감독인 영화는 없다.

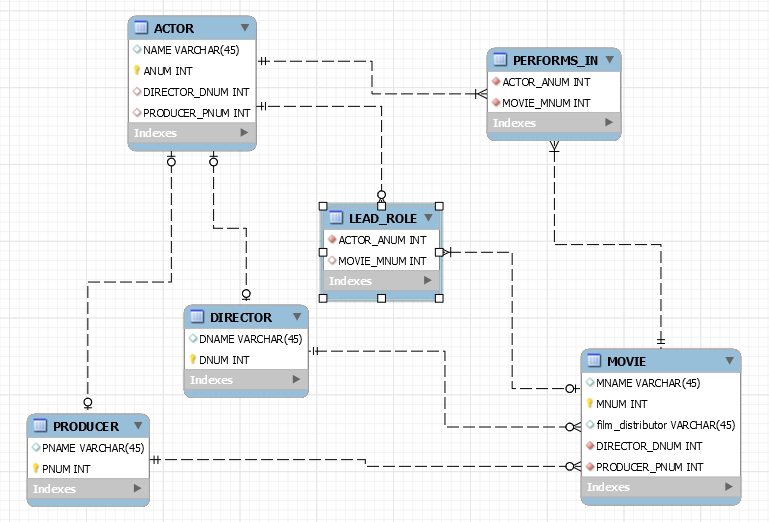
ALSO\_A\_DIRECTOR 관계에 의해서 있을 수도 있다.

-> Maybe, \* 있다고 가정한다 -> True

**2. Maybe에 대한 본인의 가정을 바탕으로 수정한 ERD를 제시할 것 (35점)**



**3. MOVIES데이타베이스를 MySQL에 생성할 것(생성한 스크립트를 제출할 것) (40점)**



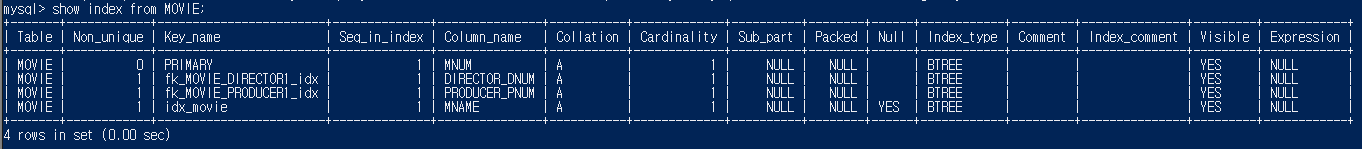
**4. 어떤 인덱스를 사용했는지 제시할 것 (20점)**

검색 속도 향상을 위해 INDEX를 만들었다. FK, 기본키를 통한 INDEX를 사용하였고, MOVIE에서는

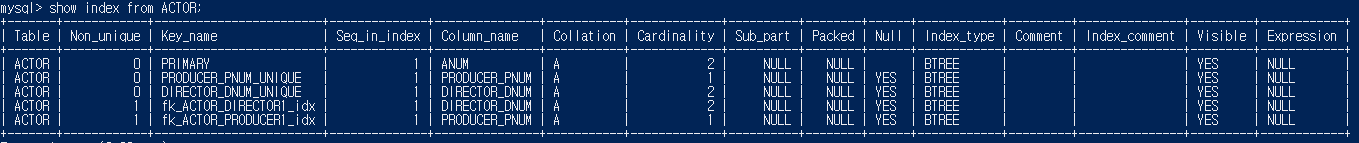
create INDEX idx\_movie ON MOVIE (MNAME);

구문을 활용해 영화 이름 INDEX도 만들어 주었다.

