

데이터베이스 설계

Final\_Project

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 12 월 13 일

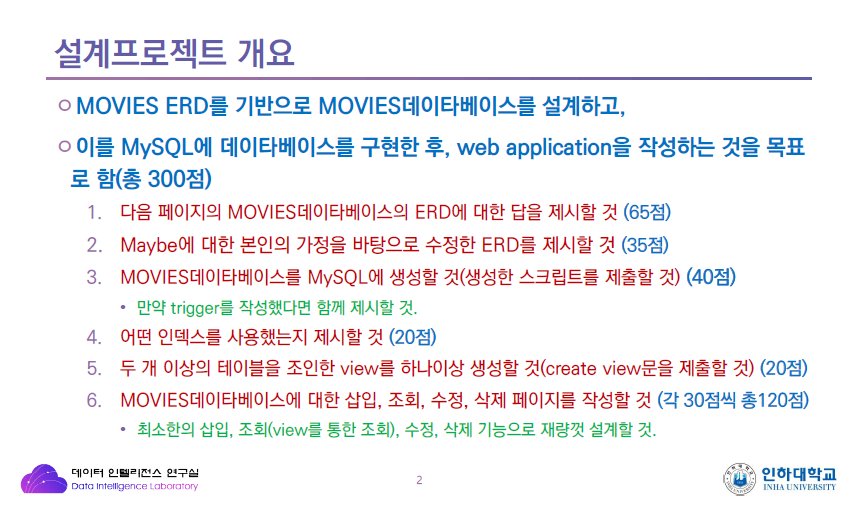
학부 정보통신공학과

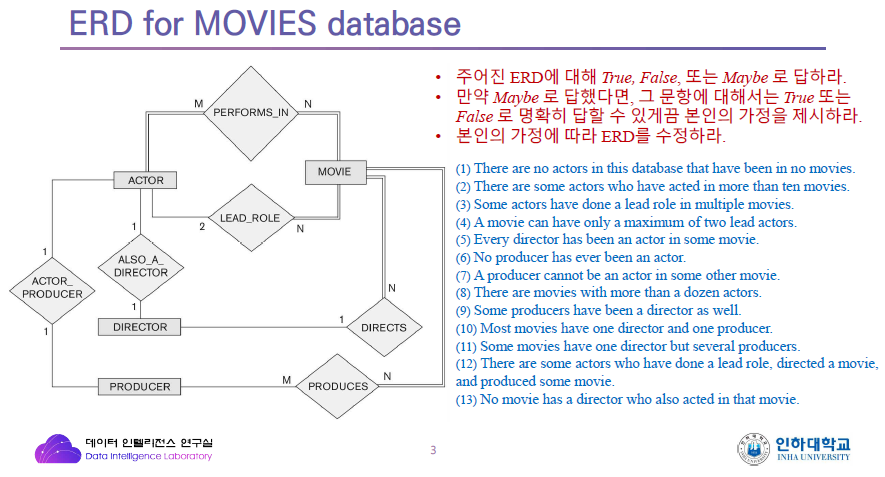
학년 4

성명 정규원

학번 12141760

**1. 개요**





(+ 12.13) 사용했던 모든 코드에 대해 같이 압축해서 업로드 하였습니다.

**2. 상세 설계내용 및 실행 화면**

**2-1. 다음 페이지에 MOVIES 데이터베이스의 ERD에 대한 답을 제시할 것. (65점)**

1. There are no actors in this database that have been in no movies.   
   True다. ACTOR 테이블에 있는 모든 Tuple은 전체 참여로 PERFORMS\_IN에 들어갔다.
2. There are some actors who have acted in more than ten movies.   
   Maybe다. ACTOR와 MOVIE 테이블이 서로 M:N으로 연결되어 있기 때문에 영화를 10편이상 찍은 배우가 있을수도(=True일 수도) 있으나, 필수적인 요소가 아니기 때문에 없을수도 (=False일 수도) 있다.

* 이를 T/F로 명확하게 구분 짓기 위해 ACTOR 테이블에 수상여부 (Win\_Award) attribute를 추가해줬다. 수상조건은 10편이상의 영화에 주연으로 출연하는 것이다.  
  그리하여 ‘배우 목록 중 공로상을 수상한 배우가 1명이 있을 때’라는 가정을 통해 2번 문항을 True로 만들 수 있다.

1. Some actors have done a lead role in multiple movies   
   Maybe 다. 물론 ACTOR와 MOVIE 테이블은 M:N관계로써 몇몇의 배우는 여러 영화에서 주연을 맡았을 수도 있지만, 영화 하나에서만 주연을 맡은 배우도 충분히 있을 수 있다.

* 이를 T/F로 명확하게 구분 짓기 위해 2번 문제에서 사용했던 가정을 가져온다. 그리하여 ‘배우 목록 중 공로상을 수상한 배우가 1명이 있을 때’라는 가정을 통해 3번 문항을 True로 만들 수 있다.

1. A movie can have only a maximum of two lead actors.   
   True다. ACTOR 테이블과 LEAD\_ROLE 관계 간의 Cardinality는 2이기 때문에 최대 2명의 주연배우가 있다.
2. Every director has been an actor in some movie.   
   Maybe다. ACTOR와 DIRECTOR 테이블이 연결이 되어있으므로 모든 디렉터가 배우 경험이 있을 수 있다. 하지만 전체 참여로 연결되어 있지 않으므로 배우 경험이 있는 감독이 한 명도 없을 수 있다.

* 이를 T/F로 명확하게 구분 짓기 위해 가정을 ‘배우 겸 감독이 없을 때’로 설정하였고, ERD에서는 ACTOR와 DIRECTOR간의 Connection을 끊었다. 그러면 5번문항에 대해 False로 대답할 수 있다.

1. No producer has ever been an actor.   
   Maybe다. ACTOR와 PRODUCER 간의 연결이 있기 때문에 FALSE일 수도 있으나, 전체참여가 아니기 때문에 한 명도 없을 수도(=True일 수도) 있다.

* 이를 T/F로 명확하게 구분 짓기 위해 가정을 ‘배우 겸 프로듀서가 없을 때’로 설정하였고, ERD에서는 ACTOR와 PRODUCER 간의 Connection을 끊었다. 그러면 6번 문항에 대해 True로 대답할 수 있다.

1. A producer cannot be an actor in some other movie   
   False다. PRODUCER와 ACTOR간 연결이 있기 때문에 Producer가 배우가 될 수 없다는 명제는 False가 된다.

* 그러나 6번을 진행하면서 ACTOR와 PRODUCER 테이블간 연결을 끊었기 때문에 이 명제는 True가 될 예정이다.

1. There are movies with more than a dozen actors.   
   Maybe다. 2번과 같은 이유로 12명이상의 배우가 출연하는 영화가 있을 수 있지만, 12명이 아닐 수도 있기 때문이다.

* 그래서 이번에는 조금 독특한 가정을 해보았다. MOVIE 테이블에 존재하기 때문에 10명이하의 팀(배우, 감독, 프로듀서 포함)으로 되어있다는 가정이다. ERD를 수정하지 않으면서도 8번문항에 대해 False로 답할 수 있다. 이는 Trigger나 web 어플리케이션에서 별도로 구현해준다.

1. Some producers have been a director as well.   
   Maybe다. ACTOR와 PRODUCER, DIRECTOR 테이블은 ACTOR 테이블을 통해 연결이 되어있다. 그러므로 몇몇의 Producer는 Director 일 수도 있고, 아닐 수도 있다.

