INHA UNIVERSITY

데이터베이스 설계
Database Design

Final Project

Movies Database Design & Web Application

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 3 일

학부 정보통신공학과
학년 3
성명 류덕형
학번 12161725

개요 4

상세 설계내용 5

Database 설계 5

주어진 ERD에 대한 True or False 문항 대답 : 5

수정한 ERD 6

MySQL Workbench로 작성한 EER Diagram : 7

Database 생성 Script (Index 포함) 7

기본적인 Value들 Insert 8

View Table 생성 8

Web 설계 8

HTML 설계 8

DTO&DAO, Servlet 설계 8

과제 실행화면 10

Docker Upload 18

고찰 18

Appendix 19

Movies Database 생성 스크립트 (Index 포함) 19

기본 Value Insert 문 22

View Table Create문 24

Create Actor html 25

Create Producer html 25

Create Movie html 26

Read Movie html 27

Read Actors html 27

Read Producer html 28

Read Director&Actor html 28

Read Producer&Actor html 28

Update Director&Actor html 29

Update Producer&Actor html 29

Delete Actor html 30

Delete Producer html 30

Index html 31

DTO java 32

DAO java 35

Servlet java 48

1. 개요

MOVIES ERD를 기반으로 MOVIES 데이터베이스를 설계하고 데이터베이스에 CRUD를 수행할 수 있는 web application을 작성한다.

첨부파일 :

Final_ERD.mwb (Workbench 작업모습)

Create_Table.sql (Database 생성 스크립트(index생성 포함))

Insert.sql (처음에 기본적인 값들 Insert 했던 스크립트)

View_Create (View Table 생성 스크립트)

html파일 13개 (C 3개 R 5개 U 2개 D 2개 index 1개)

moviesDTO.java

moviesDAO.java

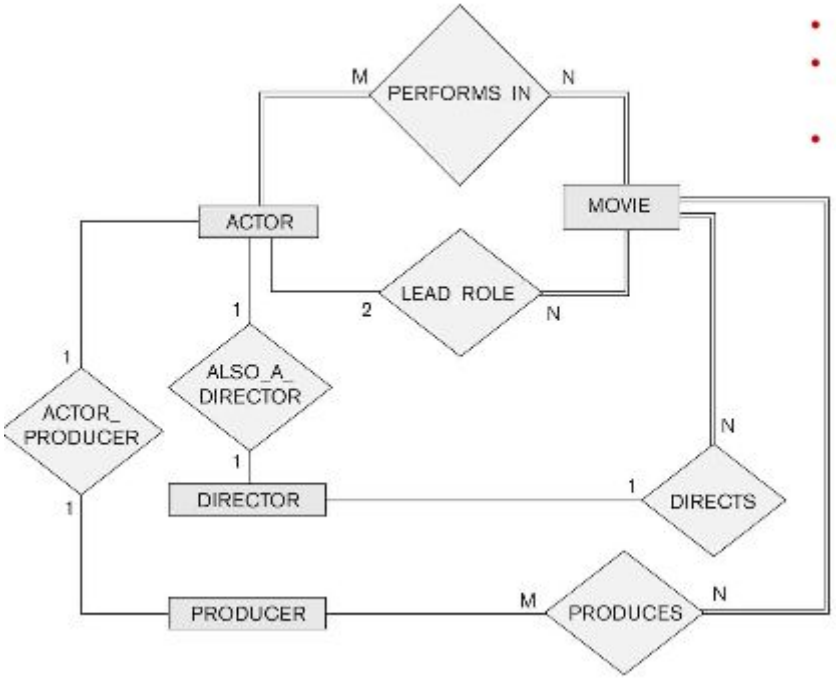
movie_servlet.java

이 보고서에 올라간 code source와 첨부된 파일은 모두 docker mysql을 사용하고 내 컴퓨터의 로컬 tomcat-server를 사용했을 때의 코드임. docker tomcat-server를 사용할 때는 조금의 수정이 있었는데 이는 Docker Upload부분에서 설명함.

2. 상세 설계내용

Database 설계

주어진 ERD에 대한 True or False 문항 대답 :



(1) There are no actors in this database that have been in no movies. True. No Actors without movies.

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

(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. No more or less. Just TWO.