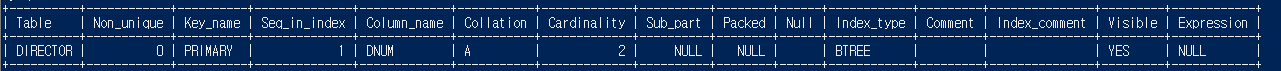
MOVIE



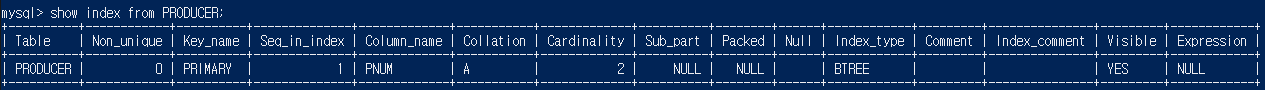
ACTOR



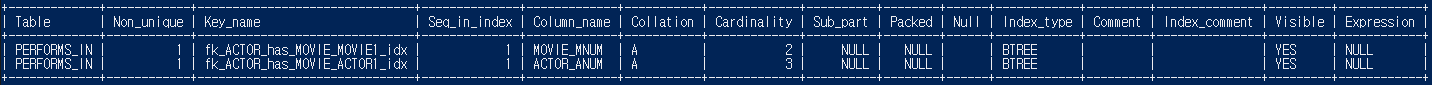
DIRECTOR



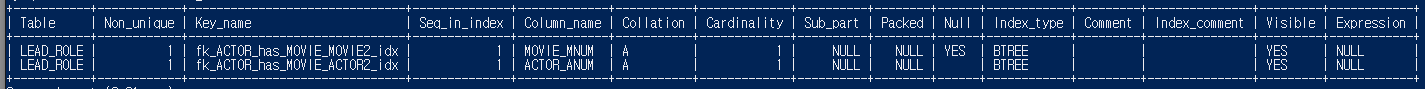
PRODUCER



PERFORMS\_IN



LEAD\_ROLE



**5. 두 개 이상의 테이블을 조인한 view를 하나이상 생성할 것(create view문을 제출할 것) (20점)**

어떤 배우가 어떤 영화에서 주연을 맡았는지에 대한 VIEW문을 만들었다. 기존에는 영화 번호와 배우의 번호로 연결되어 있기 때문에, 이름을 나타내어 더 직관적으로 보이는 용도의 VIEW이다.

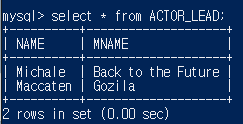
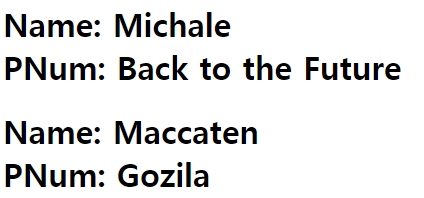
create view ACTOR\_LEAD as