* 이는 5번, 6번에서 ACTOR와 PRODUCER, ACTOR와 DIRECTOR 테이블간의 연결을 끊었기 때문에 자동으로 False가 된다.

1. Most movies have one director and one producer.   
   Maybe다. Most movies라는 표현이 모호할 뿐더러, ERD 상으로는 DIRECTOR는 한 명이지만, PRODUCER는 여러명일 수 있기 때문이다.

* 그래서 PRODUCER와 MOVIE 테이블 간의 관계를 1:N으로 정의하여 ‘모든 영화는 한 명의 감독과 한 명의 프로듀서를 가질 때)라는 가정을 추가해주었다. 10번 문항은 이 가정에 의해 True로 정의된다.

1. Some movies have one director but several producers.   
   Maybe다. MOVIE와 PRODUCER 테이블은 M:N Cardinality로써, 여러 명의 프로듀서를 가질 수도 있지만, 한 명의 프로듀서를 가질 수도 있기 때문이다.

* 이를 10번 문항의 가정을 통해 PRODUCER와 MOVIE 테이블 간의 관계를 1:N으로 정의하여 모든 영화는 프로듀서가 한 명만 존재할 수 있게 해주었다. 이 가정에 의해 11번 문항은 False로 정의된다.

1. There are some actors who have done a lead role, directed a movie, and produced a movie.   
   Maybe다. ACTOR와 PRODUCER, DIRECTOR 테이블은 ACTOR 테이블을 통해 연결이 되어있다. 그러므로 몇몇의 배우는 주연을 맡고, 영화 감독도 했으며, 프로듀서일 수도 있다. 허나 어디까지나 ‘그럴 수도 있다’일 뿐, 확실히 존재한다고 보기는 어렵다.

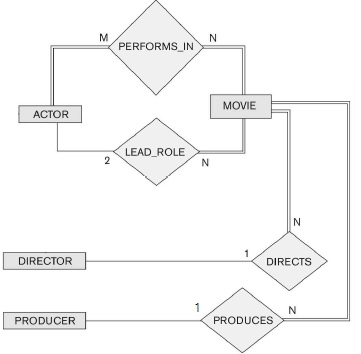
* 이는 5, 6번 문항에서의 가정 ‘ACTOR와 DIRECTOR 테이블 간 연결은 없으며, ACTOR와 PRODUCER 테이블 간 연결은 없다.’를 통해 False로 정의내릴 수 있다.

1. No movie has a director who also acted in that movie.   
   Maybe다. ACTOR와 DIRECTOR 테이블이 연결되어 있기 때문에 배우 겸 감독이 있는 영화는 존재할 수 있다. 그렇지만 필수는 아니기에 존재하지 않을 수도 있다.

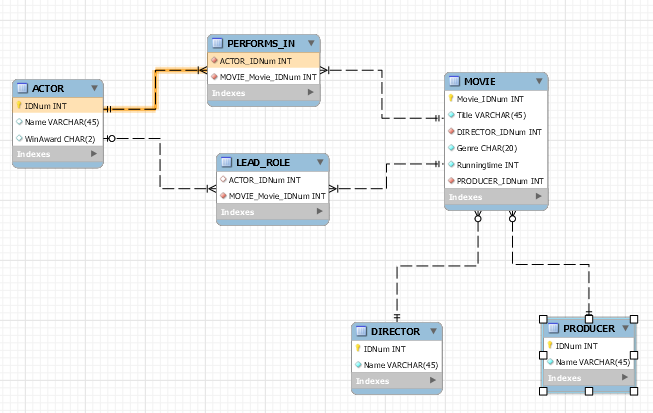
* 이는 5번 문항에서 ACTOR와 DIRECTOR 테이블 간 연결을 끊어주었기 때문에 True로 정의 내릴 수 있다.

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

가정을 위해 수정해준 ERD는 다음과 같다.



이를 workbench위에 나타내면 아래와 같다.



**2-3. MOVIES데이터베이스를 MySQL에 생성할 것(생성한 스크립트를 제출할 것) (40점)**

mysql> create database MOVIES

-> ;

Query OK, 1 row affected (0.01 sec)

mysql> show databases;

+--------------------+

| Database |

+--------------------+

| MOVIES |

| information\_schema |

| movies01 |

| mysql |

| performance\_schema |

| sys |

| testdb |

+--------------------+

7 rows in set (0.00 sec)

mysql> use MOVIES

Database changed

mysql> create table ACTOR

-> (IDNum int not null,

-> Name VARCHAR(45) NOT NULL,

-> WinAward INT NOT NULL,

-> PRIMARY KEY (IDNum))

-> ENGINE = MEMORY;

Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE DIRECTOR

-> ( IDNum INT NOT NULL,

-> Name VARCHAR(45) NOT NULL,

-> PRIMARY KEY (IDNum))

-> ENGINE=MEMORY;

Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE MOVIE (

-> Movie\_IDNum INT NOT NULL,

-> Title VARCHAR(45) NOT NULL,

-> Genre CHAR(20) NOT NULL,

-> Runningtime INT NOT NULL,

-> DIRECTOR\_IDNum INT NOT NULL,

-> PRODUCER\_IDNum INT NOT NULL,

-> PRIMARY KEY (Movie\_IDNum),

-> CONSTRAINT fk\_MOVIE\_DIRECTOR

-> FOREIGN KEY (DIRECTOR\_IDNum)

-> REFERENCES DIRECTOR (IDNum)

-> ON DELETE SET NULL

-> ON UPDATE CASCADE,

-> CONSTRAINT fk\_MOVIE\_PRODUCER

-> FOREIGN KEY (PRODUCER\_IDNum)

-> REFERENCES PRODUCER (IDNum)

-> ON DELETE SET NULL

-> ON UPDATE CASCADE)

-> ENGINE = MEMORY;

Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE PERFORMS\_IN(

-> ACTOR\_IDNum INT NOT NULL,

-> MOVIE\_Movie\_IDNum INT NOT NULL,

-> CONSTRAINT fk\_TO\_ACTOR

-> FOREIGN KEY (ACTOR\_IDNum)

-> REFERENCES ACTOR (IDNum)

-> ON DELETE SET NULL ON UPDATE CASCADE,

-> CONSTRAINT fk\_TO\_MOVIE

-> FOREIGN KEY (MOVIE\_Movie\_IDNum)

-> REFERENCES MOVIE (Movie\_IDNum)

-> ON DELETE SET NULL ON UPDATE CASCADE)

-> ENGINE = MEMORY;

Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE LEAD\_ROLE (

-> ACTOR\_IDNum INT NULL,

-> MOVIE\_Movie\_IDNum INT NOT NULL,

-> CONSTRAINT fk\_LEAD\_TO\_ACTOR

-> FOREIGN KEY (ACTOR\_IDNum)

-> REFERENCES ACTOR (IDNum)

-> ON DELETE SET NULL ON UPDATE CASCADE,

-> CONSTRAINT fk\_LEAD\_TO\_MOVIE

-> FOREIGN KEY (MOVIE\_Movie\_IDNum)

-> REFERENCES MOVIE (Movie\_IDNum)

-> ON DELETE SET NULL ON UPDATE CASCADE)

-> ENGINE = MEMORY;

Query OK, 0 rows affected (0.00 sec)

여기까지가 MOVIES DB를 생성하는데 사용한 스크립트이다. (INSERT문은 생략했으며, INDEX문은 2-4에서 다룰 예정이다. VIEW문은 2-5에서 다룰 예정이다.)

