INHA UNIVERSITY

DB 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 13 일

학부 정보통신공학과

학년 3학년

성명 김소원

학번 12181748

1. 개요

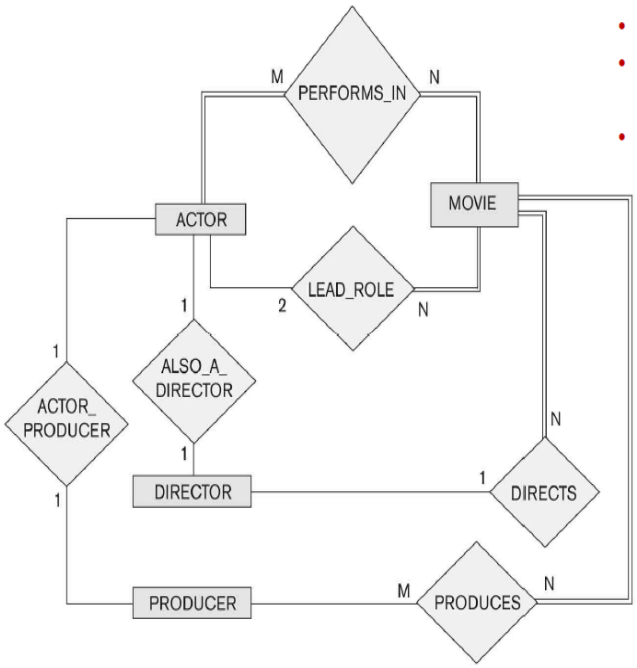
ㅇMOVIES ERD를 기반으로 MOVIES데이타베이스를 설계하고,

ㅇ이를 MySQL에 데이타베이스를 구현한 후, web application을 작성하는 것을 목표

1. 다음 페이지의 MOVIES데이타베이스의 ERD에 대한 답을 제시
2. Maybe에 대한 본인의 가정을 바탕으로 수정한 ERD를 제시
3. MOVIES데이타베이스를 MySQL에 생성(생성한 스크립트 제출)
   - 만약 trigger를 작성했다면 함께 제시
4. 어떤 인덱스를 사용했는지 제시
5. 두 개 이상의 테이블을 조인한 view를 하나이상 생성(create view문 제출)
6. MOVIES데이타베이스에 대한 삽입, 조회, 수정, 삭제 페이지를 작성
   - 최소한의 삽입, 조회(view를 통한 조회), 수정, 삭제 기능으로 재량껏 설계

2. 상세 설계내용

1. MOVIES데이타베이스의 ERD에 대한 답을 제시



- 주어진 ERD에 대해 *True, False*, 또는 *Maybe* 로 답하라.
- 만약 *Maybe* 로 답했다면, 그 문항에 대해서는 *True* 또는 *False* 로 명확히 답할 수 있게끔 본인의 가정을 제시하라.
- 본인의 가정에 따라 ERD를 수정하라.

(1) There are no actors in this database that have been in no movies.
(2) There are some actors who have acted in more than ten movies.
(3) Some actors have done a lead role in multiple movies.
(4) A movie can have only a maximum of two lead actors.
(5) Every director has been an actor in some movie.
(6) No producer has ever been an actor.
(7) A producer cannot be an actor in some other movie.
(8) There are movies with more than a dozen actors.
(9) Some producers have been a director as well.
(10) Most movies have one director and one producer.
(11) Some movies have one director but several producers.
(12) There are some actors who have done a lead role, directed a movie, and produced some movie.
(13) No movie has a director who also acted in that movie.

먼저 주어진 ERD에 대해 TRUE, FALSE 또는 MAYBE로 답하는 것을 해보았다.

(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**

10편 이상의 영화에 출연했던 배우는 없다고 가정하고 false가 되게 함.

(3) Some actors have done a lead role in multiple movies. =>true

(4) A movie can have only a maximum of two lead actors. =>maybe

영화의 주연배우는 2명이상이 될 수 있다 가정하고 false가 되게 함.

(5) Every director has been an actor in some movie. =>maybe

어떤 영화에서는 감독이 배우가 될 수 있다 가정하고 true가 되게 함.

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

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

제작사는 어떤 다른 영화에서 배우가 될 수 있다 가정하고 false가 되게 함.

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

12명 이상의 배우가 출연하는 영화는 없다 가정하고 false가 되게 함.

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

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

모든 영화는 1명의 감독과 1명의 제작자가 있다고 가정을 하고 위 조건을 true가 되게 하였다.

(11) Some movies have one director but several producers.=>**false (일부가 아닌 모두)**

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

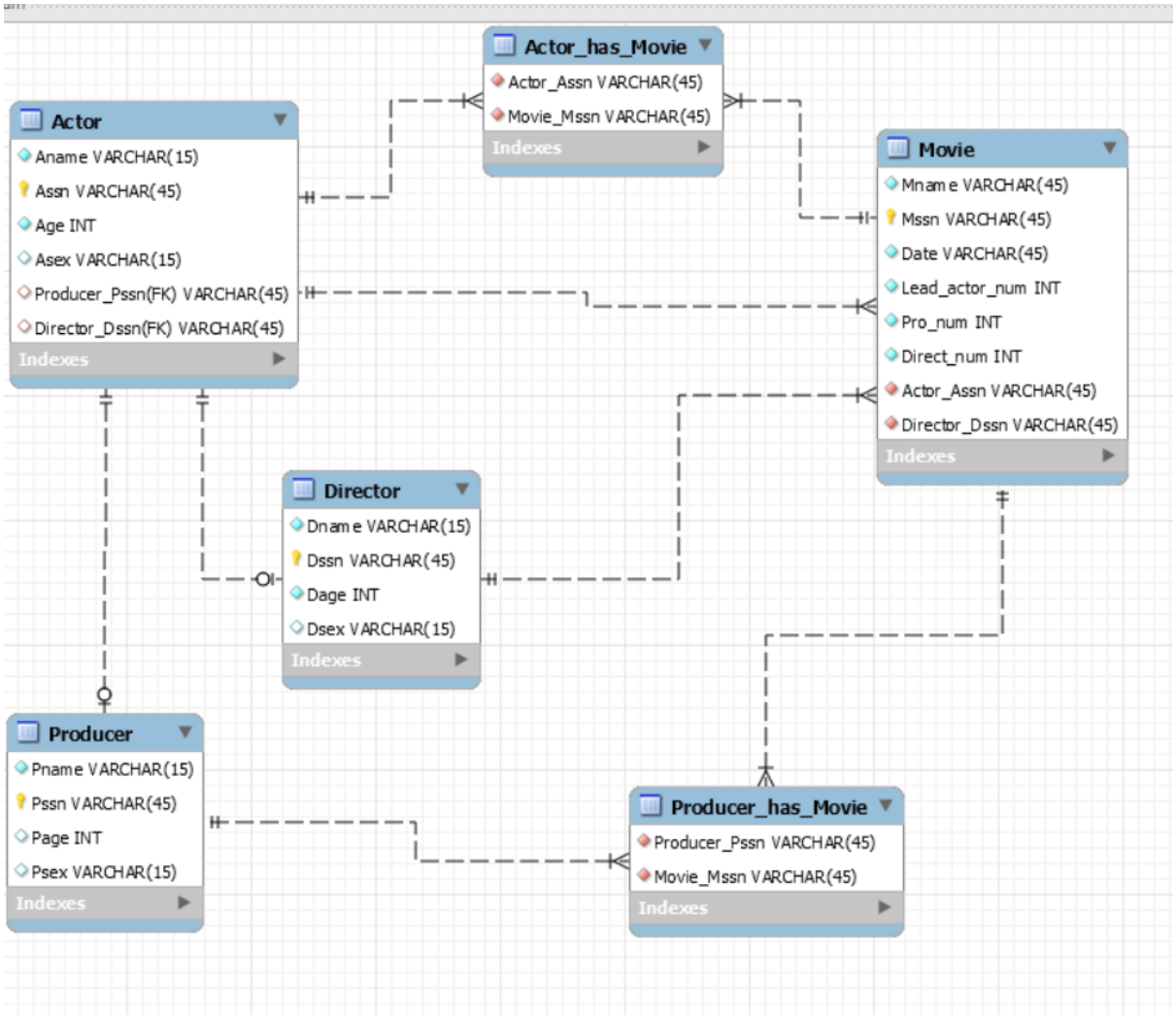
주연, direct, produce를 다 한 배우는 없다고 가정하고 false가 되게 함.

(13) No movie has a director who also acted in that movie=> maybe.

그 영화에 출연하는 감독이 있는 영화는 없다 가정하고 true가 되게 함.

위와 같이 true, false를 정한 후 maybe인 경우에는 바로 밑에 가정을 제시하여 maybe역시 true, false중 한가지가 되도록 하였다. 이제 이것을 바탕으로 workbench에 다이어그램을 작성해

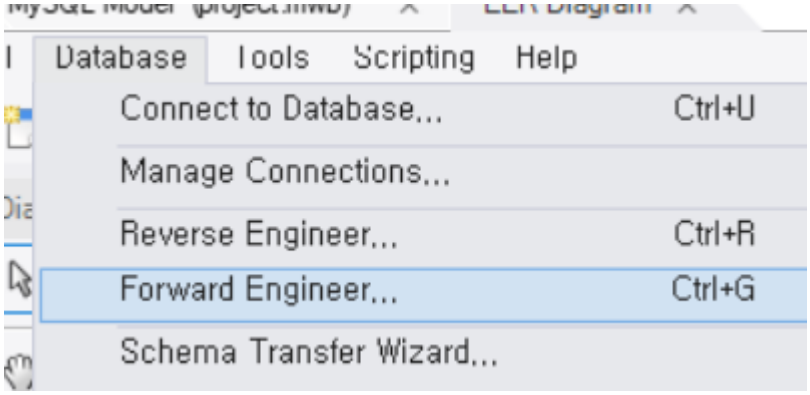
보았다.



Maybe에 대한 나의 가정을 바탕으로 위와 같이 ERD를 만들어보았다.