select NAME, MNAME

from ACTOR A, LEAD\_ROLE P, MOVIE M

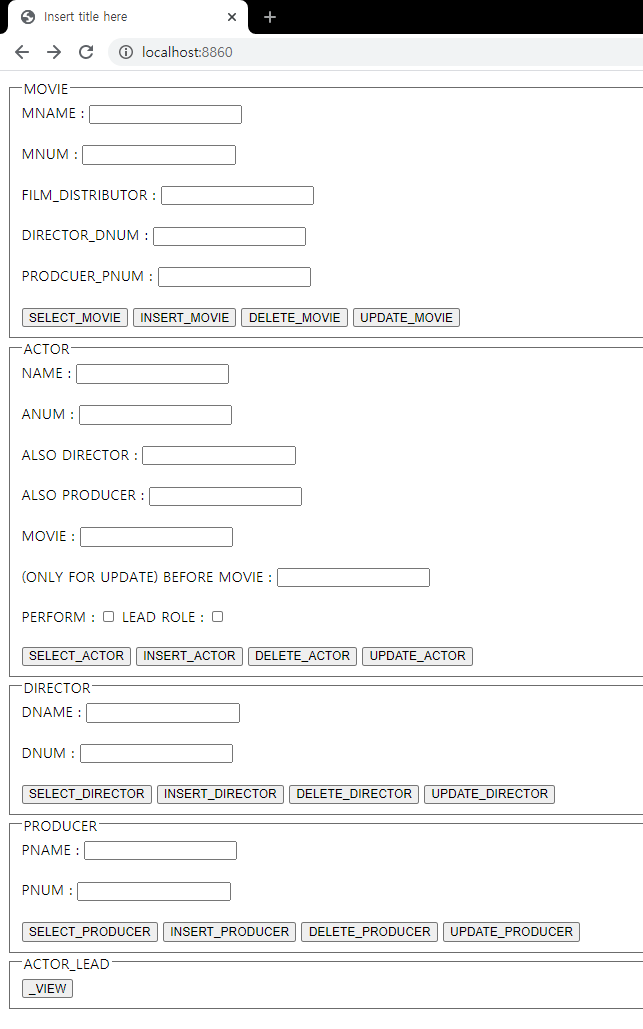
where A.ANUM = P.ACTOR\_ANUM AND P.MOVIE\_MNUM = M.MNUM;

아래 왼쪽은 jsp를 이용하여 view 문을 조회한 것이다.



**6. MOVIES데이타베이스에 대한 삽입, 조회, 수정, 삭제 페이지를 작성할 것 (각 30점씩 총120점) • 최소한의 삽입, 조회(view를 통한 조회), 수정, 삭제 기능으로 재량껏 설계할 것.**

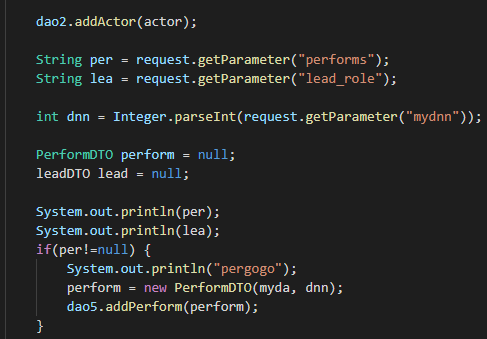
하나의 html 페이지에서 actor, movie, director, producer를 각각 삽입, 조회, 수정, 삭제할 수 있도록 하였다.



**ACTOR**

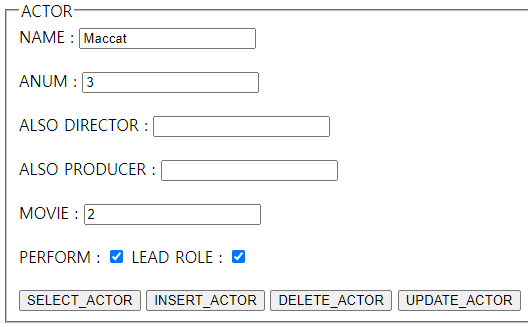
(1) 삽입

actor를 삽입할 때 performs\_in과 lead\_role에 해당하는지 체크하도록 하였다. 그러면 입력된 영화의 번호와 함께 actor의 번호를 performs\_in, lead\_role에 넘겨주어 또 다른 엔티티를 생성하도록 하였다.

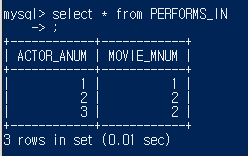


예를 들어, Michale 이라는 배우를 입력하면, view문을 통해서 아래와 같이 어떤 영화에 출연하였는지도 관계 지어지게 된다.



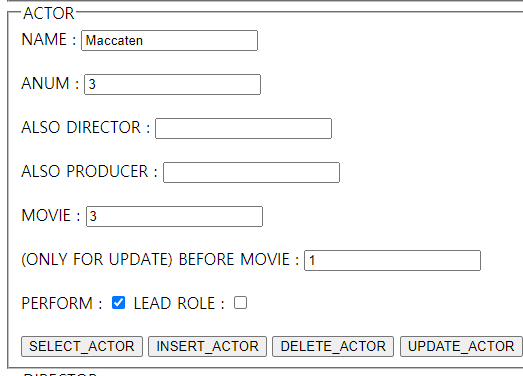


아래는 HTML의 insert를 이용한 결과이다.