* **Trigger (\*\* 만약 trigger를 작성했다면 함께 제시할 것)**

mysql> delimiter //

mysql> create trigger Lead\_Role\_Violation

-> before insert on LEAD\_ROLE

-> for each row

-> begin

-> set @numof\_leadrole = (SELECT COUNT(\*) FROM LEAD\_ROLE WHERE MOVIE\_Movie\_IDNum=NEW.MOVIE\_Movie\_IDNum);

-> if @numof\_leadrole > 1 then

-> SIGNAL SQLSTATE '45000'

-> SET MESSAGE\_TEXT = 'A movie can have only a maximum of two lead actors';

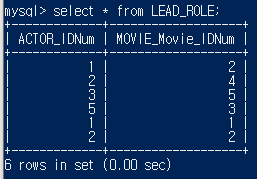
-> end if;

-> end; //

Query OK, 0 rows affected (0.03 sec)

mysql> delimiter ;

위 스크립트는 MOVIES DB를 생성하면서 작성해준 Trigger 문이다. ERD 상으로 2:N이므로 우선 M:N Cardinality 연결을 한 다음, trigger를 통해서 한 영화에 나올 수 있는 최대 주연배우의 수를 2로 제한했다.   
@numof\_leadrole 이라는 변수를 통해 이 변수가 1보다 클 경우 (=2일 경우) 더 이상 배우를 주연배우로서 해당 영화에 출연하지 못하게 막아놓았다.



현재 2번 영화에는 1번 배우와 2번 배우가 주연으로서 출연한다. 즉, 두 명의 주연 배우가 있다.



이 상황에서 3번 배우를 2번 영화의 주연으로 입력하려고 시도했으나, 기작성해둔 Trigger로 인하여 입력되지 않는다. 즉, Trigger가 제대로 동작하고 있음을 확인할 수 있다.

**2-4. 어떤 인덱스를 사용했는지 제시할 것 (20점)**

mysql> CREATE INDEX ACTOR\_idx1 on ACTOR (IDNum) using HASH;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> CREATE INDEX DIRECTOR\_idx1 on DIRECTOR (IDNUM) using HASH;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> CREATE INDEX MOVIE\_idx1 on MOVIE (MOVIE\_IDNum) using HASH;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> CREATE INDEX MOVIE\_idx2 on MOVIE (Runningtime) using BTREE;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

mysql> CREATE INDEX PRODUCER\_idx1 on PRODUCER (IDNum) using HASH;

Query OK, 0 rows affected (0.02 sec)

Records: 0 Duplicates: 0 Warnings: 0

위 스크립트는 인덱스 문의 스크립트이다. 정확한 값을 빠르게 찾기 위해서는 HASH를 사용하는 것이 좋으며, range index. 즉, 특정 범위 안에 있는 값을 빠르게 찾기 위해서는 BTREE를 사용하는 것이 좋다.

그러므로, 범위를 지정하여 검색하기 좋은 MOVIE 테이블의 Runningtime 애트리뷰트만 BTREE 인덱스를 사용하였고, 나머지 테이블의 나머지 애트리뷰트는 HASH 인덱스를 사용하였다.

**2-5. 두 개 이상의 테이블을 조인한 VIEW를 하나 이상 생성할 것(Create view문을 제출할 것) (20점)**

mysql> CREATE VIEW Who\_takes\_lead\_role as

-> select B.Title, A.IDNum, A.Name

-> from ACTOR AS A, MOVIE AS B, LEAD\_ROLE AS C

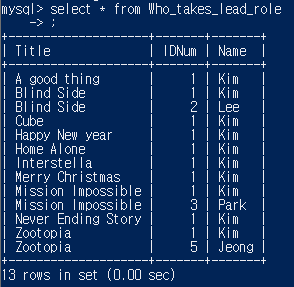
-> WHERE A.IDNum=C.ACTOR\_IDNum

-> AND B.Movie\_IDNum=C.MOVIE\_Movie\_IDNum

-> ORDER BY B.Title;

Query OK, 0 rows affected (0.01 sec)

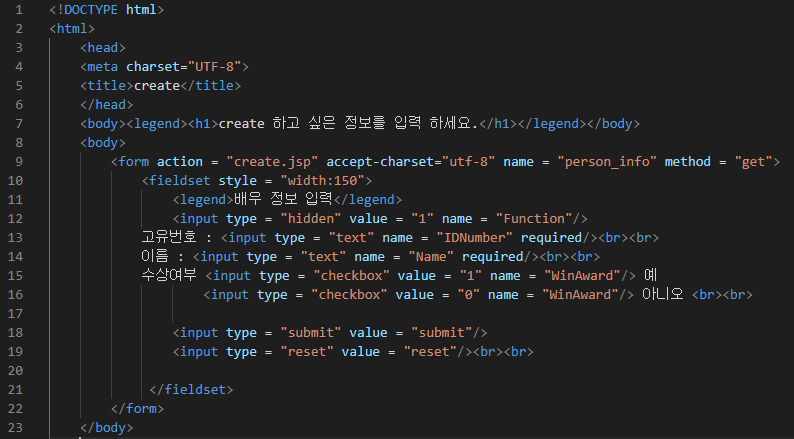
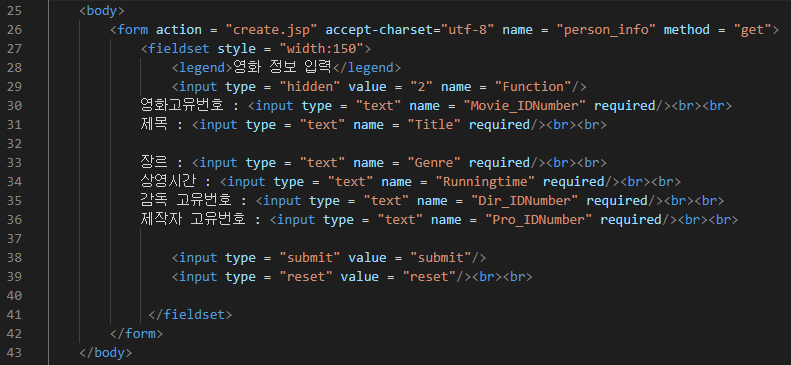
위 스크립트는 누가 어느 영화의 주연을 맡고 있는지를 볼 수 있는 VIEW 문이다.

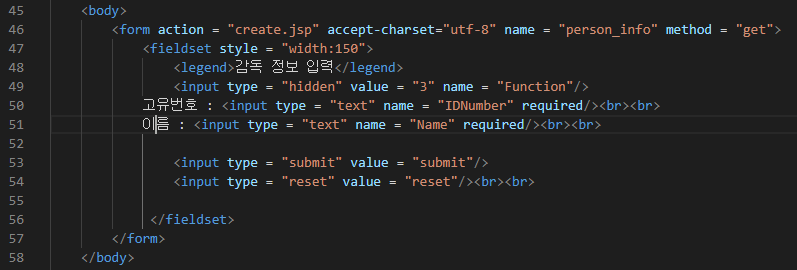
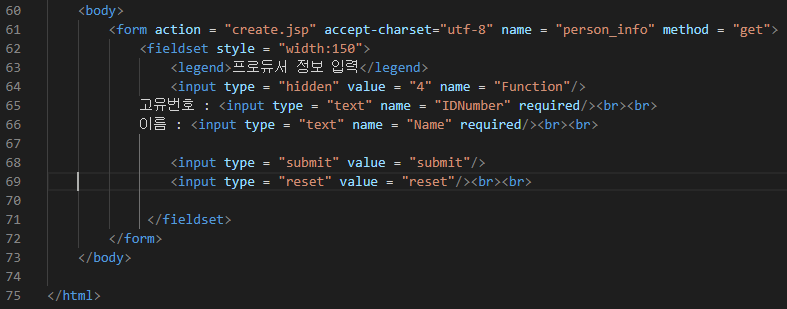


위와 같은 결과가 출력된다. 어느 영화에 누가 주연을 맡았는지 알 수 있다.