(5) Every director has been an actor in some movie. False. No need to.

(6) No producer has ever been an actor. False. It's possible.

(7) A producer cannot be an actor in some other movie. False. They can be.

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

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

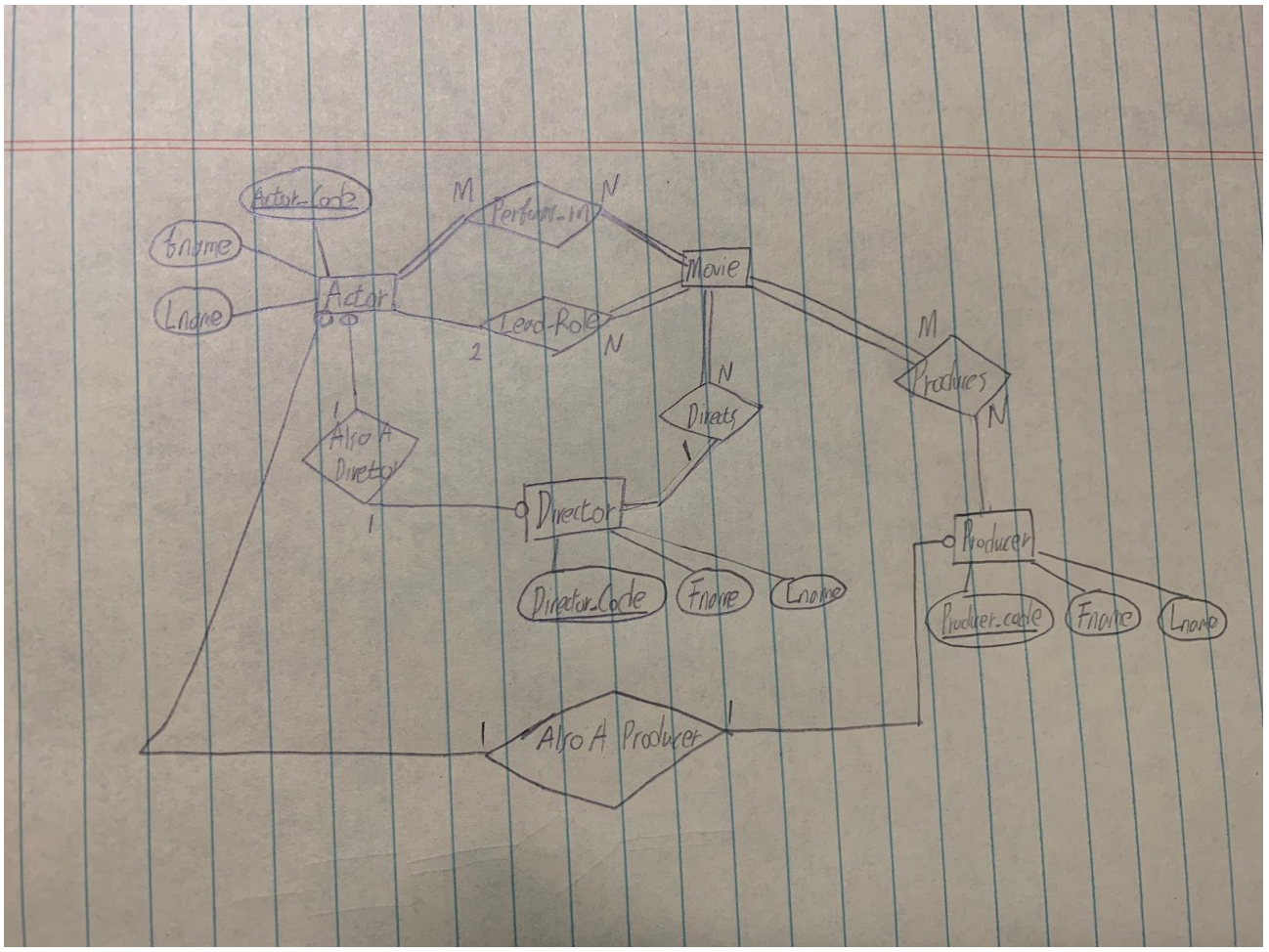
(10) Most movies have one director and one producer. Maybe. But I'll make it Ture.