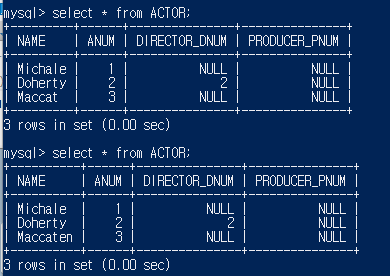


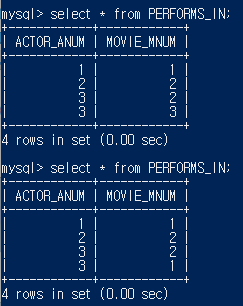
(2) 수정

수정의 경우에는 삽입과 마찬가지로 관계를 갱신하도록 했다.

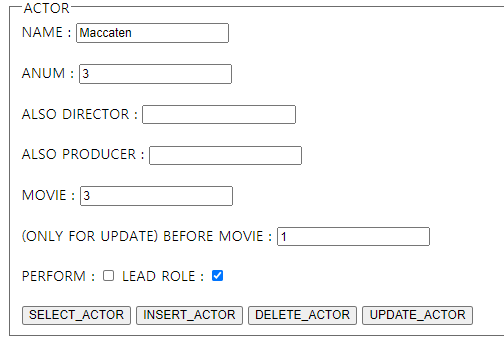
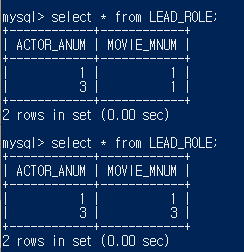


기존에 1번 영화에 출연했다고 한 부분을 3번으로, 이름을 Maccat에서 Maccaten으로 수정할 것이다.



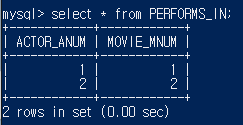
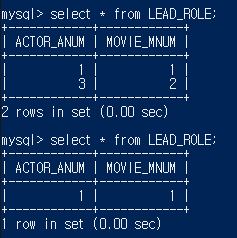


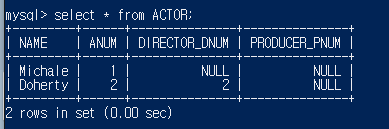
위의 결과로 update를 이용했을 때 어떤 영화에 출연하였는지 수정할 수 있음을 알 수 있다.



마찬가지로 LEAD\_ROLE도 체크 후 Update로 수정할 수 있다.

(3) 삭제

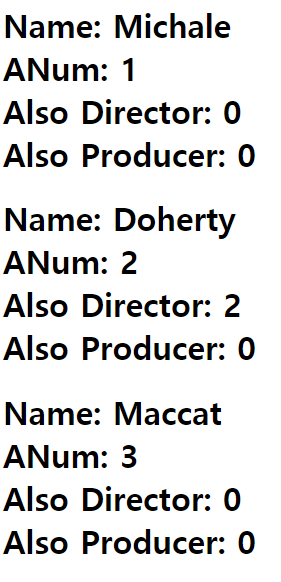




이전에 insert한 3번 배우가 PERFORMS\_IN, LEAD\_ROLE, ACTOR 모두에서 사라진 것을 확인할 수 있다.