**2-6. MOVIES데이터베이스에 대한 삽입, 조회, 수정, 삭제 페이지를 작성할 것 (각 30점씩 총 120점)**

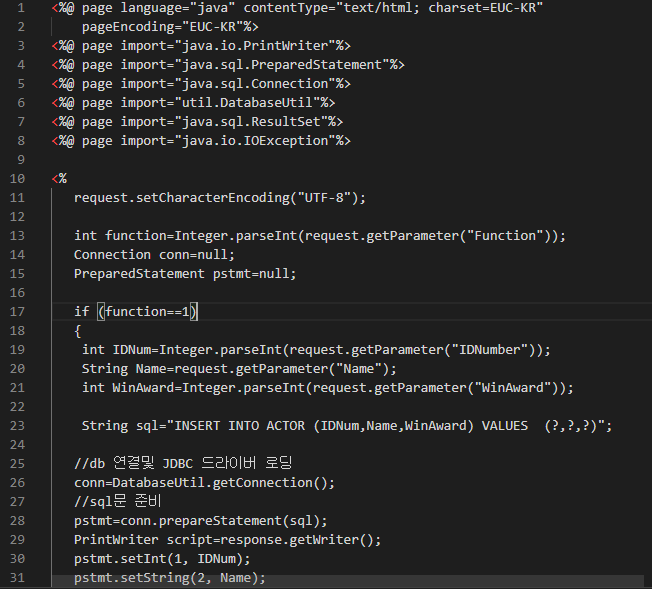
* 삽입 (Create) 페이지 html과 jsp 소스

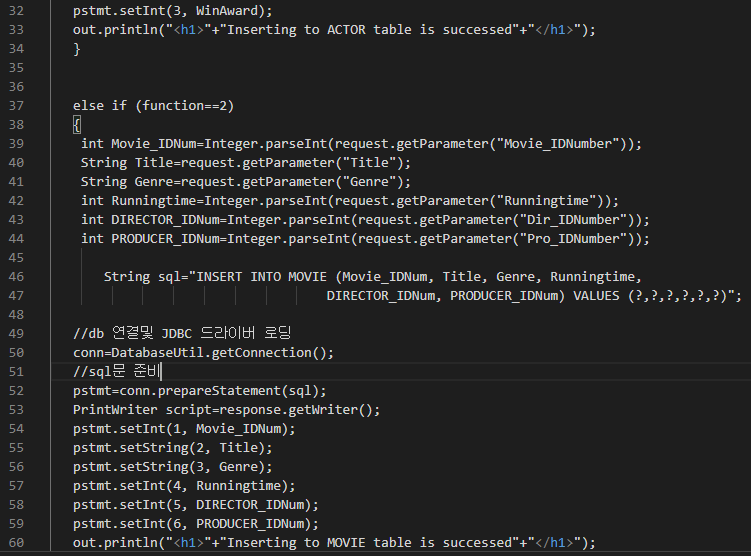
  


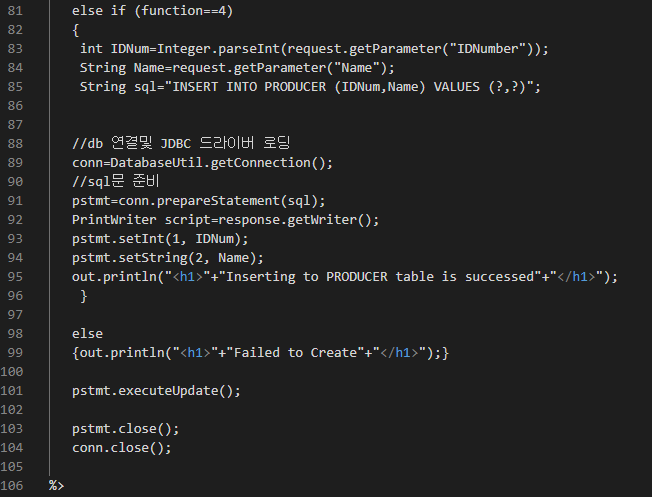
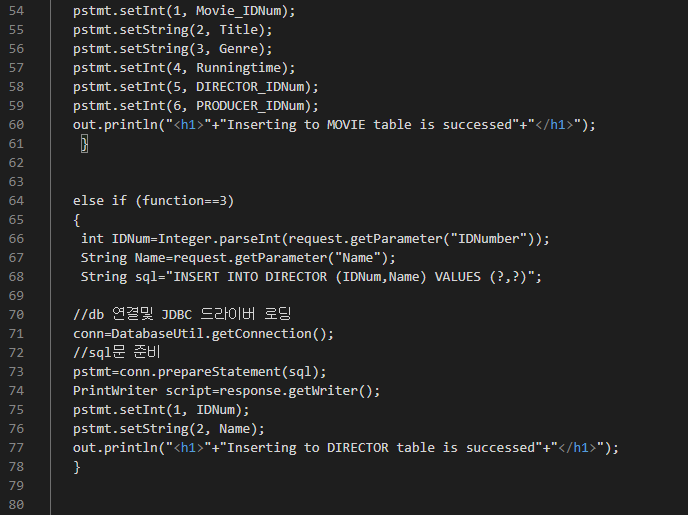
  


(여기까지 Create.html)

기본적인 CRUD 페이지는 학기 초에 했던 실습의 형태를 가져왔다. 각 Table 마다 입력할 수 있도록 body를 나눠놓았으며, 각 body마다 hidden type으로 번호를 매긴 뒤, 이를 통해 create.jsp 문에 들어갔을 시에 행동을 나눠놓았다.



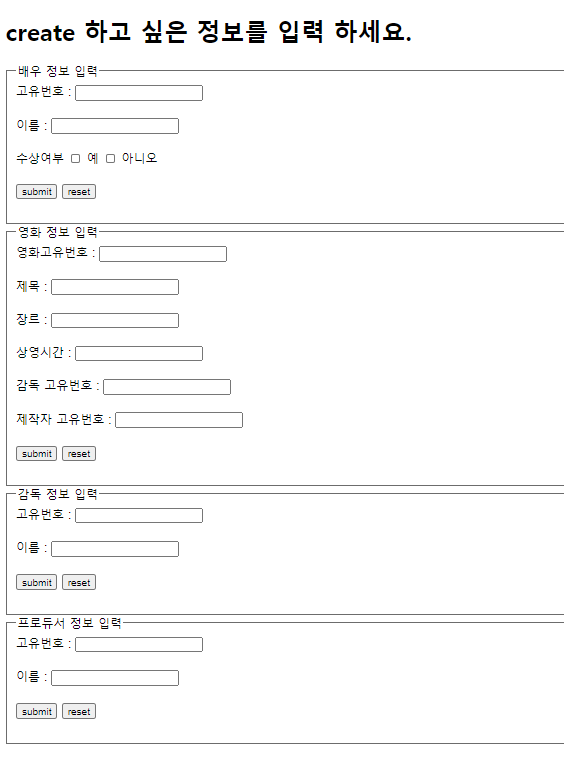




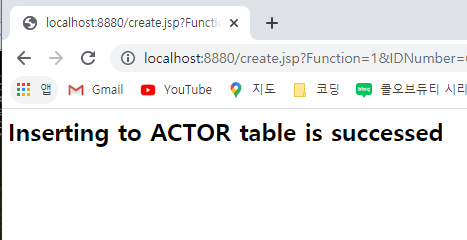
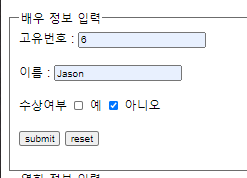
(여기까지가 Create.jsp)

Pstmt를 통해 sql 문을 저장하였고, html에서 입력한 값을 받기 위해 sql문의 특정 변수가 위치하는 자리를 ?로 처리해주었다. 그리고 그 ?에 해당하는 값을 pstmt.setInt or pstmt.setString 등의 함수를 통해 html에서 값을 받아 넣어주도록 하였다.