(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

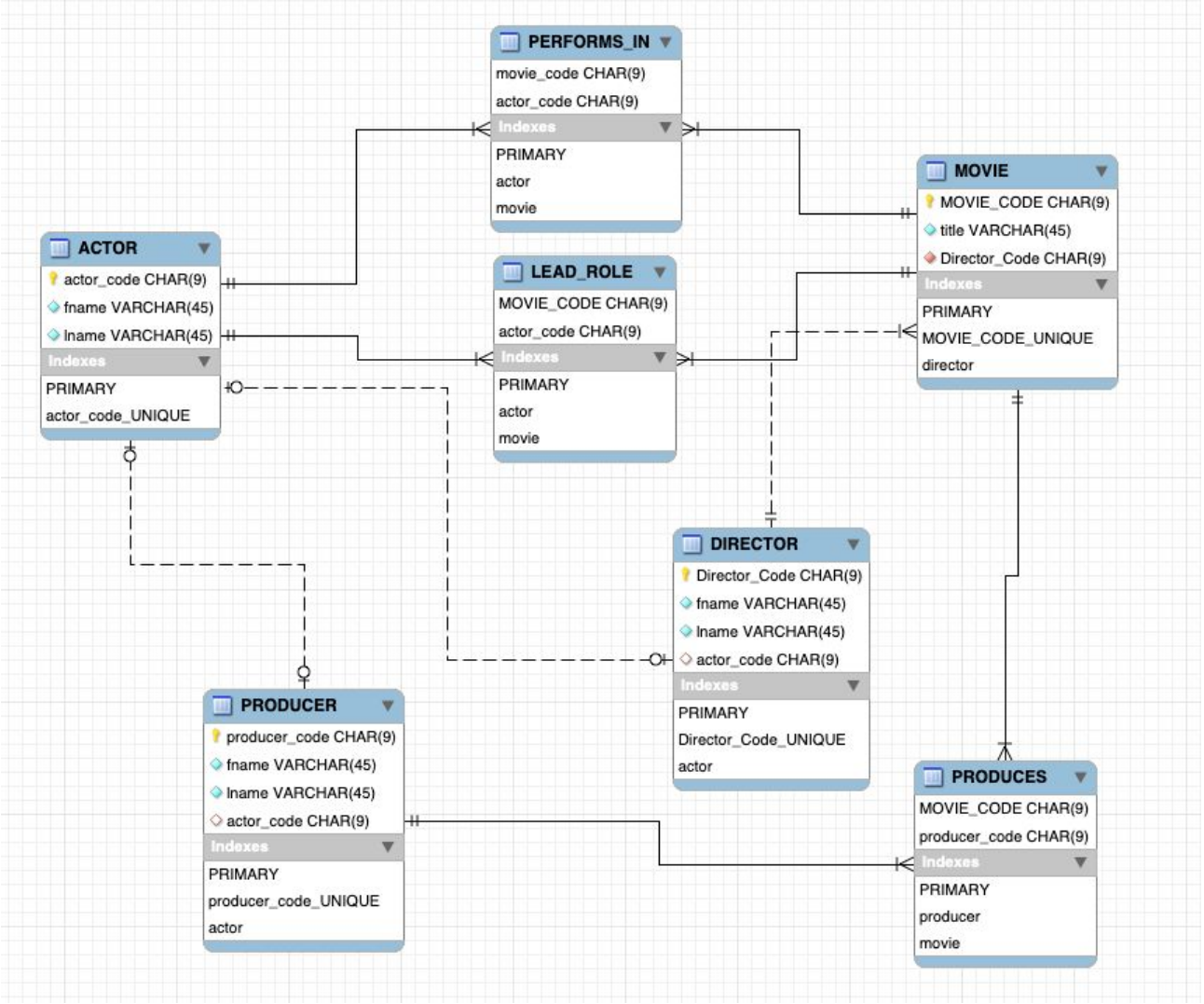
수정한 ERD



혹시라도 동명이인이 있을 수 있다고 생각해 각자 code를 만들어 이것을 pk로 지정해주었다.

True or False 문항 대답에 부합하도록 했고, 여기에서 답했듯이 Lead_Role은 더도 덜도 아닌 무조건 2명이여야 하는걸로 받아들여 설계한다.