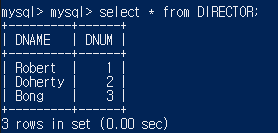
(4) 조회

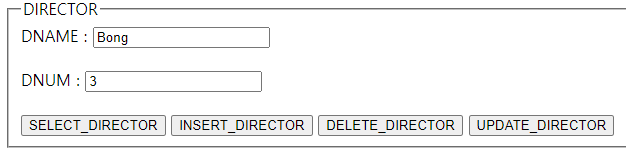
조회는 전체조회로 만들었다. 개인을 조회하는 것에 loop를 돌려 만든 것이다. Also 부분에서 0은 NULL을 뜻한다(FK가 지정되지 않은 것). 2가 지정된 것을 감독 번호 2번과 동일인물이라는 뜻이다.



**DIRECTOR**

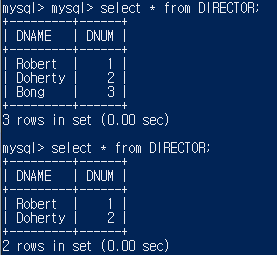
(1) 삽입





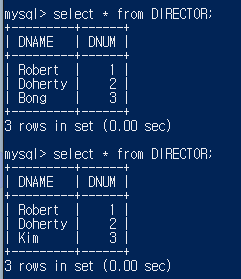
감독 삽입 전 후의 모습이다.

(2) 삭제



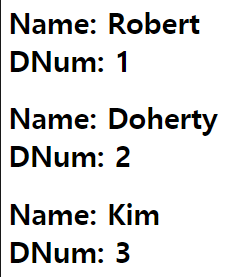
삭제한 모습이다.

(3) 수정



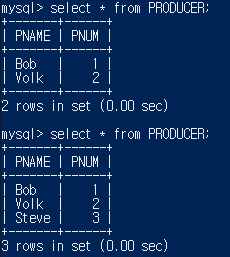
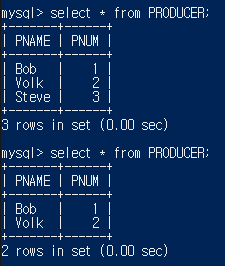
수정한 모습이다.

(4) 조회



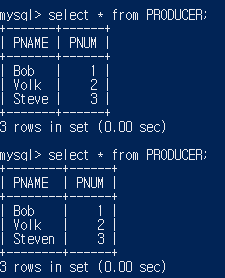
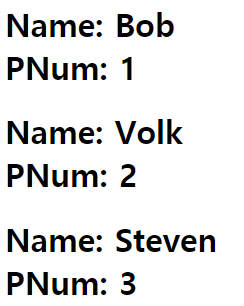
**PRODUCER**

(1) 삽입/삭제



왼쪽은 삽입, 오른쪽은 삭제한 모습이다.

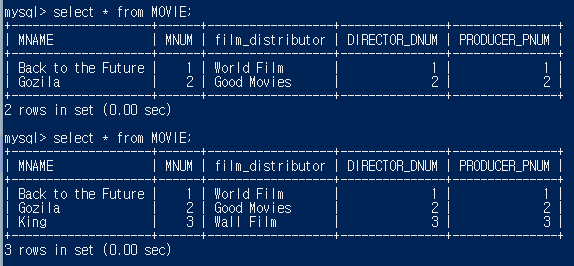
(2) 수정/조회



수정한 모습과 조회한 모습이다.

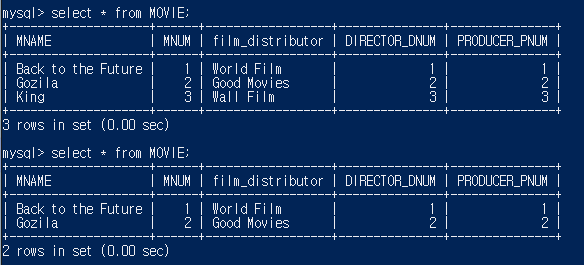
**MOVIE**

(1) 삽입



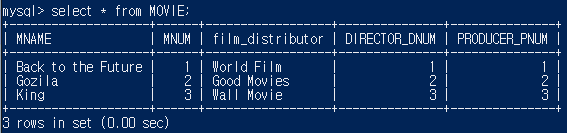
삽입한 모습이다.