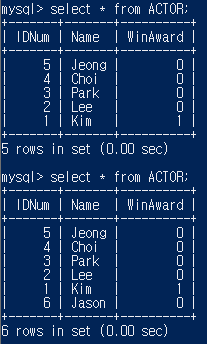
* 삽입 페이지 실행 화면



삽입 페이지는 위와 같이 구현을 하였다. 실행은 맨 위의 ACTOR 테이블에 대한 삽입만 대표로 실행하도록 한다.

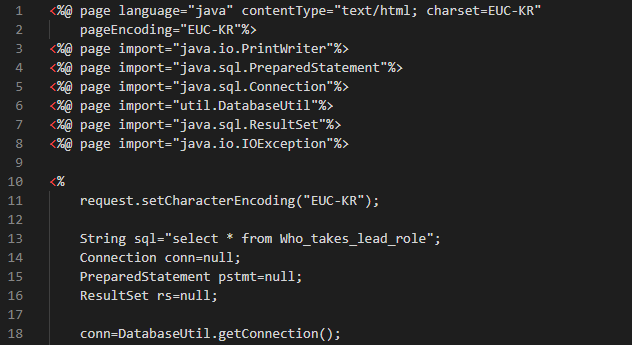


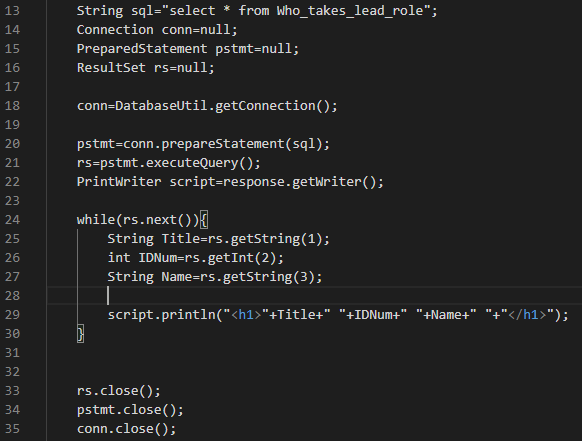
배우 정보에 고유번호와 이름을 입력하고 submit 버튼을 누르면 ACTOR 테이블에 입력되었다는 내용이 출력된다. 이후, Powershell을 통해 table을 확인 해보면,



별도의 입력없이, 웹페이지를 통해 정보가 입력되었음을 확인할 수 있다.

* 조회(Read) 페이지 html과 jsp 소스





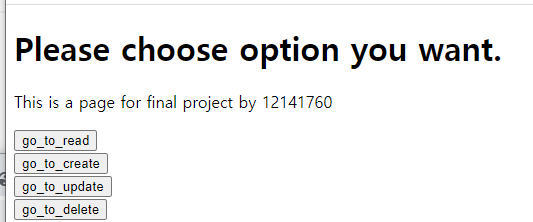
(이상 Read.jsp 소스코드)

조회(Read) 페이지에서는 기존에 만들어놓은 view를 통해 조회하도록 하였다. Who\_takes\_lead\_role 이라는 view를 통해 어떤 영화에서 누가 주연인지를 조회할 수 있도록 하였고, 이를 html에서 받아서 볼 수 있도록 위와 같이 코드를 구성하였다.

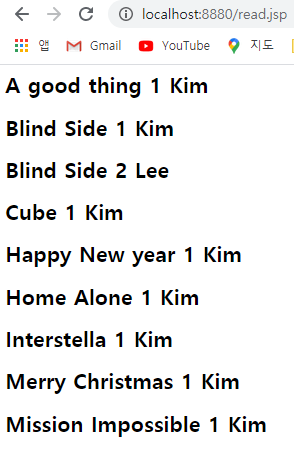
Sql 문으로 select \* from Who\_takes\_lead\_role 을 실행시켜서 웹 위에 출력하는 코드이며, pstmt를 사용해 sql문을 받은 뒤 rs를 통해 실행하고, getString, getInt 등의 함수를 통해 관련 데이터를 가져오는 코드다.

삽입, 수정, 삭제 페이지와는 다르게 조회는 jsp 파일 하나로만 구성되어 있다.

* 조회 페이지 실행 화면

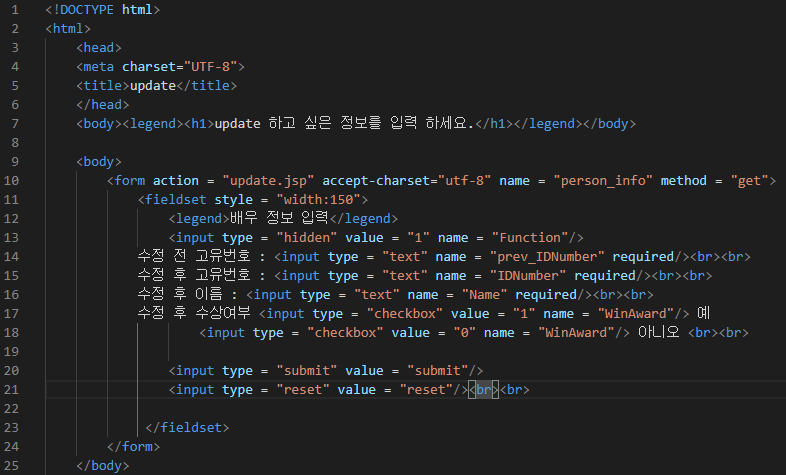
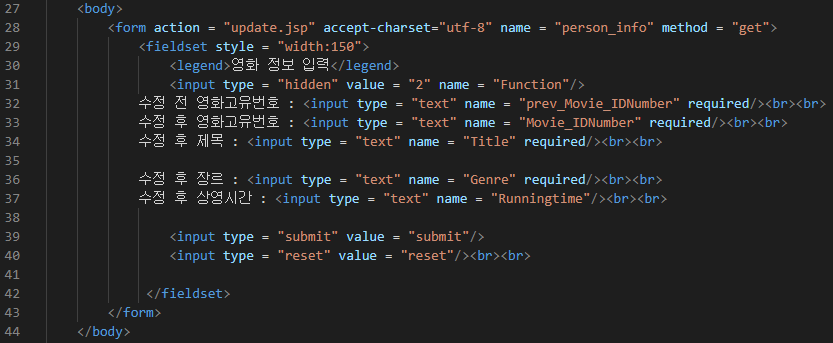
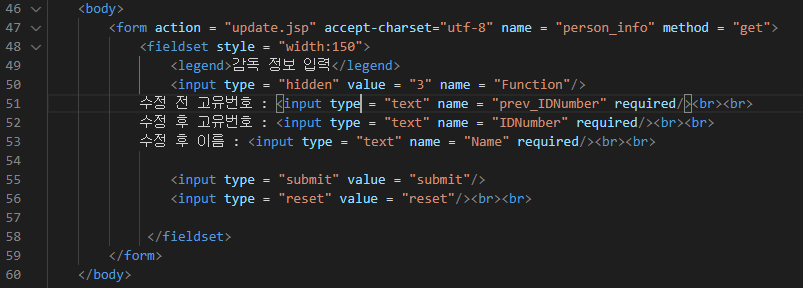