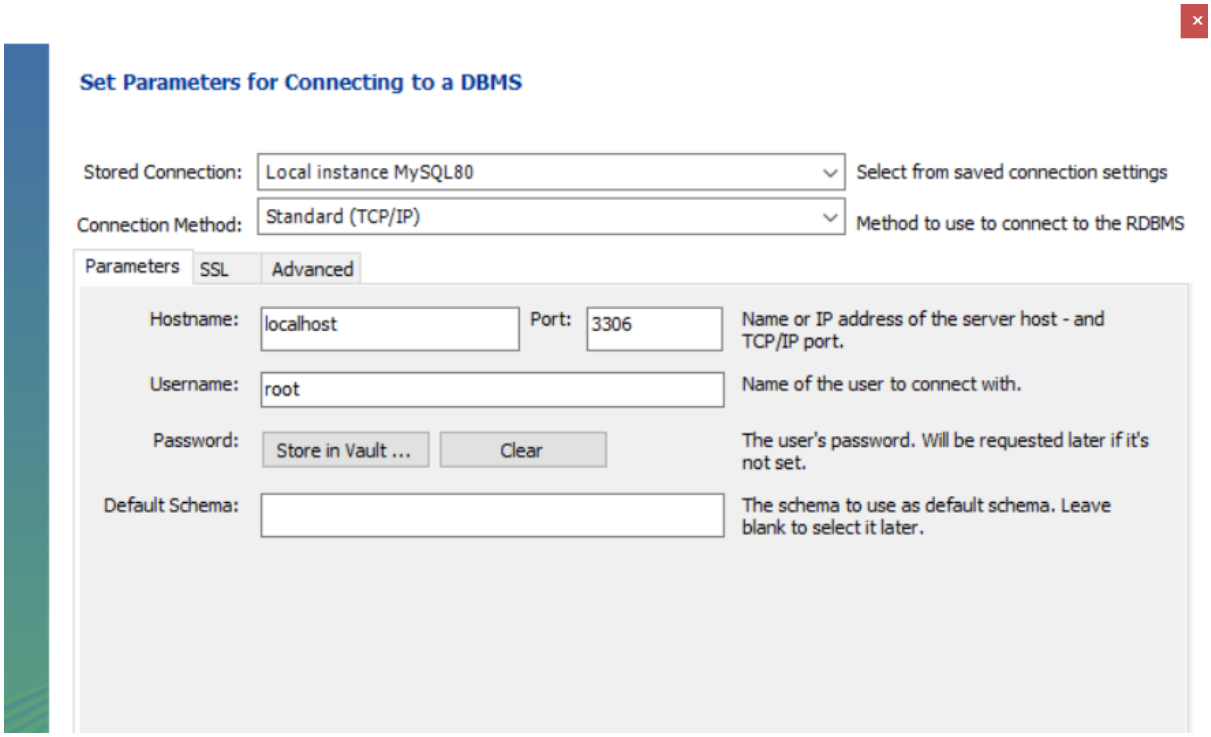
먼저 각각의 테이블에 대한 릴레이션을 만들어주었다. actor는 릴레이션으로 aname, assn, age, asex를 가지고 있으며 producer와 director를 참조받고 있다. 나이인 age만 int type으로 지정해주었으며 다른건 다 varchar타입을 가지고 있다. pk는 주민번호를 의미하는 assn으로 지정해주었다. producer는 pname, pssn, page, psex를 속성으로 가지고 있으며 pk는 제작자의 주민번호를 의미하는 pssn으로 지정해주었다. director는 속성으로 dname, dssn, dage, dsex를 가지고 있으며 이름을 제외한 나머지는 다 varchar타입으로 지정해주었다. pk는 주민번호를 의미하는 dssn으로 설정하였다. movie는 mname, mssn, date, lead_actor_num, pro_num, direct_num으로 속성을 가지고 있으며 영화의 일련번호를 의미하는 mssn이 pk로 지정되었다. actor_has_movie는 배우와 영화 사이의 관계이며 erd에서는 테이블로 보여진다. producer_has_movie 역시 제작자와 영화사이의 관계를 의미하며 테이블로 보여진다. 둘다 m:n관계이다 actor와 producer의 관계는 actor인 사람이 producer도 될 수 있다는 것을 생각으로 1:1관계를 가지며 이때 actor인 사람이

producer가 아닐 수도 있기 때문에 0을 가지도록 해주었다. 마찬가지로 actor와 director의 관계는 actor인 사람이 director도 될 수 있다는 것을 생각으로 1:1관계를 가지며 이때 actor인 사람이 director가 아닐 수도 있기 때문에 0을 가지도록 해주었다. 배우와 영화, 제작자와 영화, 감독과 영화는 다 1:n관계를 가지도록 하였다. 배우 한명이 여러 개의 영화를 찍을 수 있으며 마찬가지로 제작자와 감독 역시 여러 개의 영화를 찍을 수 있다 생각하였기 때문이다. 모두 다 실선이 아닌 점선으로 표현하였다.



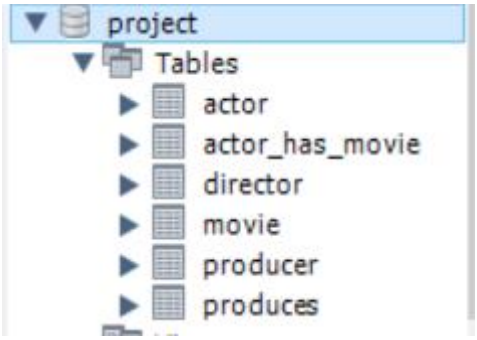
만들어준 모델을 저장하기 위해 database의 forward enginner를 누른다.

forward enginner를 통해 Mysql 서버로 전송이 되어 내가 만든 모델이 반영된다.



아래 상단의 next를 누름

next를 다 누르고 forward enginner가 성공적으로 마무리되면 schemas에 생성되었는지 확인.



schemas에 project란 이름으로 db가 생겨났으며 내가 만든 테이블도 생긴 것을 확인할 수 있었다.

<workbench에서 만든 테이블 스크립트>

```
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ER
ROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema project
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema project
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `project` DEFAULT CHARACTER SET utf8 ;
USE `project` ;

-- -----------------------------------------------------
-- Table `project`.`Producer`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `project`.`Producer` (
  `Pname` VARCHAR(15) NOT NULL,
  `Pssn` VARCHAR(45) NOT NULL,
  `Page` INT NULL,
```

```
  `Psex` VARCHAR(15) NULL,
  PRIMARY KEY (`Pssn`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `project`.`Director`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `project`.`Director` (
  `Dname` VARCHAR(15) NOT NULL,
  `Dssn` VARCHAR(45) NOT NULL,
  `Dage` INT NOT NULL,
  `Dsex` VARCHAR(15) NULL,
  PRIMARY KEY (`Dssn`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `project`.`Actor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `project`.`Actor` (
  `Aname` VARCHAR(15) NOT NULL,
  `Assn` VARCHAR(45) NOT NULL,
  `Age` INT NOT NULL,
  `Asex` VARCHAR(15) NULL,
  `Producer_Pssn(FK)` VARCHAR(45) NULL,
  `Director_Dssn(FK)` VARCHAR(45) NULL,
  PRIMARY KEY (`Assn`),
  INDEX `fk_Actor_Producer1_idx` (`Producer_Pssn(FK)` ASC) VISIBLE,
  INDEX `fk_Actor_Director1_idx` (`Director_Dssn(FK)` ASC) VISIBLE,
  CONSTRAINT `fk_Actor_Producer1`
    FOREIGN KEY (`Producer_Pssn(FK)`)
    REFERENCES `project`.`Producer` (`Pssn`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Actor_Director1`
```

```
    FOREIGN KEY (`Director_Dssn(FK)`)
    REFERENCES `project`.`Director` (`Dssn`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `project`.`Movie`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `project`.`Movie` (
  `Mname` VARCHAR(45) NOT NULL,
  `Mssn` VARCHAR(45) NOT NULL,
  `Date` VARCHAR(45) NOT NULL,
  `Lead_actor_num` INT NOT NULL,
  `Pro_num` INT NOT NULL,
  `Direct_num` INT NOT NULL,
  `Actor_Assn` VARCHAR(45) NOT NULL,
  `Director_Dssn` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`Mssn`),
  INDEX `fk_Movie_Actor1_idx` (`Actor_Assn` ASC) VISIBLE,
  INDEX `fk_Movie_Director1_idx` (`Director_Dssn` ASC) VISIBLE,
  CONSTRAINT `fk_Movie_Actor1`
    FOREIGN KEY (`Actor_Assn`)
    REFERENCES `project`.`Actor` (`Assn`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Movie_Director1`
    FOREIGN KEY (`Director_Dssn`)
    REFERENCES `project`.`Director` (`Dssn`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
```

```
-- Table `project`.`Producer_has_Movie`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `project`.`Producer_has_Movie` (
  `Producer_Pssn` VARCHAR(45) NOT NULL,
  `Movie_Mssn` VARCHAR(45) NOT NULL,
  INDEX `fk_Producer_has_Movie_Movie1_idx` (`Movie_Mssn` ASC) VISIBLE,
  INDEX `fk_Producer_has_Movie_Producer1_idx` (`Producer_Pssn` ASC) VISIBLE,
  CONSTRAINT `fk_Producer_has_Movie_Producer1`
    FOREIGN KEY (`Producer_Pssn`)
    REFERENCES `project`.`Producer` (`Pssn`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Producer_has_Movie_Movie1`
    FOREIGN KEY (`Movie_Mssn`)
    REFERENCES `project`.`Movie` (`Mssn`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `project`.`Actor_has_Movie`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `project`.`Actor_has_Movie` (
  `Actor_Assn` VARCHAR(45) NOT NULL,
  `Movie_Mssn` VARCHAR(45) NOT NULL,
  INDEX `fk_Actor_has_Movie_Movie1_idx` (`Movie_Mssn` ASC) VISIBLE,
  INDEX `fk_Actor_has_Movie_Actor1_idx` (`Actor_Assn` ASC) VISIBLE,
  CONSTRAINT `fk_Actor_has_Movie_Actor1`
    FOREIGN KEY (`Actor_Assn`)
    REFERENCES `project`.`Actor` (`Assn`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Actor_has_Movie_Movie1`
    FOREIGN KEY (`Movie_Mssn`)
    REFERENCES `project`.`Movie` (`Mssn`)
```

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;

SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;

SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

```
C:₩Users₩zzsow>mysql -uroot -p
Enter password: **********
Welcome to the MySQL monitor.  Commands end with ; or ₩g.
Your MySQL connection id is 44
Server version: 8.0.21 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '₩h' for help. Type '₩c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| dbdbdesign         |
| dbdesign           |
| dbdesign1          |
| hw3                |
| information_schema |
| mydb               |
| mysql              |
| performance_schema |
| project            |
| sakila             |
| sys                |
| testdb             |
| world              |
+--------------------+
13 rows in set (0.00 sec)
```

mysql에 접속하여 한번 더 확인해주었다. database에 project가 생겼다.

```
mysql> use project
Database changed
mysql> show tables;
+--------------------+
| Tables_in_project  |
+--------------------+
| actor              |
| actor_has_movie    |
| director           |
| movie              |
| producer           |
| produces           |
+--------------------+
6 rows in set (0.00 sec)

mysql>
```

project안에 위와 같이 내가 만든 테이블이 나오는 것을 확인하였다.

```
mysql> use project
Database changed
mysql> create user 'projectuser'@'%' identified by 'projectuser123!';
Query OK, 0 rows affected (0.04 sec)

mysql> grant allprivileges on project. * to projectuser@'%';
ERROR 3619 (HY000): Illegal privilege level specified for ALLPRIVILEGES
mysql> grant all privileges on project. * to projectuser@'%';
Query OK, 0 rows affected (0.01 sec)

mysql> SET GLOBAL time_zone='+9:00';
Query OK, 0 rows affected (0.00 sec)

mysql> SET time_zone='+9:00';
Query OK, 0 rows affected (0.00 sec)

mysql> flush privileges;
Query OK, 0 rows affected (0.01 sec)
```

데이터베이스project의 아이디와 비밀번호를 정하고 권한을 준 후 severTimeZone을 생성한다.
아이디: projectuser/ 비밀번호: projectuser123!

그 다음으로 각각의 테이블에 내가 임의로 지정한 튜플 값들을 insert해주었다. 이때 pk값은 중복되지 않도록 유의한다.