(2) 삭제



삭제한 모습이다.

(3) 수정



수정한 모습이다.

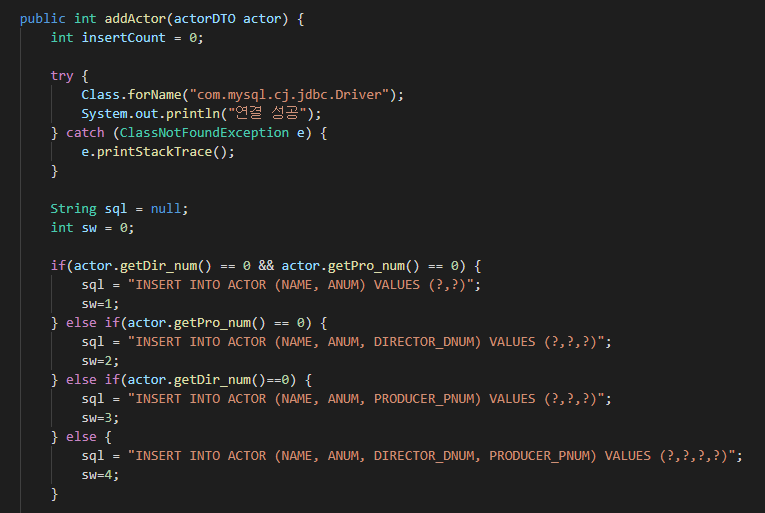
(4) 조회



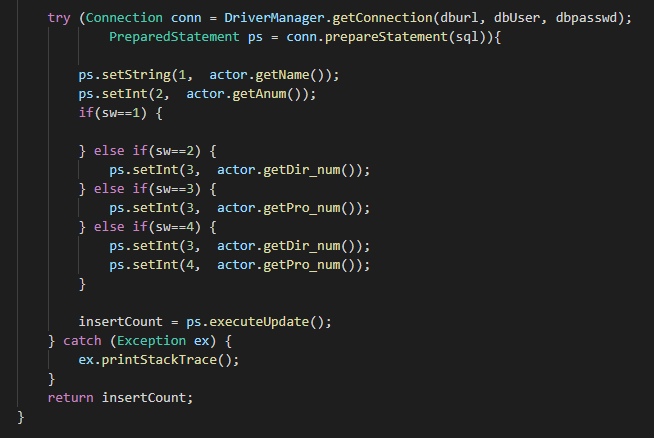
**7. 코드 분석**

DAO의 삽입, 수정, 삭제, 조회는 각 엔티티에서 동일한 원리로 작동한다. 따라서 ACTOR만 분석했다.

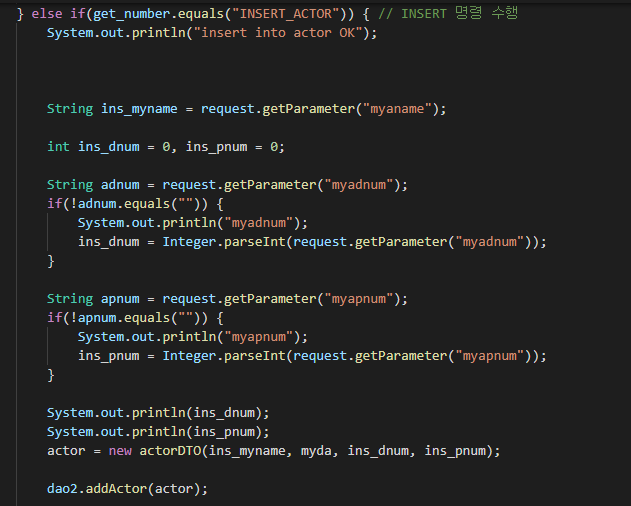
(1) 삽입



연결에 성공하면, Director num과 Producer num에 따라서 쿼리문을 다르게 적용한다.