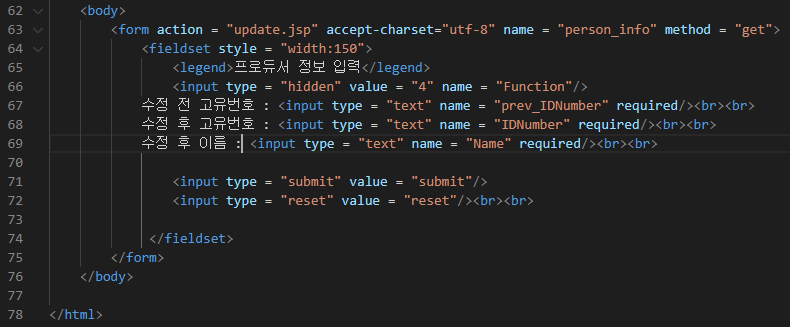
Index.jsp 화면에서 go\_to\_read 버튼을 누르면 바로 조회가 실행되도록 설계하였다.



이런 식으로 Who\_takes\_lead\_role view가 실행된다. 리스트가 다소 길어 후략하도록 한다.

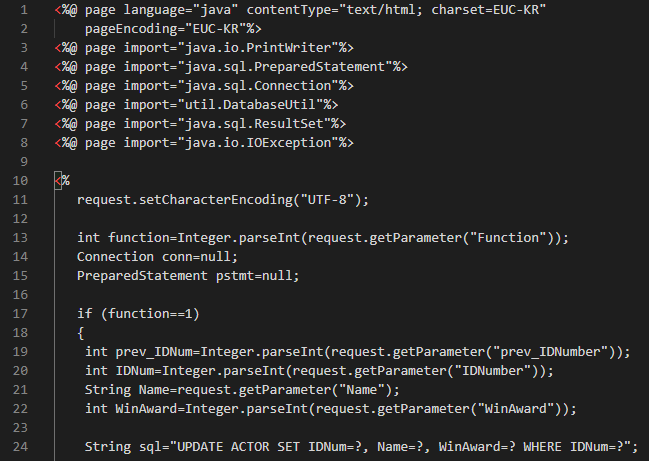
* 수정(Update) 페이지 html과 jsp 소스

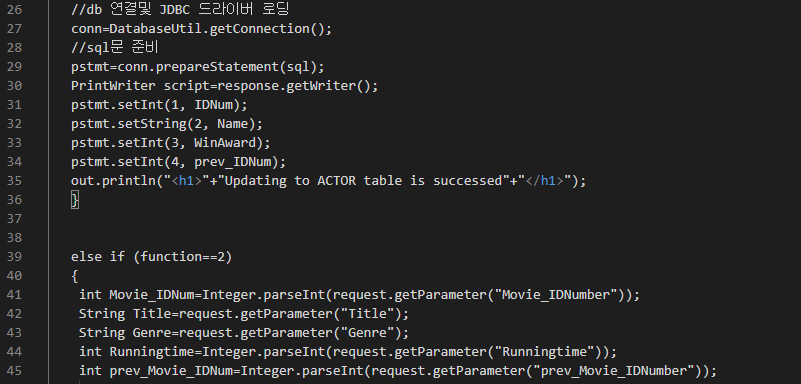
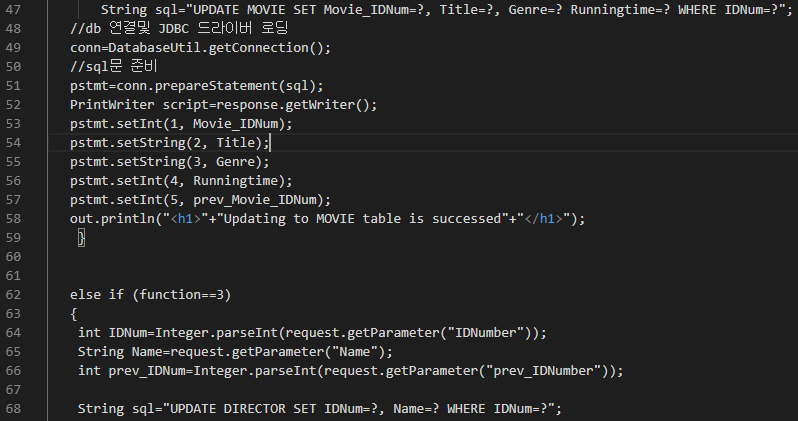
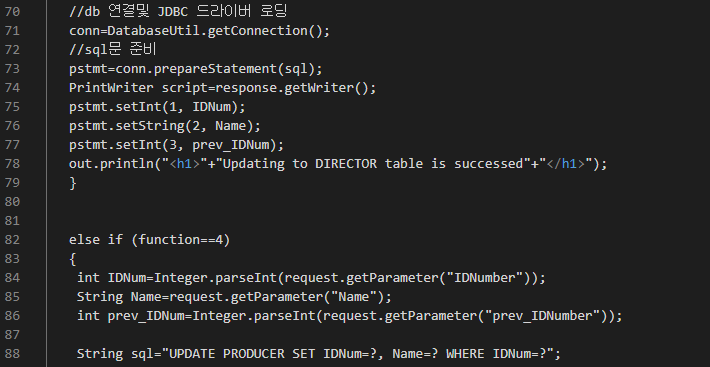
  
  


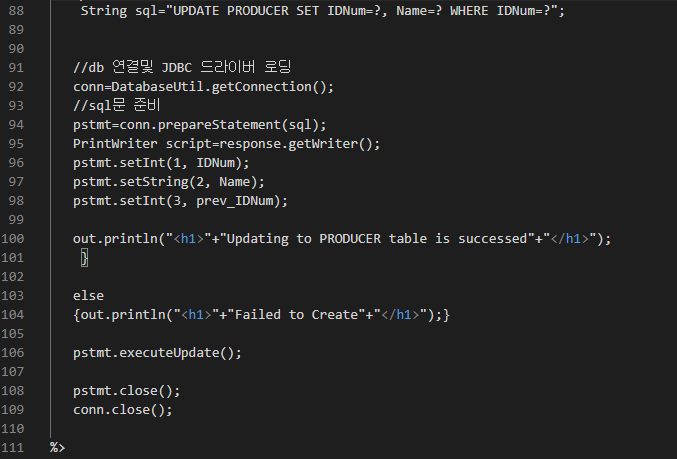


(이상 Update.html 소스 코드)

역시나 각 테이블 (배우, 영화, 감독, 프로듀서) 마다 수정이 가능하도록 했으며, 수정을 원하는 테이블을 선택하고 수정을 원하는 Tuple의 고유번호를 입력한 다음, 수정 후 애트리뷰트를 직접 입력하는 식으로 구현하였다.