<producer insert문>

```
mysql> insert into producer
    -> values('Alicia', '998877665', 48, 'F');
Query OK, 1 row affected (0.01 sec)

mysql> insert into producer
    -> values('Ramesh', '111222333', 52, 'M');
Query OK, 1 row affected (0.01 sec)

mysql> insert into producer
    -> values('Joyce', '444555666', 44, 'F');
Query OK, 1 row affected (0.00 sec)

mysql> insert into producer
    -> values('Ahmad', '777888999', 35, 'M');
Query OK, 1 row affected (0.01 sec)
```

<director insert문>

```
mysql> insert into director
    -> values('James', '999888777', 37, 'M');
Query OK, 1 row affected (0.01 sec)

mysql> insert into director
    -> values('Erica', '666555444', 50, 'F');
Query OK, 1 row affected (0.00 sec)

mysql> insert into director
    -> values('Henry', '333222111', 29, 'M');
Query OK, 1 row affected (0.00 sec)

mysql> insert into director
    -> values('Jennifer', '121234343', 33, 'F');
Query OK, 1 row affected (0.01 sec)
```

<actor insert문>

```
mysql> insert into actor(Aname, Assn, Age, Asex)
    -> values ('John', '123456789', 21,'M') ;
Query OK, 1 row affected (0.01 sec)

mysql> insert into actor(Aname, Assn, Age, Asex)
    -> values ('John', '123456789', 21,'M') ;
Query OK, 1 row affected (0.01 sec)

mysql> insert into actor(Aname, Assn, Age, Asex)
    -> values('Rucy', '987654321', 34, 'F');
Query OK, 1 row affected (0.01 sec)

mysql> insert into actor
    -> values('Alicia', '998877665', 48,'F', '998877665', NULL);
Query OK, 1 row affected (0.01 sec)

mysql> insert into actor
    -> values('Jennifer', '121234343', 33, 'F', NULL, '121234343');
Query OK, 1 row affected (0.01 sec)
```

```
mysql> select * from producer;
+--------+-----------+------+------+
| Pname  | Pssn      | Page | Psex |
+--------+-----------+------+------+
| Ramesh | 111222333 |   52 | M    |
| Joyce  | 444555666 |   44 | F    |
| Ahmad  | 777888999 |   35 | M    |
| Alicia | 998877665 |   48 | F    |
+--------+-----------+------+------+
4 rows in set (0.00 sec)
```

```
mysql> select * from director;
+----------+-----------+------+------+
| Dname    | Dssn      | Dage | Dsex |
+----------+-----------+------+------+
| Jennifer | 121234343 |   33 | F    |
| Henry    | 333222111 |   29 | M    |
| Erica    | 666555444 |   50 | F    |
| James    | 999888777 |   37 | M    |
+----------+-----------+------+------+
4 rows in set (0.00 sec)
```

먼저 producer와 director의 insert 스크립트 문을 만들어주었다. 그 후 producer와 director의 테이블을 출력해보면 내가 insert한 튜플들이 들어가 있는 것을 볼 수 있다. producer와 director의 테이블을 먼저 만들어준 이유로는 actor를 먼저 하게 되면 acter안에는 producer와 director의 ssn을 fk로 받는 것이 있기 때문에 producer와 director를 먼저 만들어준 후 해야된다.

```
mysql> select * from actor;
+----------+-----------+-----+------+-----------------+-----------------+
| Aname    | Assn      | Age | Asex | Producer_Pssn(FK) | Director_Dssn(FK) |
+----------+-----------+-----+------+-----------------+-----------------+
| Franklin | 112233445 |  63 | M    | NULL            | NULL            |
| Jennifer | 121234343 |  33 | F    | NULL            | 121234343       |
| John     | 123456789 |  21 | M    | NULL            | NULL            |
| Rucy     | 987654321 |  34 | F    | NULL            | NULL            |
| Alicia   | 998877665 |  48 | F    | 998877665       | NULL            |
+----------+-----------+-----+------+-----------------+-----------------+
5 rows in set (0.00 sec)
```

<movie insert>

```
mysql> insert into movie
    -> values('coco', 'A111111', '2017-11-22', 2, 1, 1, '112233445', '999888777');
```

```
Query OK, 1 row affected (0.01 sec)

mysql> insert into movie
    -> values('coco', 'A111112', '2017-11-22', 2, 1, 1, '123456789', '999888777');
Query OK, 1 row affected (0.01 sec)


mysql> insert into movie
    -> values('car', 'B222222', '2004-05-25', 1, 1, 1, '987654321', '666555444');
Query OK, 1 row affected (0.01 sec)

mysql> insert into movie
    -> value('lion king', 'C333333', '2011-09-03', 1,1,1,'998877665', '333222111');
Query OK, 1 row affected (0.01 sec)

mysql> insert into movie
    -> value('Dumbo', 'D444444', '2007-10-16', 1, 1, 1, '121234343', '333222111');
Query OK, 1 row affected (0.01 sec)

mysql> insert into movie
    -> value('toy story', 'E555555', '2013-08-13', 1, 1, 1, '987654321', '121234343');
Query OK, 1 row affected (0.01 sec)
```

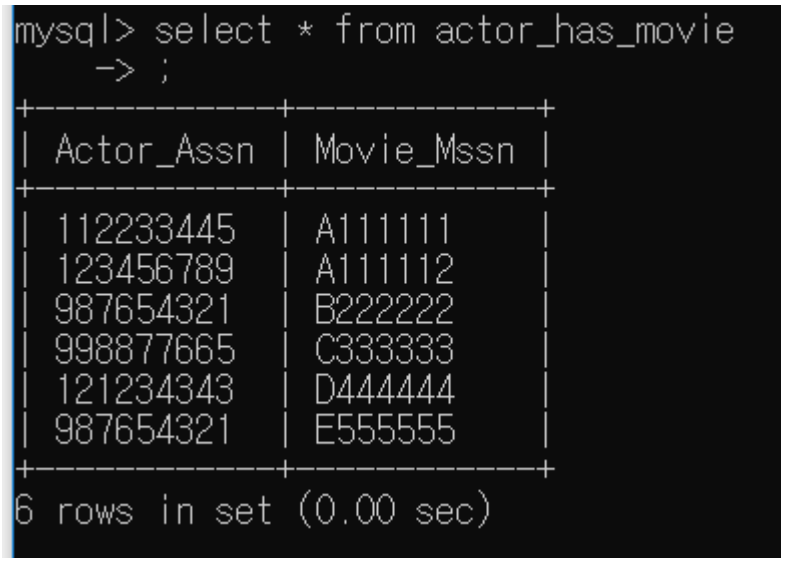
coco라는 제목의 영화에는 2명의 주연배우와 각각 한명의 프로듀서, 디렉터가 있다고 가정하였다. 배우에는 ssn이 112233445인 Franklin과 123456789인 John이 주연배우이며 디렉터는 ssn가 999888777인 James이다.

```
mysql> select * from movie;
+-----------+---------+------------+----------------+---------+------------+------------+---------------+
| Mname     | Mssn    | Date       | Lead_actor_num | Pro_num | Direct_num | Actor_Assn | Director_Dssn |
+-----------+---------+------------+----------------+---------+------------+------------+---------------+
| coco      | A111111 | 2017-11-22 |              2 |       1 |          1 | 112233445  | 999888777     |
| coco      | A111112 | 2017-11-22 |              2 |       1 |          1 | 123456789  | 999888777     |
| car       | B222222 | 2004-05-25 |              1 |       1 |          1 | 987654321  | 666555444     |
| lion king | C333333 | 2011-09-03 |              1 |       1 |          1 | 998877665  | 333222111     |
| Dumbo     | D444444 | 2007-10-16 |              1 |       1 |          1 | 121234343  | 333222111     |
| toy story | E555555 | 2013-08-13 |              1 |       1 |          1 | 987654321  | 121234343     |
+-----------+---------+------------+----------------+---------+------------+------------+---------------+
6 rows in set (0.00 sec)
```

<actor_has_movie 스크립트>

```
mysql> insert into actor_has_movie
```

```
    -> value('112233445', 'A111111');
Query OK, 1 row affected (0.01 sec)

mysql> insert into actor_has_movie
    ->   value('123456789', 'A111112');
Query OK, 1 row affected (0.01 sec)

mysql> insert into actor_has_movie
    -> values('987654321', 'B222222');
Query OK, 1 row affected (0.01 sec)

mysql> insert into actor_has_movie
    -> value('998877665', 'C333333');
Query OK, 1 row affected (0.01 sec)

mysql> insert into actor_has_movie
    -> value('121234343', 'D444444')
    -> ;
Query OK, 1 row affected (0.01 sec)

mysql> insert into actor_has_movie
    -> value('987654321', 'E555555');
```

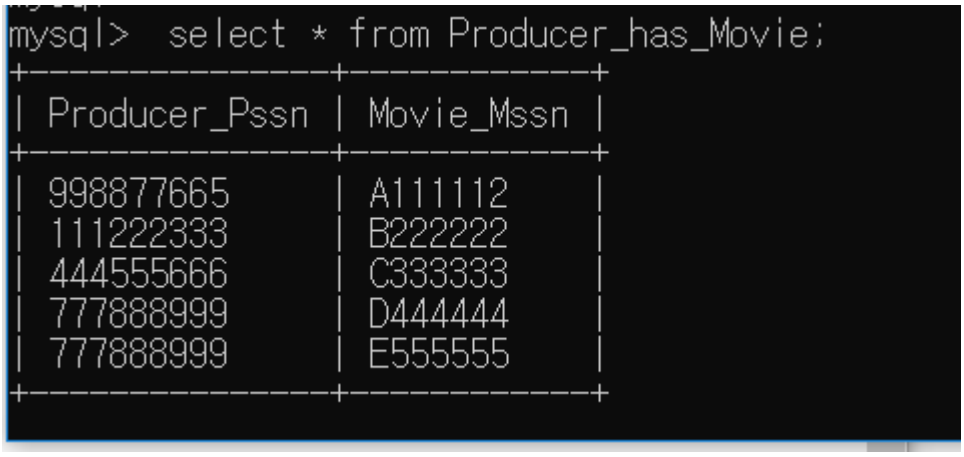


```
mysql> select * from actor_has_movie
    -> ;
+------------+------------+
| Actor_Assn | Movie_Mssn |
+------------+------------+
| 112233445  | A111111    |
| 123456789  | A111112    |
| 987654321  | B222222    |
| 998877665  | C333333    |
| 121234343  | D444444    |
| 987654321  | E555555    |
+------------+------------+
6 rows in set (0.00 sec)
```