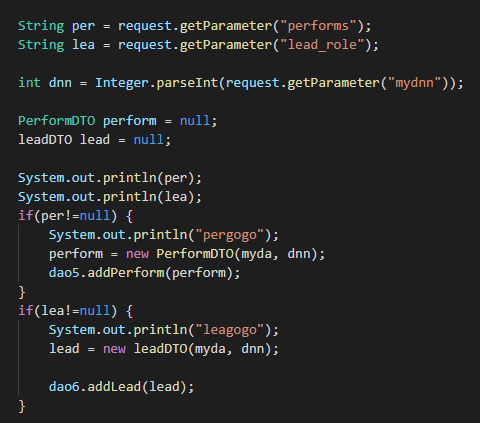


movie.java에서 실제로 사용할 때 넘겨준 인자 값들로, 쿼리 질의문을 채워서 실행하여 삽입한다.



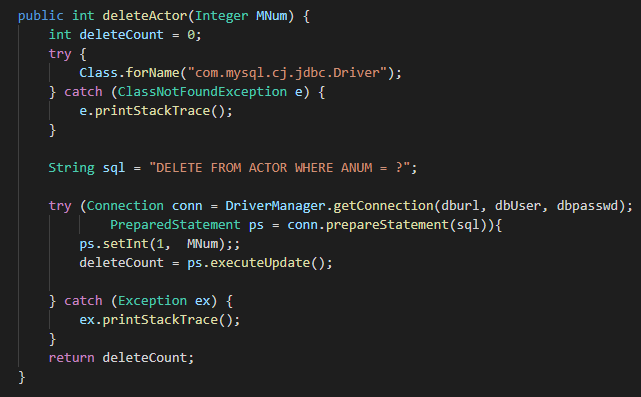
HTML(jsp)에서 UPDATE\_ACTOR를 누르면, 뒤의 5글자를 인식하여 ACTOR 엔티티에 대한 연산임을 확인한다. 또한 버튼의 VALUE값을 이용하여 삽입/삭제/조회/수정을 구분한다.

실질적으로 movie.java에서 위의 DAO를 이용하여 actor를 삽입한다. 이 때, actor는 lead\_role과 performs\_in 관계와도 관련있으므로, html에서 받은 movie 번호를 이용하여 이 둘에도 삽입한다.

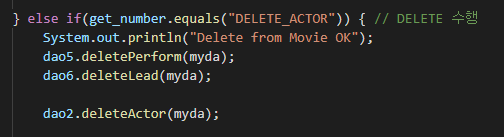


이 구문으로 체크박스에 표시되어 있으면 그 엔티티에도 삽입을 하는 모습이다. 이 때, FK로 연결되어야 하므로 반드시 ACTOR를 먼저 생성해주어야 한다. (배우가 없는 영화나 영화가 없는 배우 관계는 필요하지 않으므로 FK가 NULL이 될 수 없다.)

(2) 삭제

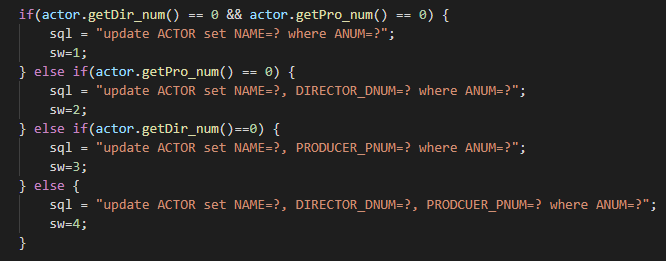


삭제는 간단한 쿼리문에, PK를 받아서 삭제한다.



마찬가지로 performs\_in, lead\_role에서도 삭제한다.