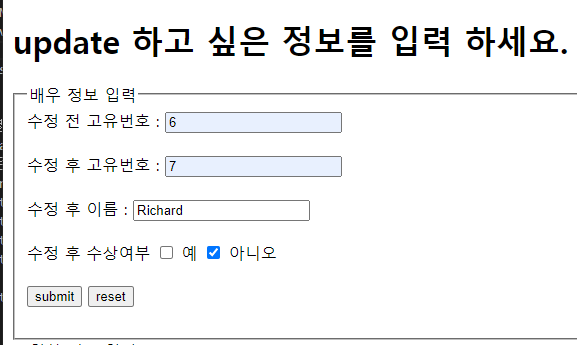


(이상 Update.jsp 소스코드)

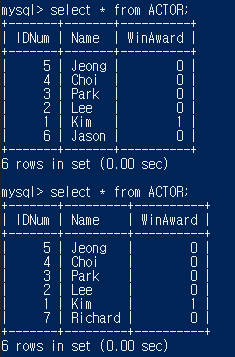
각 Table마다 수정하는 요소를 구현하기 위해 html에서 hidden 형식으로 받아온 값을 if-else 문을 통해 Update 할 테이블을 바꿔주었다. 각 테이블에 맞는 Update 문을 작성하여 각 if 문마다 넣어주었고, 각 테이블의 수정 전 고유번호를 따로 저장했다가 update 문에서 WHERE 조건에 넣어 활용해 주었다.

* 수정(Update) 페이지 실행 화면

대표로 배우 정보만 수정하도록 한다.

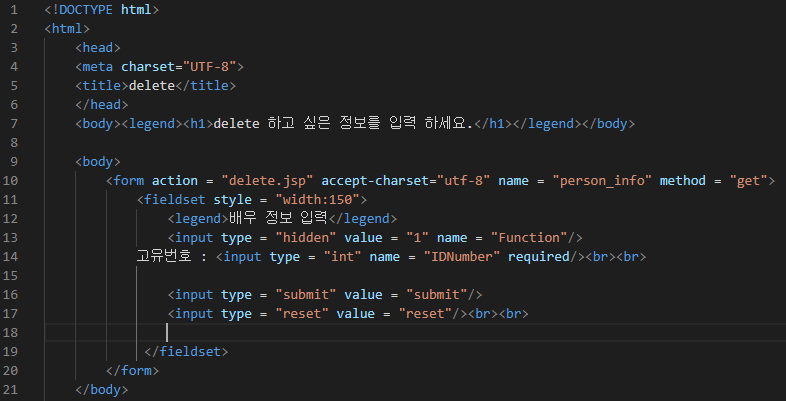


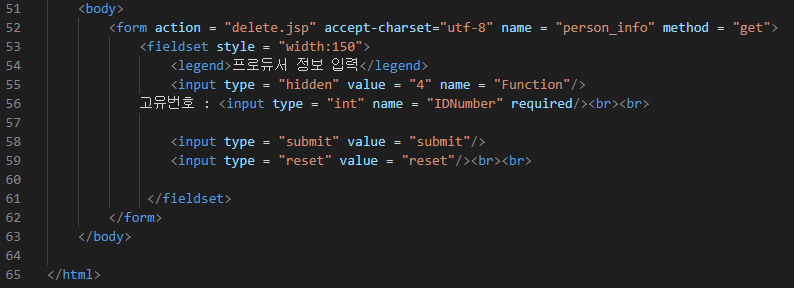
왼쪽 화면은 Update.html 페이지이다. 여기서 배우 정보에 고유번호와 이름을 입력하고 submit 버튼을 누르면 ACTOR 테이블이 수정되었다는 내용이 출력된다. 이는 오른쪽 화면과 같다. 이후, Powershell을 통해 table을 확인 해보면,



별도의 sql문 없이 웹을 통해 테이블에 접근하여 내용을 수정했음을 확인할 수 있다.

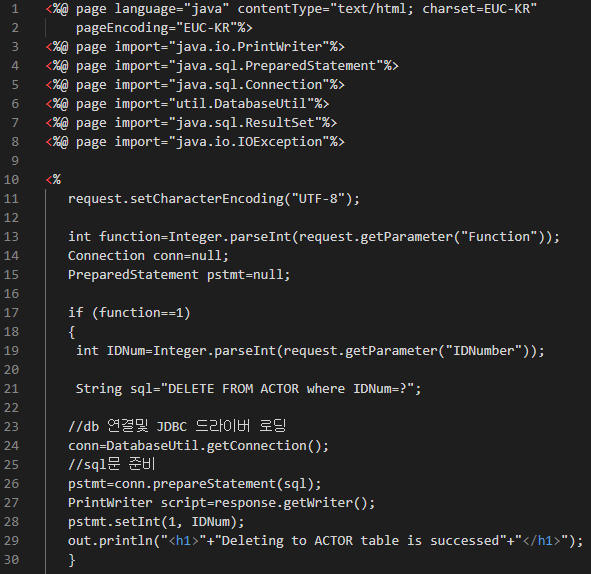
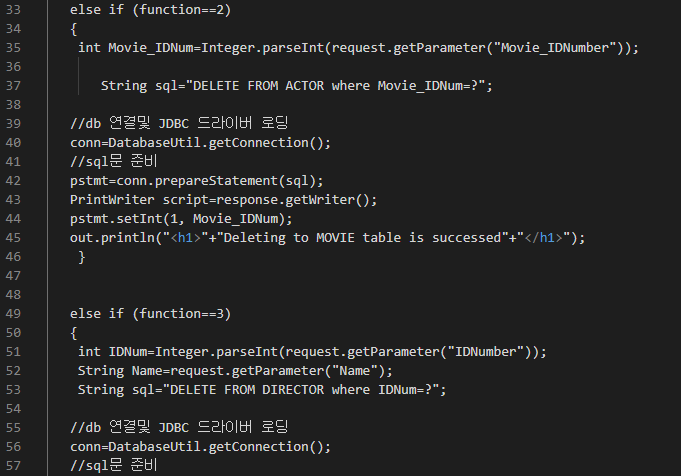
* 삭제(Delete) 페이지 html과 jsp 소스

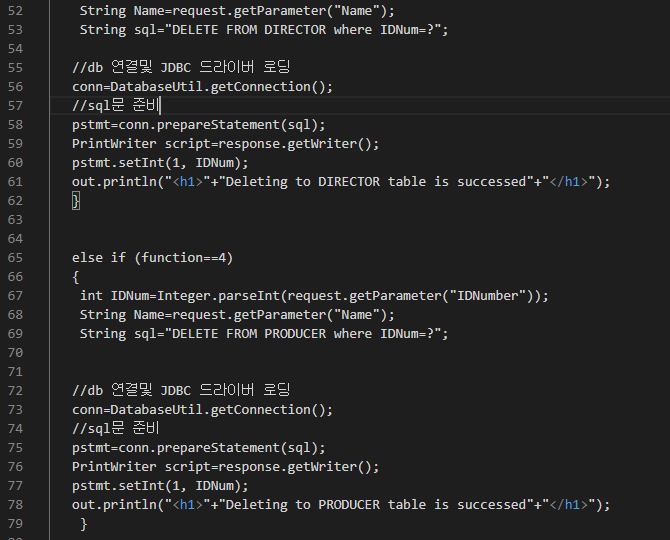
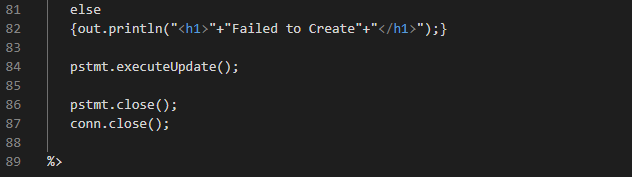


(이상 Delete.html 소스 코드)

역시나 각 테이블 (배우, 감독, 영화, 프로듀서) 마다 원하는 튜플을 삭제할 수 있도록 하였으며, 이는 고유번호로 handling 해주었다.