<producer_has_movie 스크립트>

```
mysql> insert into producer_has_movie
```

```
    -> value('998877665', 'A111111');
Query OK, 1 row affected (0.01 sec)

mysql> insert into producer_has_movie
    -> value('998877665', 'A111112');
Query OK, 1 row affected (0.01 sec)

mysql> insert into producer_has_movie
    -> value('111222333', 'B222222');
Query OK, 1 row affected (0.00 sec)

mysql> insert into producer_has_movie
    -> value('444555666', 'C333333');
Query OK, 1 row affected (0.01 sec)
mysql> insert into producer_has_movie
    -> value('777888999', 'D444444');
Query OK, 1 row affected (0.01 sec)

mysql> insert into producer_has_movie
    -> value('777888999', 'E555555');
Query OK, 1 row affected (0.01 sec)
```



```
mysql> select * from Producer_has_Movie;
+--------------+------------+
| Producer_Pssn | Movie_Mssn |
+--------------+------------+
| 998877665    | A111112    |
| 111222333    | B222222    |
| 444555666    | C333333    |
| 777888999    | D444444    |
| 777888999    | E555555    |
+--------------+------------+
```

```
mysql> create view actor_producer as
    -> select a.aname, a.age, a.asex, a.assn
    -> from actor a, producer p
    -> where a.assn=p.pssn;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from actor_producer
    -> ;
+--------+-----+------+-----------+
| aname  | age | asex | assn      |
+--------+-----+------+-----------+
| Alicia |  48 | F    | 998877665 |
+--------+-----+------+-----------+
1 row in set (0.00 sec)
```

inner 조인을 사용했으며 actor와 producer의 테이블을 조인하여 배우이면서 동시에 프로듀서인 사람의 정보를 출력하도록 만들어보았다. 그 결과 alicia라는 이름의 48세 여성이 나오는 것을 알 수 있었다. 테이블 이름은 actor_producer이며 이는 내가 insert한 값과 비교하였을 때 맞는 결과이다.

```
mysql> create view actor_director as
    -> select a.aname, a.age, a.asex, a.assn
    -> from actor a, director d
    -> where a.assn=d.dssn;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from actor_director;
+----------+-----+------+-----------+
| aname    | age | asex | assn      |
+----------+-----+------+-----------+
| Jennifer |  33 | F    | 121234343 |
+----------+-----+------+-----------+
1 row in set (0.01 sec)
```

inner 조인을 사용했으며 actor와 director의 테이블을 조인하여 배우이면서 동시에 감독인 사람의 정보를 출력하도록 만들어보았다. 그 결과 jennifer라는 이름의 33세 여성이 나오는 것을 알 수 있었다. 그 결과 alicia라는 이름의 33세 여성이 나오는 것을 알 수 있었다. 테이블 이름은 actor_director이며 이는 내가 insert한 값과 비교하였을 때 맞는 결과이다.

```
C:₩Users₩zzsow>docker exec -it mysql /bin/bash
root@c4d908bf4cff:/# mysql -uroot -p
Enter password:
ERROR 1045 (28000): Access denied for user 'root'@'localhost' (using password: YES)
root@c4d908bf4cff:/# mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or ₩g.
Your MySQL connection id is 9
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '₩h' for help. Type '₩c' to clear the current input statement.
```

도커에 이미지를 올리기 위해 위에서 만들었던 모든 쿼리문을 docker의 mysql로 다시 접속하여 다 넣어주었다. 이때 도커안의 mysql문은 대소문자를 구분하는 것을 알고 주의해야하였다.

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| project            |
| sys                |
| testdb             |
+--------------------+
6 rows in set (0.00 sec)
```

```
mysql> use project
Database changed
mysql> show tables;
+--------------------+
| Tables_in_project  |
+--------------------+
| Actor              |
| Actor_has_Movie    |
| Director           |
| Movie              |
| Producer           |
| Producer_has_Movie |
+--------------------+
6 rows in set (0.00 sec)

mysql>
```

붙여넣기 후 도커의 mysql에도 모든 데이터가 다 들어간 것을 확인하였다.

그 다음으로 jsp, html등 각각의 소스들을 만들어주었다.

<ProjectConnection.java>

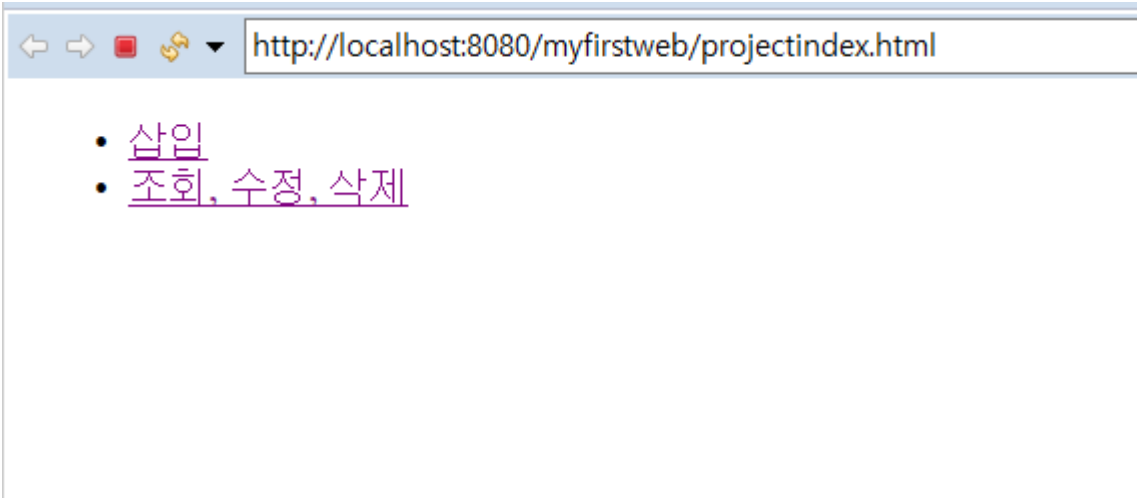
```java
package examples;

import java.sql.Connection;

public class ProjectConnection {
    public static Connection getCon() throws SQLException{
        Connection con=null;
        try{
            Class.forName("com.mysql.cj.jdbc.Driver");
            String url = "jdbc:mysql://localhost/project?useUnicode=true&useJDBCCompliantTimezoneShift=true&useLegacyDatetimeCode=false&serverTimezone=UTC
            con=DriverManager.getConnection(url, "projectuser", "projectuser123!");
            return con;
        }catch(ClassNotFoundException ce){
            System.out.println(ce.getMessage());
            return null;
        }
    }
}
```

먼저 myfirstweb=>Java Resources=>examples 안에 ProjectConnection.java 란 이름으로 자바파일을 만들어준다. 내가 이전에 만들어둔 sql 문과 연결하기 위함이며 내가 지정해뒀던 데이터베이스 project 설정한 아이디, 패스워드를 넣어 연결해준다.

<projectindex.html>

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<ul>
    <li><a href="projectinsert.html">삽입</a></li>
    <li><a href="projectlist.jsp"> 조회, 수정, 삭제</a></li>
</ul>
</body>
</html>
```

myfirstweb=>WebContent 안에 projectindex 란 이름으로 html 을 만들어준다.
MOVIES 데이타베이스에 대한 삽입, 조회, 수정, 삭제 페이지에 접속하기 위해 처음 시작하는 메인창으로 삽입을 누르면 projectinsert.html 에 들어가 삽입을 수행하게 해주며 조회는 projectlist.jsp 를 눌러 movie 에 대한 정보를 볼 수 있게 해주었다. 여기서 리스트들을 삭제 수정도 할 수 있게 해주었다.

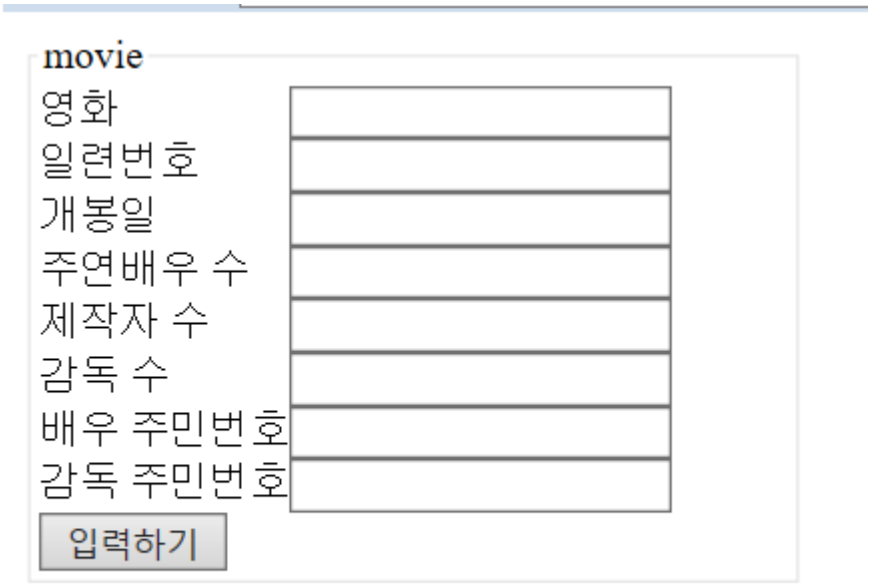


이클립스에서 실행하면 위와 같이 된다.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www
2  <html>
3  <head>
4  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5  <title>Insert title here</title>
6  <style type="text/css">
7      #regbox{    width : 300px;  }
8      #regbox label{  display : block;        width: 100px;       float : le
9  </style>
10 </head>
11 <body>
12 <form method="post" action="projectinsert.jsp">
13     <fieldset id="regbox">
14         <legend>movie</legend>
15     <label for="mname">영화</label>
16       <input type="text" name="mname"/><br/>
17
18       <label for="mssn">개봉일</label>
19       <input type="text" name="mssn"/><br/>
20
21       <label for="date">일련번호</label>
22       <input type="text" name="date"/><br/>
23
24       <label for="lead_actor_num">주연배우 수</label>
25       <input type="text" name="lead_actor_num"/><br/>
26
27       <label for="pro_num">제작자 수</label>
28       <input type="text" name="pro_num"/><br/>
29
30       <label for="direct_num">감독 수</label>
31       <input type="text" name="direct_num"/><br/>
32
33       <label for="actor_assn">배우 주민번호</label>
```

```
35
36          <label for="director_dssn">감독 주민번호</label>
37          <input type="text" name="director_dssn"/><br/>
38
39          <input type="submit" value="입력하기">
40      </fieldset>
41  </form>
42  </body>
43  </html>
```

myfirstweb=>WebContent안에 projectinsert.html을 만들어주었다. 무비 테이블에 필요한 릴레이션인 Mname, Mssn, Lead_actor_num , Pro_num , Direct_num , Actor_Assn , Director_Dssn에 맞게 투플들을 insert하게 해주었으며 text로 타입을 지정해주었으며 insert문이 완성되면 입력하기 버튼을 누를 수 있도록 만들어주었다.



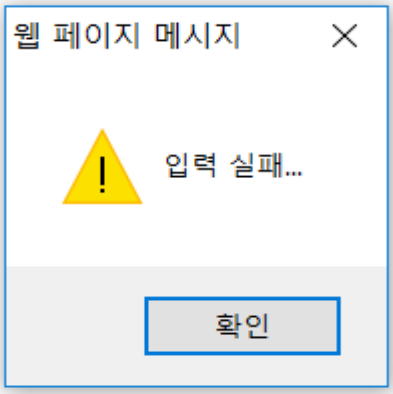
이클립스에서 실행하면 무비에 대한 릴레이션 이름들과 그것을 입력할 수 있는 창이 나오며 젤 밑에는 입력하기 버튼이 나오게 된다.

<projectinsert.jsp>