MySQL Workbench로 작성한 EER Diagram :



N:M관계는 관계를 담당하는 테이블을 만들어 구현했는데 이 데이터베이스에 식별관계는 없지만 관계에 해당하는 테이블(Lead_Role, Performs_In, Produces)은 양쪽 엔티티의 정보없이는 자기자신을 식별할 수 없는 테이블이 되므로 관계테이블은 실선처리하였다. 또한 관계테이블은 한쪽이라도 지워지면 의미없는 정보가 되므로 foreign key설정에서 on delete cascade를 추가해 한쪽 정보가 지워지면 연쇄적으로 같이 지워지도록 했다. 그리고 배우가 꼭 Producer나 Director를 해야할 이유는 없으므로 해당 Foreign Key의 Not null과 mandatory를 해제했다.

Database 생성 Script (Index 포함)

Workbench의 Database Forward Engineering을 이용해 생성했다.

Index는 Workbench에서 Database를 만들 때 같이 만들었으며 각 테이블별 PK와 주요 Foreign key들에 대해 index를 설정했다.

코드의 길이가 너무 길어 Appendix를 이용한다.

기본적인 Value들 Insert

참조관계에 주의하며 Insert 순서를 신경써서 넣었고 이 또한 코드가 길어 Appendix를 이용한다.

View Table 생성

영화제목, 감독, 주연배우 2명을 보여주는 MOVIE_INFO,

영화이름과 모든 출연배우를 보여주는 ACTORS_IN_MOVIE,

영화이름과 참여 프로듀서를 보여주는 PRODUCERS_IN_MOVIE,

감독겸 배우나 프로듀서 겸 배우를 보여주는 ACTOR_DIRECTOR, ACTOR_PRODUCER

이렇게 총 5개의 View Table을 생성했다. 코드는 Appendix에 첨부한다.

Web 설계

ERD에 ACTOR와 MOVIE의 LEAD_ROLE관계는 2:N인데 이는 데이터베이스에서 구현이 힘들어 Web에서 강제 시켰다.

HTML 설계

CRUD의 기능들을 하는 html파일들을 다 따로 만든 후(C 3개, R 5개, U 2개, D 2개 총 12개) index.html 하나를 만들어 여기서 해당파일들로 링크를 걸어서 모든 기능들을 이용할 수 있게 하였다. get method를 이용해 값들을 주고받았으며 기능수행에 필요한 값들과 추가적으로 mode라는 hidden value를 넘겨서 servlet에서 어떤 기능을 수행해야하는지 알 수 있게 했다. 꼭 있어야만 하는 정보들은 required 태그를 사용해 입력하지 않으면 넘어가지 않게 했다.

Source Code : Create(Actor, Producer, movie) Read(Movie, Actors, Producers, Director&Actor, Producer&Actor) Update(Director&Actor, Producer&Actor) Delete(Actor,

Producer) Index

DTO&DAO, Servlet 설계

HTML에서 버튼을 누르면 movie_servlet이라는 파일에서 실행을 하게 되는데 여기서는 mode를 switch문으로 나눠서 원하는 기능별로 DAO의 함수들을 사용해 실행하고 실행이 완료되었다, 혹은 에러가 났다 등의 html 메세지를 띄우게 했다.

DTO는 데이터베이스의 모든 value들을 다 커버하게끔 value들을 두었고 eclipse의 source자동생성을 통해 생성자, get, set함수들을 생성했다.

DAO는 데이터베이스와 직접적으로 연동해 Query문을 사용해 CRUD의 기능들을 하는 각각의 함수들을 구현했다. 이 기능들에 대해 설명하면

Create 관련 함수에서는 actor, producer추가에서는 movie에 전체 참여인것을 고려하여 무조건 movie 정보도 받게하고 (html에서 require걸어서) 이 영화가 데이터베이스에 있는지 체크하고 있으면 값을 넣고 없으면 에러메시지를 띄었다. 주연배우는 무조건 2명이여야 하므로 조연배우만 추가할 수 있다. 프로듀서를 추가할때 이사람이 등록된 배우이기도 하면 ActorCode도 선택적으로 같이 넣을 수 있다. 그리고 영화 전체를 추가하는것도 만들었는데 여기서는 처음에 데이터베이스에 값을 집어넣을 때 처럼 순서에 신경쓰며 sql문을 사용했다.