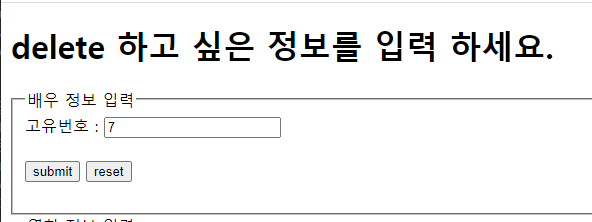
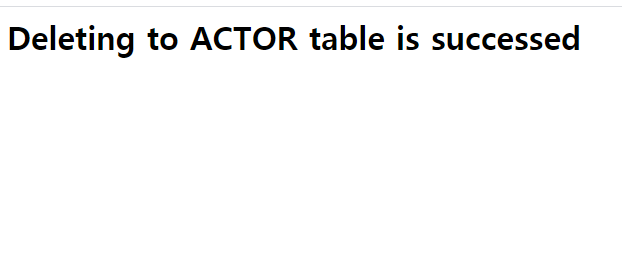
  


(이상 Delete.jsp 소스 코드)

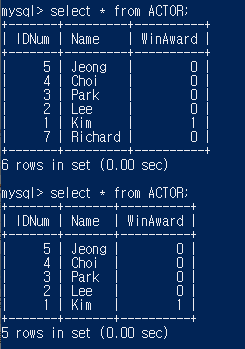
역시나 각 테이블 마다의 hidden value를 통해 해당 테이블의 delete를 구현해주었다. Delete를 하는 Tuple은 delete.html에서 고유번호를 입력 받게 했기 때문에 그것으로 설정해준다.

* 삭제 페이지 실행 결과

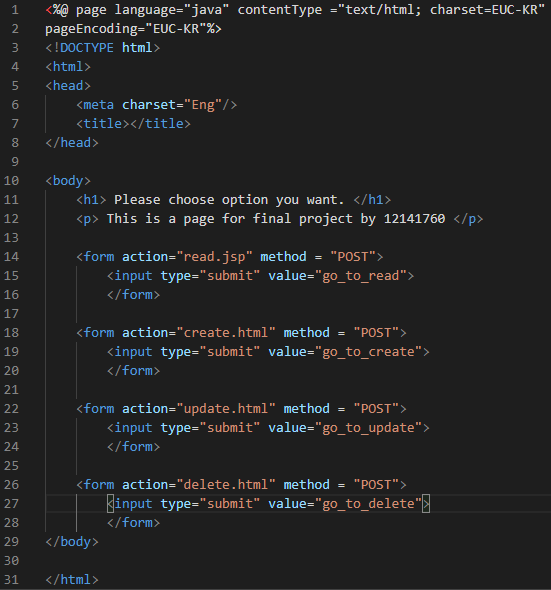
Delete.html은 왼쪽 화면과 같으며, 삭제를 원하는 배우의 고유번호인 7번을 입력하고 submit을 누르게 되면, 오른쪽 화면처럼 삭제가 진행되었다는 문구가 출력된다.

이후 powershell을 통해 database를 확인하면,



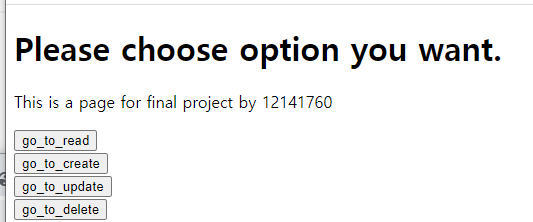
Powershell 내에서의 별도의 삭제 명령이 없음에도 불구하고, 웹으로 delete를 구현하고 실행했음을 확인할 수 있다.

* 메인화면(INDEX) 페이지 html 소스와 실행 화면



(이상 index.html 소스 코드)

그리고 위 CRUD 페이지에 들어가기 위해서 필요한 index.html의 코드다. 가장 기본이 되는 화면이며 실행화면은 아래와 같다.



* DatabaseUtil.Java Servlet 소스



그리고 Docker 내의 환경에 접근하기 위해서 필요한 Database Util 의 소스코드이다. 위에 나와있는 dbURL은 docker inspect mysql을 통해 알 수 있다.

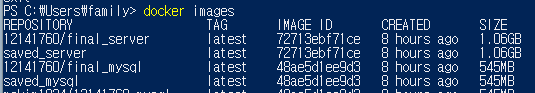
* Docker hub에 업로드한 이미지 이름

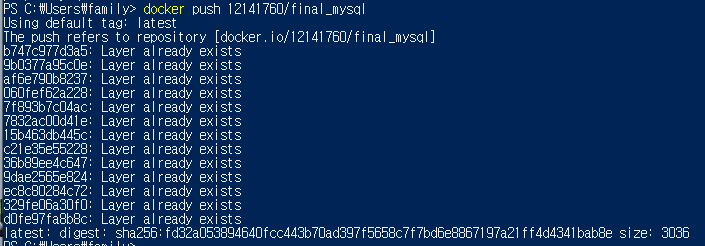

12주차 실습에서 배운 내용을 통해 우선 컨테이너를 이미지로 저장해준다.

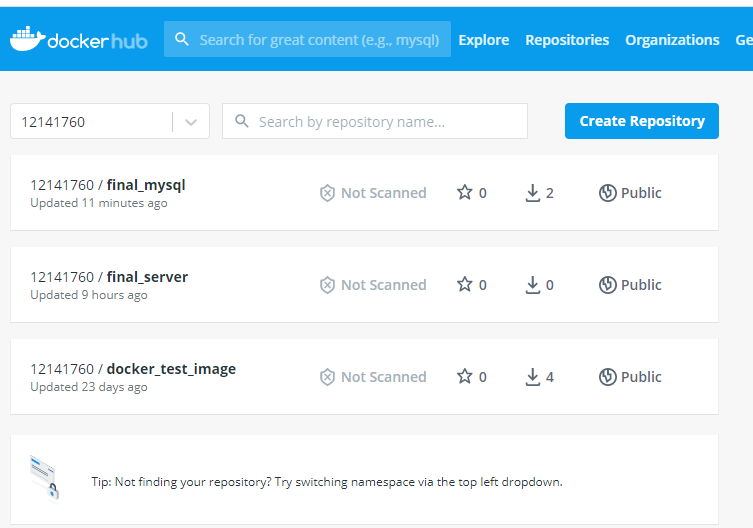

서버도 마찬가지로 저장해준 다음, docker tag 명령어를 사용하여 저장한 이미지의 이름을 지정한다.



Commit과 tag를 마치면 위와 같은 이미지가 생성이 되어 있다.



이를 docker push 명령어를 통해 docker hub에 업로드 한다. 이미 업로드를 하고 powershell을 종료한 뒤, 이미지를 docker hub에 올렸음을 증명하기 위해 보고서에 후첨한 부분이기 때문에 Layer already exists라는 안내 문구가 출력된다.



그리고 Docker Hub를 확인해 보면 성공적으로 final 프로젝트와 서버에 대한 이미지가 업로드 되었음을 확인할 수 있다.

**3. 분석 및 결론**

최종 프로젝트로 데이터베이스를 구현하고, 이를 삽입, 조회, 삭제, 수정하는 웹서비스를 Docker 환경에 구현하여 이를 직접 이용하는 과정을 진행했다.

이번 데이터베이스 프로젝트를 진행하며 테이블의 Storage Engine이 서로 다르면 입력이나 출력이 제대로 이루어지지 않음을 알 수 있었고, 일련의 DBMS 서비스 과정에 대해 이해할 수 있게 되었다. 이를 통해 앞으로 비슷한 형태의 DBMS 및, DBA로써의 역량 등을 키워나가며 프로젝트를 진행할 수 있을 것이다.