```jsp
<%@page import="java.sql.SQLException"%>
<%@page import="java.sql.PreparedStatement"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.sql.Connection" %>
<%@page import="examples.ProjectConnection"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
    request.setCharacterEncoding("UTF-8");
    String mname=request.getParameter("mname");
    String mssn=request.getParameter("mssn");
    String date=request.getParameter("date");
    String lead_actor_num=request.getParameter("lead_actor_num");
    String pro_num=request.getParameter("pro_num");
    String direct_num=request.getParameter("direct_num");
    /* int lead_actor_num=Integer.parseInt(request.getParameter("lead_actor_num"))
    int pro_num=Integer.parseInt(request.getParameter("pro_num"));
    int direct_num=Integer.parseInt(request.getParameter("direct_num")); */
    String actor_assn=request.getParameter("actor_assn");
    String director_dssn=request.getParameter("director_dssn");
    //db에 저장하기
    Connection con = null;
    PreparedStatement pstmt = null;
    String sql = "insert into movie values(?,?,?,?,?,?,?,? sysdate)";
    int n=0;

    try{
```

```
39          pstmt.setString(4, lead_actor_num);
40          pstmt.setString(5, pro_num);
41          pstmt.setString(6, direct_num);
42          /* pstmt.setInt(4, lead_actor_num);
43          pstmt.setInt(5, pro_num);
44          pstmt.setInt(6, direct_num); */
45          pstmt.setString(7, actor_assn);
46          pstmt.setString(8, director_dssn);
47
48          n = pstmt.executeUpdate();
49      }catch(SQLException se){
50          System.out.println(se.getMessage());
51      }finally{
52          try{
53              if(pstmt!=null) pstmt.close();
54              if(con!=null) con.close();
55          }catch(SQLException se){
56              System.out.println(se.getMessage());
57          }
58      }
59      // 결과 응답하기
60  %>
61  <script type="text/javascript">
62      if(<%=n%> > 0){
63          alert("입력완료!");
64          location.href="../projectindex.html";//
65      }else{
66          alert("입력 실패...");
67          history.go(-1);//이전페이지로 가기
68      }
69  </script>
70  </body>
71  </html>
```

myfirstweb=>WebContent안에 projectinsert.jsp를 만들어주었다. 입력을 잘 받았으면 입력완료란 문구가 뜨게하고 입력이 잘 안되었을 경우엔 "입력 실패..."란 문구와 함께 이전페이지로 다시 돌아가도록 설정해주었다. location.herf는 바로 앞서 만든 projectindex.html과 연결해주었다



이클립스에서 projectinsert.jsp를 실행시켜주면 위와 같이 입력 실패...”라는 문구가 떴다. 아직 내가 입력을 하지 않았기 때문에 입력실패...인것이다.

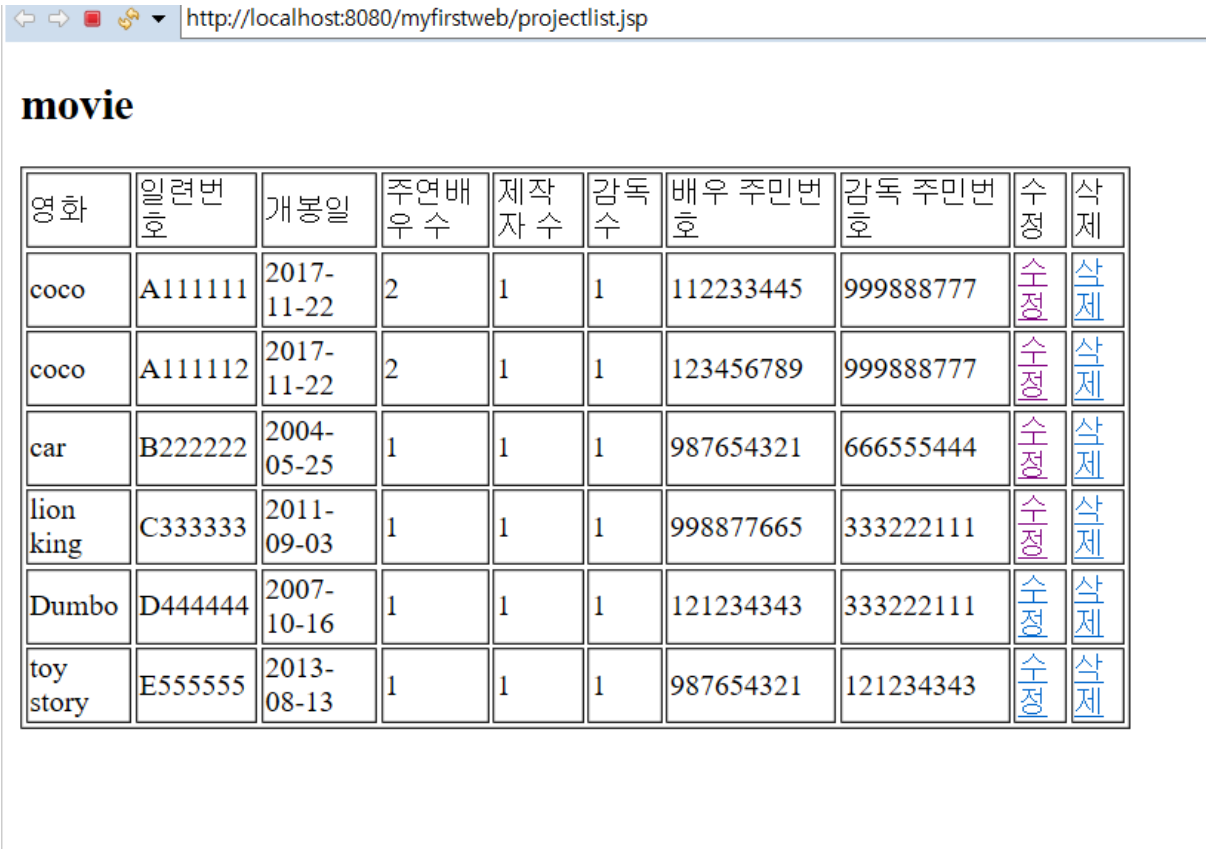
<projectlist.jsp>

```
<%@page import="examples.ProjectConnection"%>
<%@page import="java.sql.Timestamp"%>
<%@page import="java.sql.SQLException"%>
<%@page import="java.sql.ResultSet"%>
<%@page import="java.sql.PreparedStatement"%>
<%@page import="java.sql.Connection"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>movie</h2>
<table border="1" width="600">
    <tr>
        <td>영화</td>
        <td>개봉일</td>
        <td>일련번호</td>
        <td>주연배우 수</td>
        <td>제작자 수</td>
        <td>감독 수</td>
        <td>배우 주민번호</td>
        <td>감독 주민번호</td>
        <td>수정</td>
        <td>삭제</td>
    </tr>
    <%
        //db 에서 회원목록 얻어와 테이블에 출력하기.
        Connection con=null;
        PreparedStatement pstmt=null;
```

```
35        try{
36            con = ProjectConnection.getCon();
37            String sql="select * from movie";
38            pstmt = con.prepareStatement(sql);
39            rs = pstmt.executeQuery();
40            while(rs.next()){
41
42                String mname=rs.getString("mname");
43                String mssn=rs.getString("mssn");
44                String date=rs.getString("date");
45                String lead_actor_num=rs.getString("lead_actor_num");
46                String pro_num=rs.getString("pro_num");
47                String direct_num=rs.getString("direct_num");
48                /* int lead_actor_num=Integer.parseInt(rs.getString("lead_actor_num"));
49                int pro_num=Integer.parseInt(request.getParameter("pro_num"));
50                int direct_num=Integer.parseInt(request.getParameter("direct_num")); */
51                String actor_assn=rs.getString("actor_assn");
52                String director_dssn=rs.getString("director_dssn");
53
54
55 %>
56    <tr>
57        <td><%=mname %></td>
58        <td><%=mssn %></td>
59        <td><%=date %></td>
60        <td><%=lead_actor_num %></td>
61        <td><%=pro_num %></td>
62        <td><%=direct_num %></td>
63        <td><%=actor_assn %></td>
64        <td><%=director_dssn %></td>
65        <td><a href="projectupdate.jsp?mname=<%=mname%>" >수정</a></td>
66        <td><a href="projectdelete.jsp?mname=<%=mname%>" >삭제</a></td>
67    </tr>
68 <%
69            }
70        }catch(SQLException se){
71            System.out.println(se.getMessage());
72        }finally{
```

```
65          <td><a href="projectupdate.jsp?id=<%=mname%>" >수정</a></td>
66          <td><a href="projectdelete.jsp?id=<%=mname%>" >삭제</a></td>
67      </tr>
68  <%
69              }
70          }catch(SQLException se){
71              System.out.println(se.getMessage());
72          }finally{
73              try{
74                  if(rs!=null)  rs.close();
75                  if(pstmt!=null)  pstmt.close();
76                  if(con!=null)  con.close();
77              }catch(SQLException se){
78                  System.out.println(se.getMessage());
79              }
80          }
81      %>
82  </table>
83  </body>
84  </html>
```

영화에 대해 필요한 정보인 영화이름, 개봉일, 일련번호, 주연배우수, 제작자 수, 감독 수, 배우 주민번호, 감독 주민번호에 대한 리스트를 만든 다음 이것들을 수정 삭제할 수 있도록 창을 넣어주었다. 수정은 projectupdate.jsp에서 수행하고 삭제는 projectdelete.jsp에서 수행하도록 한다.



이클립스에서 실행하면 내가 이전에 mysql에서 만들었던 movie데이터베이스의 튜플들이 리스트로 나오게 된다.

<projectupdate.jsp>

projectinse... | ProjectConn... | projectindex... | projectupda... | projectupda... | Insert titl... | projectinse...

```jsp
<%@page import="java.sql.SQLException"%>
<%@page import="java.sql.Timestamp"%>
<%@page import="java.sql.ResultSet"%>
<%@page import="examples.ProjectConnection"%>
<%@page import="java.sql.PreparedStatement"%>
<%@page import="java.sql.Connection"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>

</head>
<body>
<%
    request.setCharacterEncoding("utf-8");
    String mname=request.getParameter("mname");

    Connection con=null;
    PreparedStatement pstmt=null;
    ResultSet rs = null;
    try{
        con = ProjectConnection.getCon();
        String sql="select * from movie where mname=?";

        pstmt = con.prepareStatement(sql);
                pstmt.setString(1, mname);
        rs = pstmt.executeQuery();

        //String mssn=rs.getString("mssn");
        String date=rs.getString("date");
```

Console

```jsp
        <td colspan="2" align="center">
        <input type="button" name="btn1" value="저장" onclick="javascript:frm.submit();"/>
        <input type="button" name="btn2" value="목록" onclick="javascript:location.href='projectlist.jsp';"/>
        </td>
    </tr>
</table>
</form>
<%
    }catch(SQLException se){
        System.out.println(se.getMessage());
    }finally{
        try{
            if(rs!=null)  rs.close();
            if(pstmt!=null)  pstmt.close();
            if(con!=null)  con.close();
        }catch(SQLException se){
            System.out.println(se.getMessage());
        }
    }

%>
<script type="text/javascript">
    function update(){
        document.frm.submit();
    }
    function list(){
        location.href="projectlist.jsp";
    }
</script>
</body>
</html>
```

영화 데이터베이스를 수정하도록 하는 jsp를 만들었다. 영화 릴레이션의 개봉일을 변경할 수 있게 해주었으며 수정 후 저장버튼과 목록버튼을 만들어 저장을 누르면 frm.submit()으로 제출이 되며 목록을 누르면 projectlist.jsp를 다시 수행할 수 있도록 한다.

<projectupdateSucess.jsp>