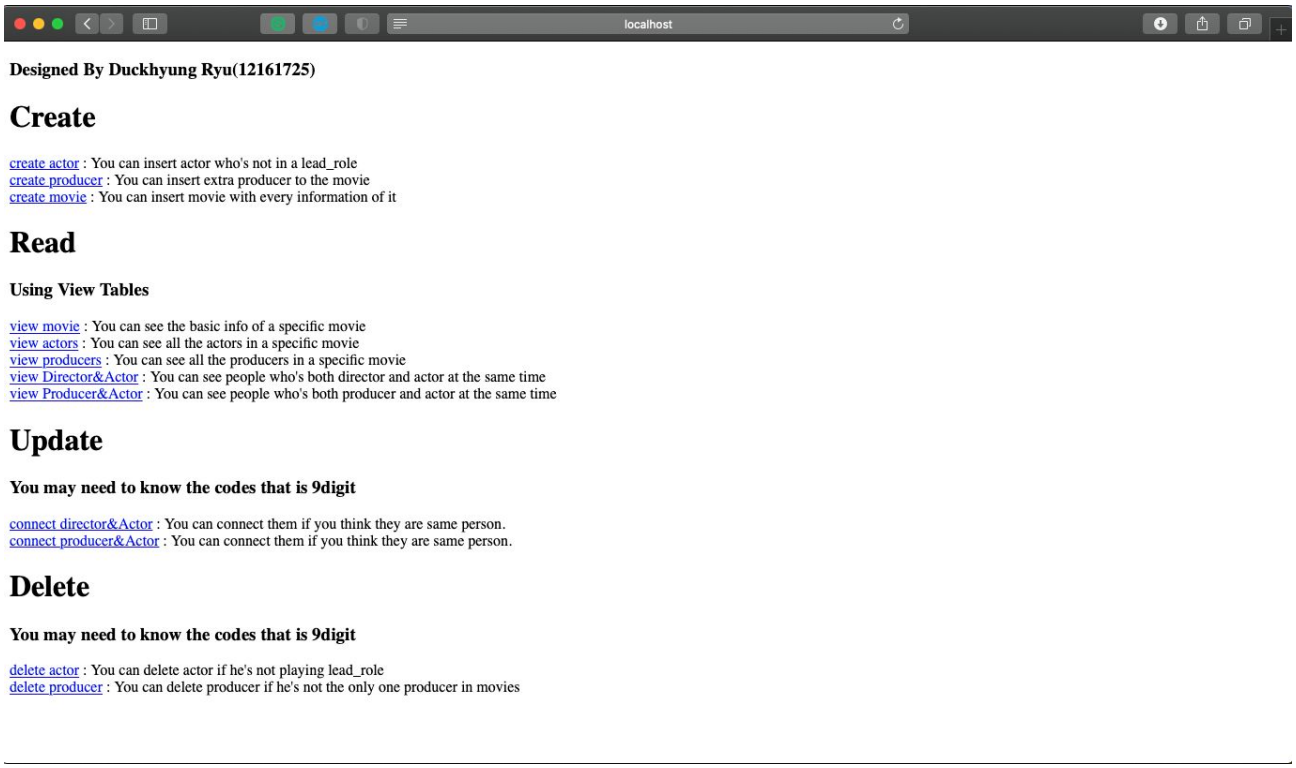
Read 관련 함수에서는 모두 View Table을 이용했으며 영화 이름을 입력받고 이를 View Table에서 검색하거나 그냥 모든 list를 보여주기도 했다.

Update 관련 함수에서는 각 코드가 제대로된 코드가 아니라면 에러를 보내고 정상이라면 수행이 되도록 했다.