(3) 수정

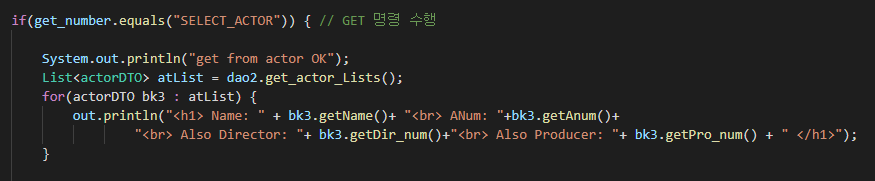


삽입과 같은 원리이며, 전체적으로 비슷하지만 쿼리문을 update 문으로 바꾸어 구현했다. 마찬가지로 lead\_role, performs\_in에도 update한다.

(4) 조회

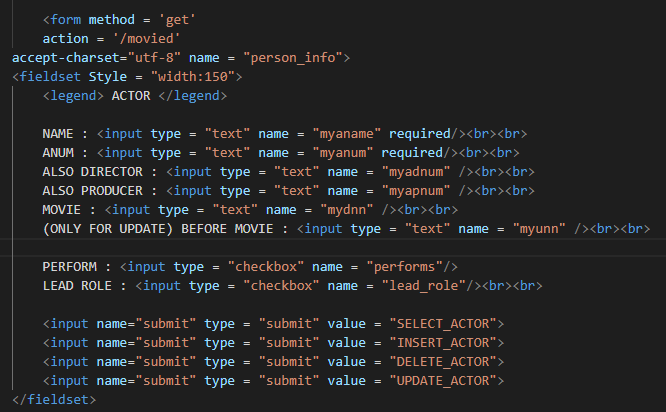


select를 이용하여 튜플을 하나씩 받고, 이것을 list에 쌓아둔다.



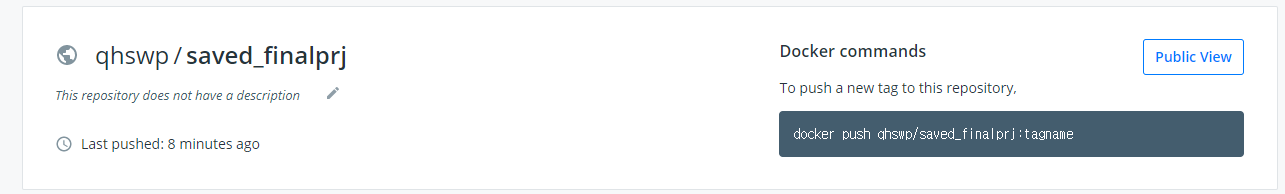
movie.java에서 실행한다.

(5) jsp



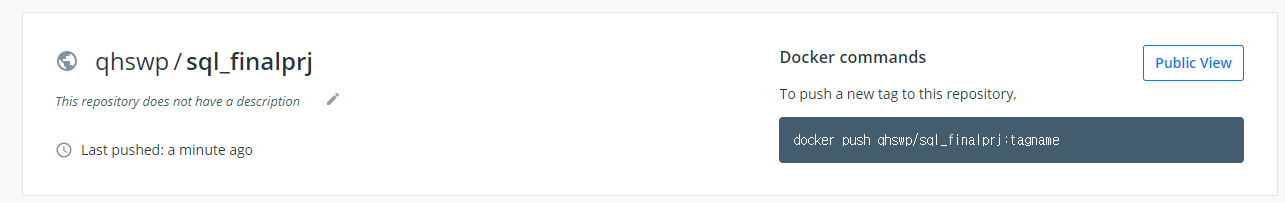
jsp에서는 web.xml에 정의된 pattern /movied로 동작한다. 이것은 movie.java를 컴파일한 servlet class를 지칭한다.

**8. docker hub**



qhswp/saved\_finalprj

docker push qhswp/saved\_finalprj:tagname



qhswp/sql\_finalprj

docker push qhswp/sql\_finalprj:tagname

포트는 8860을 사용하였습니다. mysql 연결 ip는 172.17.0.2 입니다.