```jsp
<%@page import="java.sql.SQLException"%>
<%@page import="examples.ProjectConnection"%>
<%@page import="java.sql.PreparedStatement"%>
<%@page import="java.sql.Connection"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
request.setCharacterEncoding("utf-8");
String mname=request.getParameter("mname");
//String mssn=request.getParameter("mssn");
String date=request.getParameter("date");
/* int lead_actor_num=Integer.parseInt(request.getParameter("lead_actor_num"));
int pro_num=Integer.parseInt(request.getParameter("pro_num"));
int direct_num=Integer.parseInt(request.getParameter("direct_num"));
//String lead_actor_num=request.getParameter("lead_actor_num");
//String pro_num=request.getParameter("pro_num");
//String direct_num=request.getParameter("direct_num");
String actor_assn=request.getParameter("actor_assn");
String director_dssn=request.getParameter("director_dssn");
 */
   //db에 저장하기
   Connection con = null;
   PreparedStatement pstmt = null;
   String sql = "update mname set date=? where mname=?" ;
   //"update mname set mssn=?,date=? , lead_actor_num=?, pro_num=?, direct_num=?, actor_assn=?, director_dssn=? where mname= ?";
   int n=0;

    try{
        con = ProjectConnection.getCon();
        pstmt = con.prepareStatement(sql);
        pstmt.setString(1, mname);
        //pstmt.setString(2, mssn);
        pstmt.setString(3, date);

        n = pstmt.executeUpdate();
    }catch(SQLException se){
        System.out.println(se.getMessage());
    }finally{
        try{
            if(pstmt!=null) pstmt.close();
            if(con!=null) con.close();
        }catch(SQLException se){
            System.out.println(se.getMessage());
        }
    }
%>
```

```
58
59 <script type="text/javascript">
60     if(<%=n%>>0 ){
61         alert("수정완료");
62         location.href="projectlist.jsp";
63     }else{
64         alert("수정 실패");
65         history.go(-1);
66     }
67 </script>
68 </body>
69 </html>
70
```

영화 데이터베이스의 개봉일을 성공적으로 수정하였으면 수정완료란 문구가 뜨며 수정이 잘못되면 수정실패와 함께 history.go(-1);를 사용하여 이전 페이지로 돌아갈 수 있도록 하였다.

<projectdelete.jsp>

```
1  <%@page import="examples.ProjectConnection"%>
2  <%@page import="java.sql.SQLException"%>
3  <%@page import="java.sql.PreparedStatement"%>
4  <%@page import="java.sql.Connection"%>
5  <%@ page language="java" contentType="text/html; charset=UTF-8"
6      pageEncoding="UTF-8"%>
7  <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
8  <html>
9  <head>
10 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
11 <title>Insert title here</title>
12 </head>
13 <body>
14 <%
15     //삭제할 아이디
16     Connection con = null;
17     PreparedStatement pstmt = null;
18
19     String mname = request.getParameter("mname");
20     int n=0;
21     try{
22         con = ProjectConnection.getCon();
23         String sql="delete from movie where mname=?";
24         pstmt = con.prepareStatement(sql);
25         pstmt.setString(1, mname);
26
27         n = pstmt.executeUpdate();
28     }catch(SQLException se){
29         System.out.println(se.getMessage());
30     }finally{
31         if(pstmt!=null) pstmt.close();
32         if(con!=null) con.close();
33     }
34     response.sendRedirect("projectlist.jsp");
35 %>
36 </body>
37 </html>
```

마지막으로 영화 데이터베이스를 삭제해주는 jsp를 만들었다.

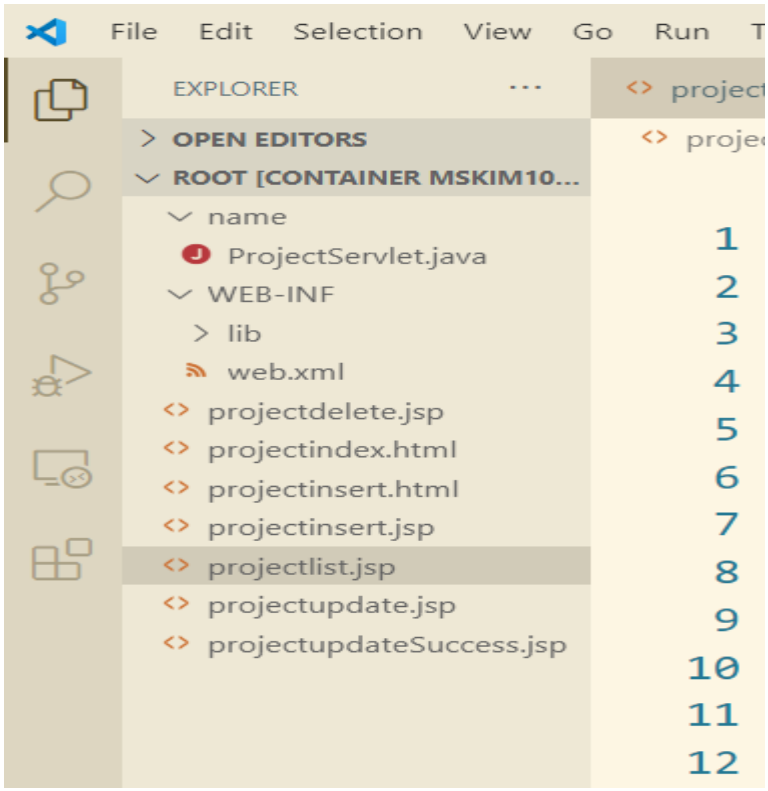
3. 실행 화면

마지막으로 도커에 내가 만든 이미지를 연동시켜주기 위해 먼저 다음과 같이 아이피 주소를 보았다.

docker inspect mysql 로 아이피 주소 보기

비주얼 스튜디오 코드로 들어가 f1키를 누른 후 attach to running container 선택 후 html-servlet을 연동하는 작업을 거쳐주었다.

```
 * Starting Apache httpd web server apache2
 *
Using CATALINA_BASE:   /usr/local/tomcat8
Using CATALINA_HOME:   /usr/local/tomcat8
Using CATALINA_TMPDIR: /usr/local/tomcat8/temp
Using JRE_HOME:        /usr/local/java
Using CLASSPATH:       /usr/local/tomcat8/bin/bootstrap.jar:/usr/local/tomcat8/b
in/tomcat-juli.jar
Using CATALINA_OPTS:
Tomcat started.
root@8a2ec6384495:/usr/local/tomcat8/webapps/ROOT# javac ProjectServlet.java -d
../WEB-INF/classes/
javac: file not found: ProjectServlet.java
Usage: javac <options> <source files>
use -help for a list of possible options
root@8a2ec6384495:/usr/local/tomcat8/webapps/ROOT# service apache2 restart
 * Restarting Apache httpd web server apache2                             [ OK ]
root@8a2ec6384495:/usr/local/tomcat8/webapps/ROOT# service tomcat8 restart
Using CATALINA_BASE:   /usr/local/tomcat8
Using CATALINA_HOME:   /usr/local/tomcat8
Using CATALINA_TMPDIR: /usr/local/tomcat8/temp
Using JRE_HOME:        /usr/local/java
Using CLASSPATH:       /usr/local/tomcat8/bin/bootstrap.jar:/usr/local/tomcat8/b
in/tomcat-juli.jar
```



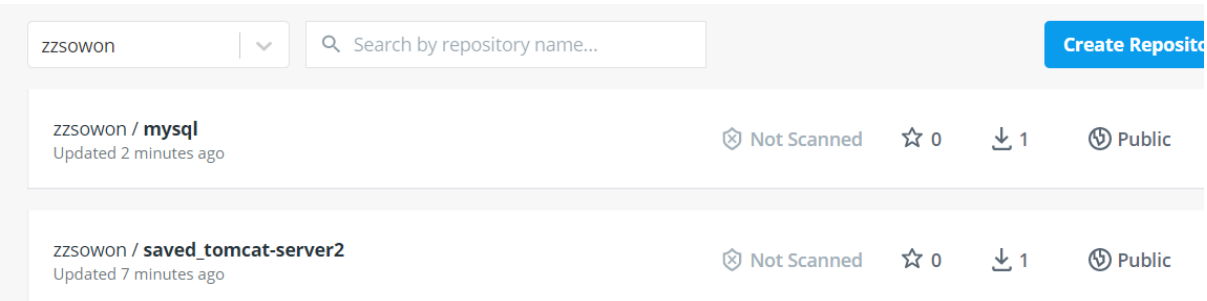
이클립스에서 만들었던 jsp, html소스들을 옮겨주었다. (이클립스의 projectConnection이 여기선 projectServlet.Java로 이름을 지정)

```
C:₩Users₩zzsow>docker commit tomcat-server2 saved_tomcat-server2
sha256:23360bc1432ee8a5f5f6769d21fc0965bc8bcbfaa69881cd6654425f26958a00

C:₩Users₩zzsow>docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
saved_tomcat-server2        latest              23360bc1432e        7 seconds ago       875MB
saved_mysql                 latest              6a9cebe584df        16 hours ago        545MB
zzsowon/pj                  latest              6a9cebe584df        16 hours ago        545MB
saved_tomcat-server         latest              a9d579e0fb73        3 weeks ago         981MB
zzsowon/docker_test_image   latest              a9d579e0fb73        3 weeks ago         981MB
lect11_mskim                latest              5a3597a1b836        4 weeks ago         878MB
lect10                      latest              f2ca062b3114        4 weeks ago         545MB
mskim1024/docker_db_test    latest              cf101b30a306        4 weeks ago         770MB
mysql                       latest              db2b37ec6181        7 weeks ago         545MB

C:₩Users₩zzsow>docker tag saved_tomcat-server2 zzsowon/saved_tomcat-server2

C:₩Users₩zzsow>docker images
REPOSITORY                  TAG                 IMAGE ID            CREATED             SIZE
saved_tomcat-server2        latest              23360bc1432e        2 minutes ago       875MB
zzsowon/saved_tomcat-server2   latest           23360bc1432e        2 minutes ago       875MB
saved_mysql                 latest              6a9cebe584df        16 hours ago        545MB
zzsowon/pj                  latest              6a9cebe584df        16 hours ago        545MB
saved_tomcat-server         latest              a9d579e0fb73        3 weeks ago         981MB
zzsowon/docker_test_image   latest              a9d579e0fb73        3 weeks ago         981MB
lect11_mskim                latest              5a3597a1b836        4 weeks ago         878MB
lect10                      latest              f2ca062b3114        4 weeks ago         545MB
mskim1024/docker_db_test    latest              cf101b30a306        4 weeks ago         770MB
mysql                       latest              db2b37ec6181        7 weeks ago         545MB

C:₩Users₩zzsow>docker push zzsowon/saved_tomcat-server2
The push refers to repository [docker.io/zzsowon/saved_tomcat-server2]
4945769d6f9b: Pushed
e4edb7863185: Mounted from zzsowon/docker_test_image
aa794496f8e9: Mounted from zzsowon/docker_test_image
7a694df0ad6c: Mounted from zzsowon/docker_test_image
3fd9df553184: Mounted from zzsowon/docker_test_image
805802706667: Mounted from zzsowon/docker_test_image
latest: digest: sha256:9bd83e72725f42ce51f64c46af6d98afcb161f3f88b491ccb87f33288dab7cc2 size: 1580