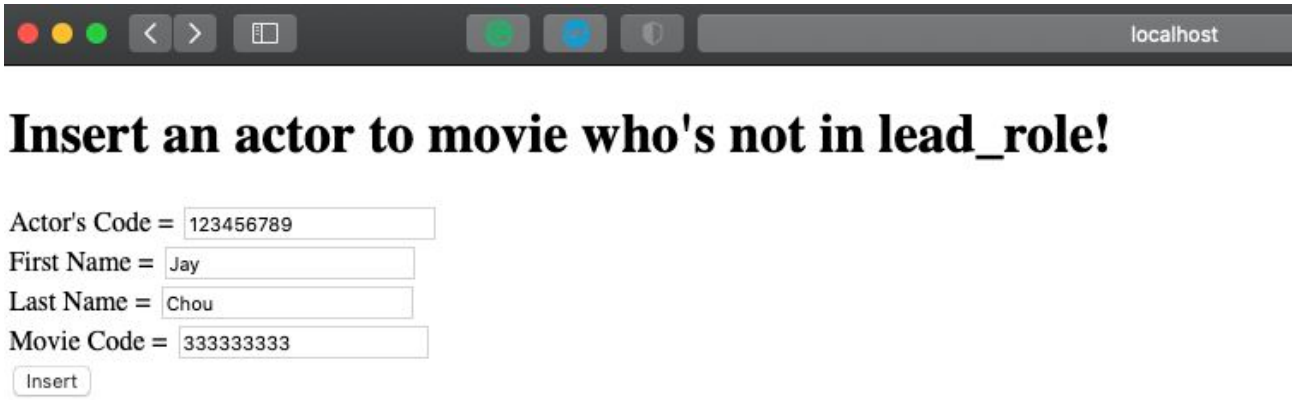
Delete 관련 함수에서는 배우삭제시에는 주연배우는 항상 2명이여야 하므로 주연배우 리스트에있지 않은 사람인지 검색 후 아닐경우에만 삭제를 수행하고 producer삭제시에는 영화당 프로듀서가 꼭 한명이상 있어야 하므로 해당프로듀서가 참여한 영화들에 이 사람이 유일한 사람인지 검색해 아닐 경우에만 삭제가 진행되도록 했다.

Source Code : DTO, DAO, Servlet

3. 과제 실행화면



기본 화면 (index.html)모습. 위에서부터 하나씩 해볼것임.

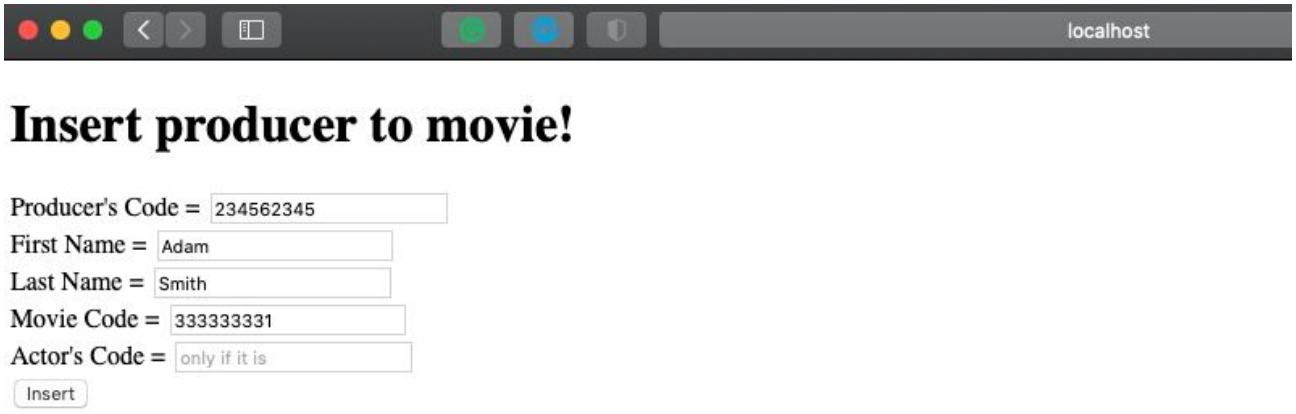


Create Actor 선택하고 값 입력함.

```
mysql> select * from ACTOR;
+------------+---------+-------------+
| actor_code | fname   | lname       |
+------------+---------+-------------+
| 111111111  | Brad    | Pitt        |
| 111111112  | George  | Clooney     |
| 111111113  | Matt    | Damon       |
| 111111114  | Matthew | McConaughey |
| 111111115  | Tom     | Cruise      |
| 111111116  | Woddy   | Allen       |
| 123456789  | Jay     | Chou        |
+------------+---------+-------------+
```

```
mysql> select * from PERFORMS_IN where ACTOR_CODE='123456789';
+------------+------------+
| movie_code | actor_code |
+------------+------------+
| 333333333  | 123456789  |
+------------+------------+
```

입력완료 메세지, 데이터베이스(Actor, Performs_In테이블)에도 정상적으로 값이 입력되었음.