C:₩Users₩zzsow>
```

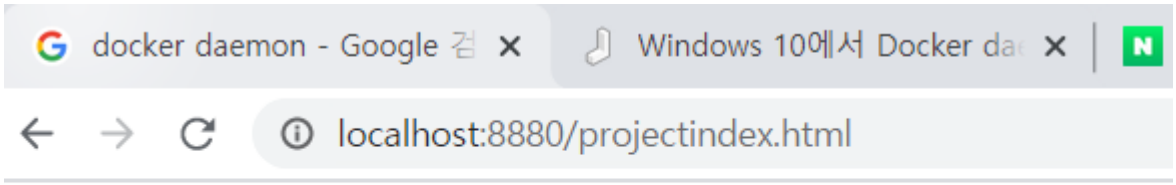
먼저 톰캣서버 이미지를 저장해주었다. 서버이미지이름: saved_tomcat-server2

```
C:₩Users₩zzsow>docker tag saved_mysql zzsowon/mysql

C:₩Users₩zzsow>docker images
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
saved_tomcat-server2          latest              23360bc1432e        9 minutes ago       875MB
zzsowon/saved_tomcat-server2  latest              23360bc1432e        9 minutes ago       875MB
zzsowon/pj                    latest              6a9cebe584df        16 hours ago        545MB
saved_mysql                   latest              6a9cebe584df        16 hours ago        545MB
zzsowon/mysql                 latest              6a9cebe584df        16 hours ago        545MB
saved_tomcat-server           latest              a9d579e0fb73        3 weeks ago         981MB
zzsowon/docker_test_image     latest              a9d579e0fb73        3 weeks ago         981MB
lect11_mskim                  latest              5a3597a1b836        4 weeks ago         878MB
lect10                        latest              f2ca062b3114        4 weeks ago         545MB
mskim1024/docker_db_test      latest              cf101b30a306        4 weeks ago         770MB
mysql                         latest              db2b37ec6181        7 weeks ago         545MB

C:₩Users₩zzsow>docker push zzsowon/mysql
The push refers to repository [docker.io/zzsowon/mysql]
1d27588c424b: Mounted from zzsowon/pj
9b0377a95c0e: Mounted from zzsowon/pj
af6e790b8237: Mounted from zzsowon/pj
060fef62a228: Mounted from zzsowon/pj
7f893b7c04ac: Mounted from zzsowon/pj
7832ac00d41e: Mounted from zzsowon/pj
15b463db445c: Mounted from zzsowon/pj
c21e35e55228: Mounted from zzsowon/pj
36b89ee4c647: Mounted from zzsowon/pj
9dae2565e824: Mounted from zzsowon/pj
ec8c80284c72: Mounted from zzsowon/pj
329fe06a30f0: Mounted from zzsowon/pj
d0fe97fa8b8c: Mounted from zzsowon/pj
latest: digest: sha256:0e6e9ff3403755261b150caa2f3b082b5e9a5eefe6436bf6bdd86c7c28c1e7d0 size: 3036

C:₩Users₩zzsow>
```

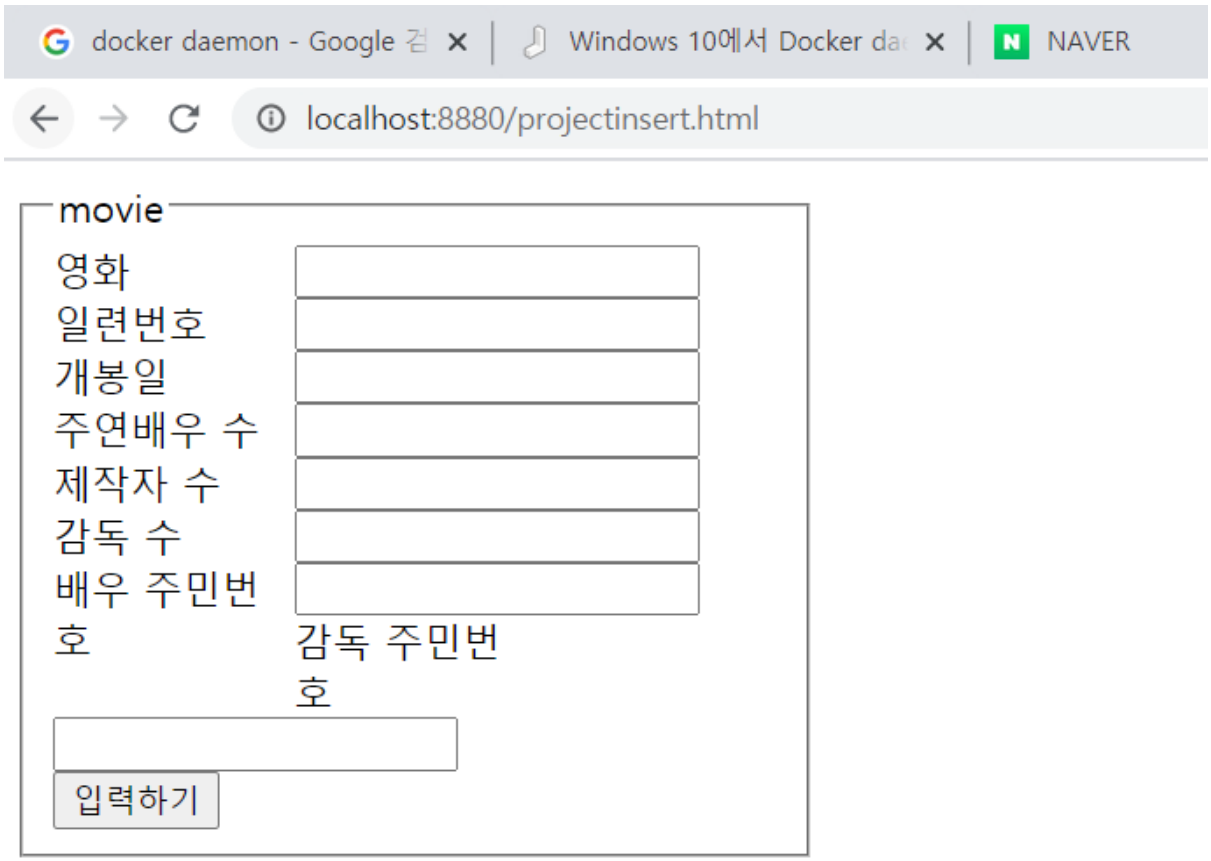
mysql이미지를 (이름:mysql) 도커허브에 저장해주었다.



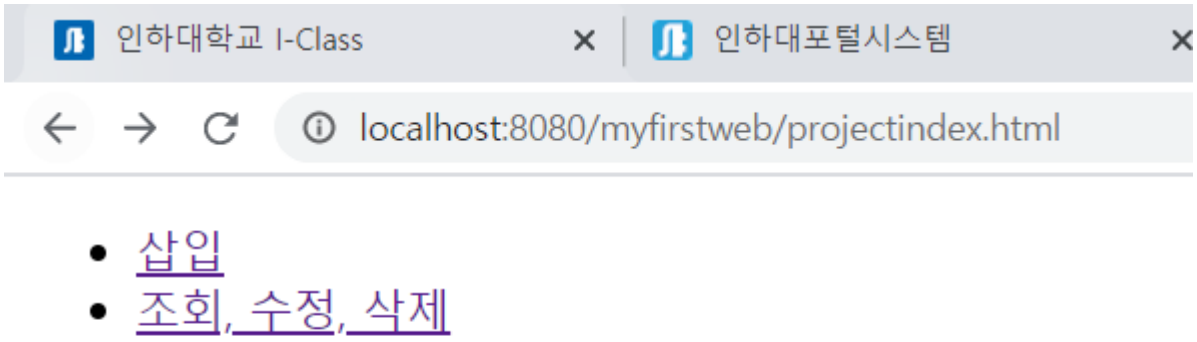
도커 허브에서 확인을 하니 mysql과 톰캣서버 두개가 올라가있는 것을 확인할 수 있었다.



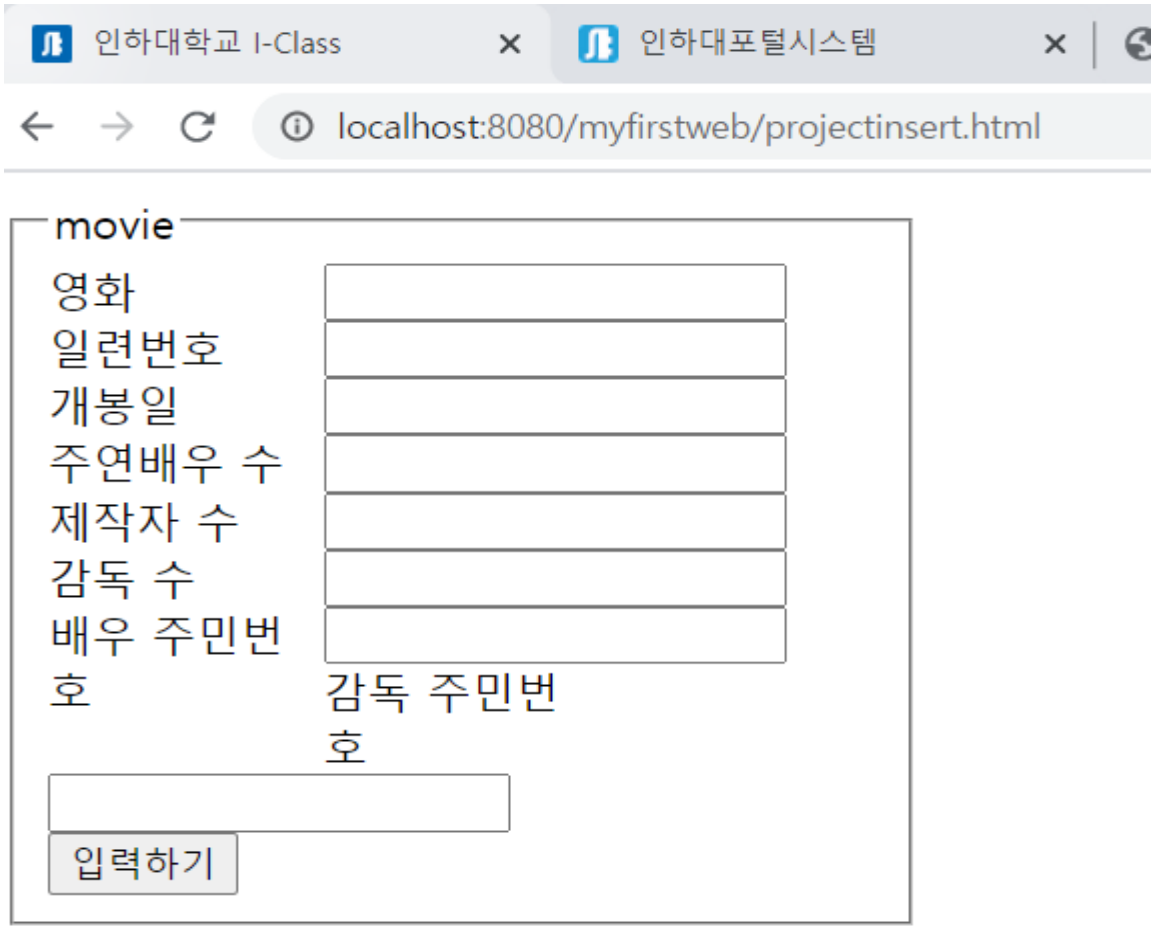
- 삽입
- 조회, 수정, 삭제



위와 같이 도커에 올린 이미지를 실행하니 다음과 같이 나오는 걸 확인 할 수 있었다.(도커에서 올린 것이 잘 안되는 것 같아 자바 이클립스에서 다시 수행해보았다.)

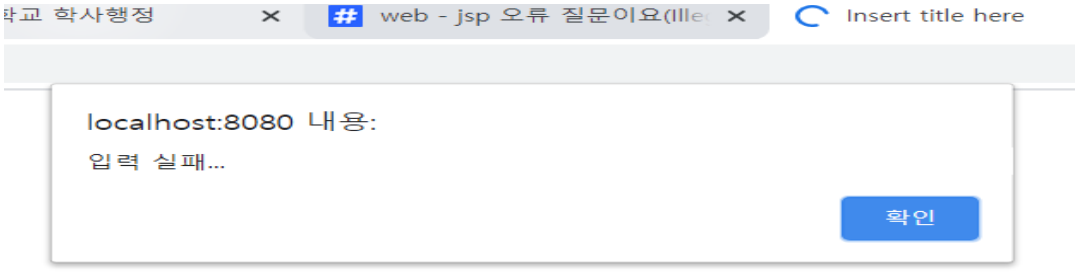


크롬에 접속하여 실행해보았다. 먼저 위와 같이 삽입, 조회, 삭제를 선택하는 버튼 하나, 조회를 선택하는 버튼 두개가 있다. 삽입, 조회, 삭제를 누르면

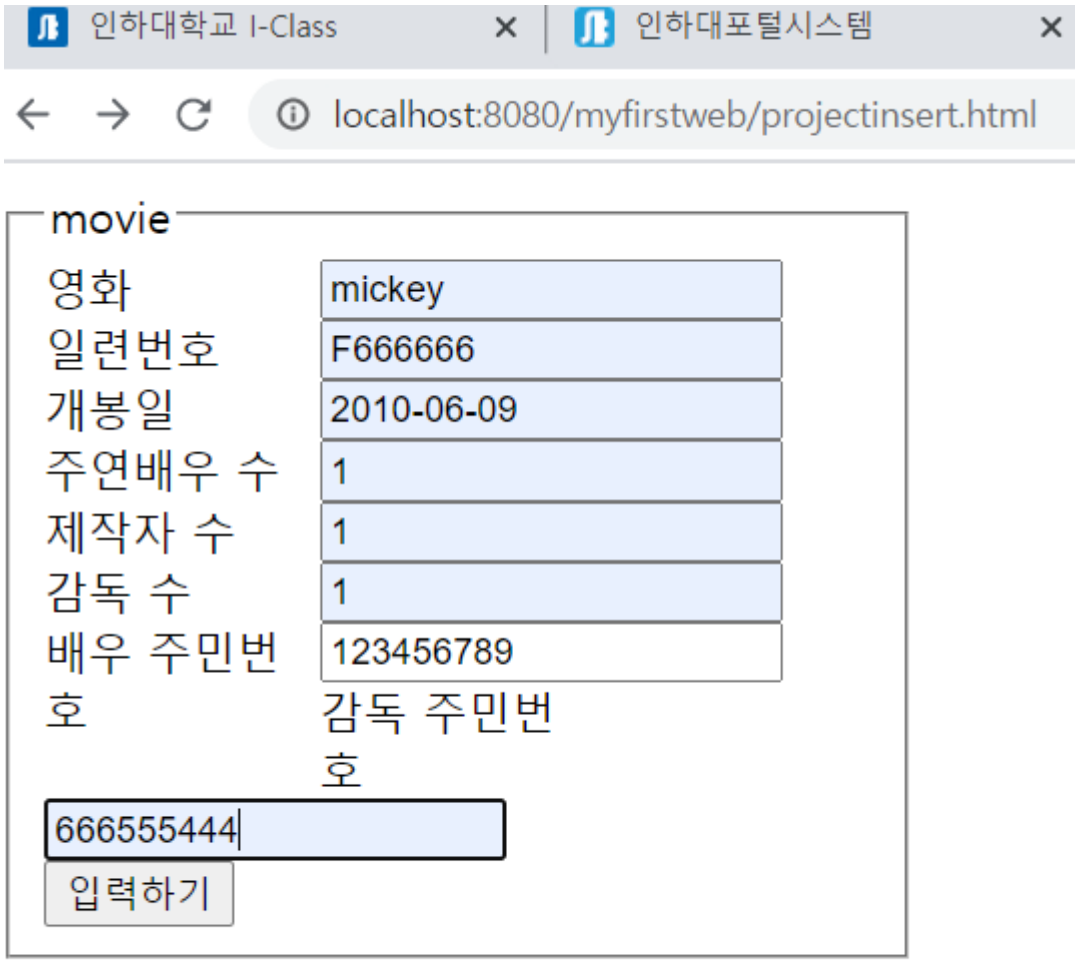


조회를 선택하면 바로 아래와 같이 연결되며 리스트들을 볼 수 있다.

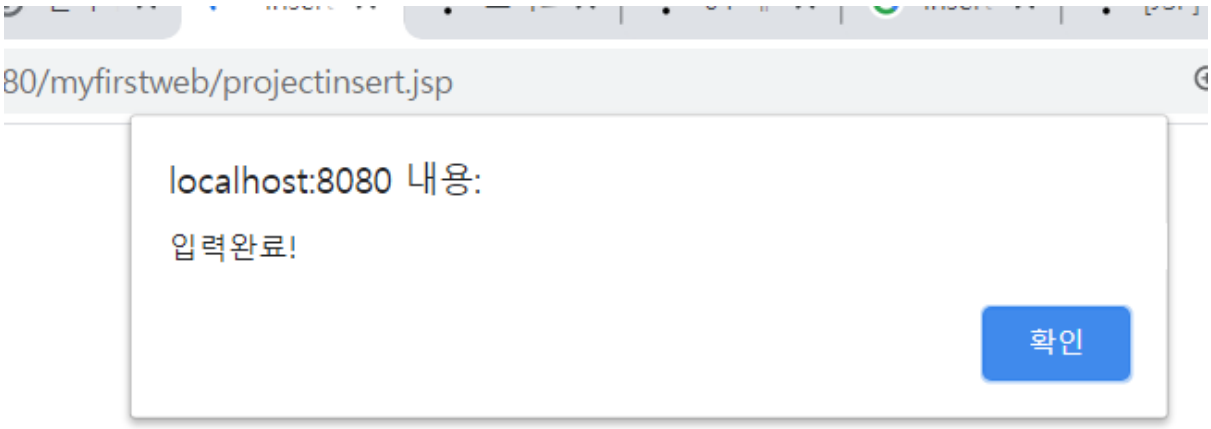
먼저 아무것도 입력을 안한 채 입력하기 버튼을 눌러보았다.



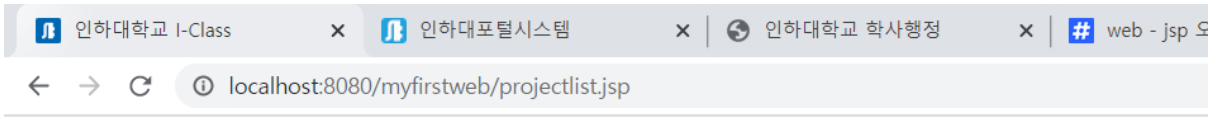
아무것도 입력을 안한채 입력하기를 누르면 위와 같이 입력 실패...라고 뜨는 것을 확인할 수 있었다.



위와 같이 입력 후 입력하기 버튼을 눌러주었다.



성공적으로 입력된 것을 알 수 있다.



movie

영화	일련번호	개봉일	주연배우 수	제작자 수	감독 수	배우 주민 번호	감독 주민 번호	수정	삭제
coco	A111111	2017-11-22	2	1	1	112233445	999888777	수정	삭제
coco	A111112	2017-11-22	2	1	1	123456789	999888777	수정	삭제
car	B222222	2004-05-25	1	1	1	987654321	666555444	수정	삭제
lion king	C333333	2011-09-03	1	1	1	998877665	333222111	수정	삭제
Dumbo	D444444	2007-10-16	1	1	1	121234343	333222111	수정	삭제
toy story	E555555	2013-08-13	1	1	1	987654321	121234343	수정	삭제

```
mysql> select * from movie;
+-----------+---------+------------+----------------+---------+------------+------------+--------------+
| Mname     | Mssn    | Date       | Lead_actor_num | Pro_num | Direct_num | Actor_Assn | Director_Dssn |
+-----------+---------+------------+----------------+---------+------------+------------+--------------+
| coco      | A111111 | 2017-11-22 |              2 |       1 |          1 | 112233445  | 999888777    |
| coco      | A111112 | 2017-11-22 |              2 |       1 |          1 | 123456789  | 999888777    |
| car       | B222222 | 2004-05-25 |              1 |       1 |          1 | 987654321  | 666555444    |
| lion king | C333333 | 2011-09-03 |              1 |       1 |          1 | 998877665  | 333222111    |
| Dumbo     | D444444 | 2007-10-16 |              1 |       1 |          1 | 121234343  | 333222111    |
| toy story | E555555 | 2013-08-13 |              1 |       1 |          1 | 987654321  | 121234343    |
+-----------+---------+------------+----------------+---------+------------+------------+--------------+
6 rows in set (0.00 sec)
```

movie리스트를 조회한 결과는 mysql문과 비교하였을 때 동일한 값들을 출력하는 것을 알 수 있다.

4. 고찰

디비 설계 프로젝트를 통해 jsp, html소스를 만드는 것부터 er 다이어그램, 마지막으로 도커까지 자세히 알아가는 경험이 되었다. 먼저 처음 erd를 만드는 과정에서 pk와 fkfmf 어떤걸로 할지가 가장 고민이 되었는데 pk와 fk를 잘 설정하여 중간에 오류가 생기지 않도록 해야겠다 느꼈다. 한번 잘못설정하니 다시 돌리는데 매우 많은 시간이 걸렸기 때문이다. 또한 insert 문을 만드는 데 있어서 어느 테이블을 먼저 insert할지도 매우 중요하다는 것을 깨달았다. 어떤 테이블이 참조하는지, 어떤 테이블은 참조 받는지를 잘 파악해 참조하고 있는 테이블을 먼저 insert문을 다 만들어주어야 오류가 나오지 않음을 알 수 있었다. 자바 이클립스에서 jsp, html소스들을 만드는데 mysql과의 연결도 은근 어려움이 있었다. 또한 get인지 string인지 함수 타입은 어떤지를 잘 파악해야됨을 느꼈다. 조인문도 무엇과 무엇을 연결시켜야 되는지 확인하며 마지막으로 도커를 연결하는데 나는 가장 큰 어려움이 걸렸다. 요번 설계 프로젝트를 통해 직접 데이터베이스를 구현하고 web을 응용해보는 것을 통해 나중에 큰 도움이 될 것이라 생각한다.