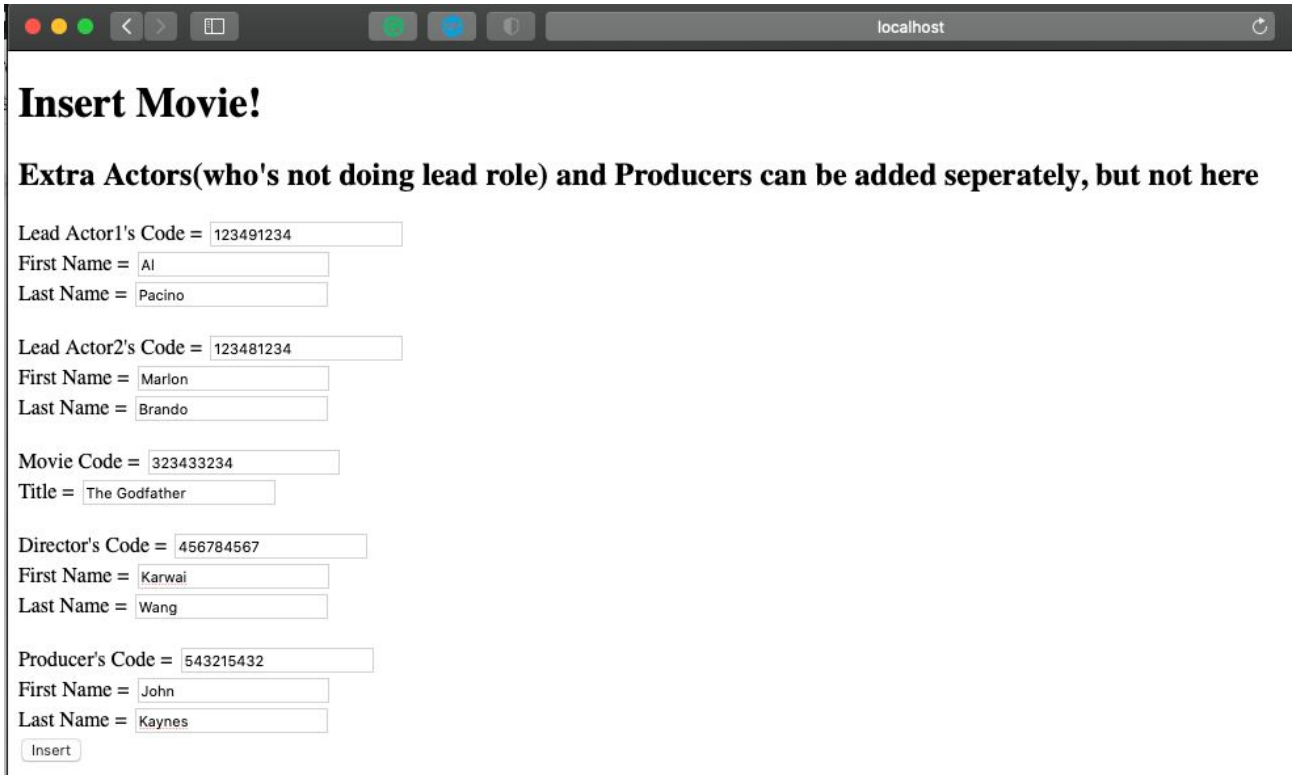


Create Producer 선택 후 값 입력


```
mysql> select * from PRODUCER where PRODUCER_CODE='234562345';
+---------------+-------+-------+------------+
| producer_code | fname | lname | actor_code |
+---------------+-------+-------+------------+
| 234562345     | Adam  | Smith | NULL       |
+---------------+-------+-------+------------+
```

```
mysql> select * from PRODUCES where PRODUCER_CODE='234562345';
+------------+---------------+
| MOVIE_CODE | producer_code |
+------------+---------------+
| 333333331  | 234562345     |
+------------+---------------+
```

입력완료 메시지와 데이터베이스 (Producer, Produces 테이블)에도 정상적으로 입력된 모습.

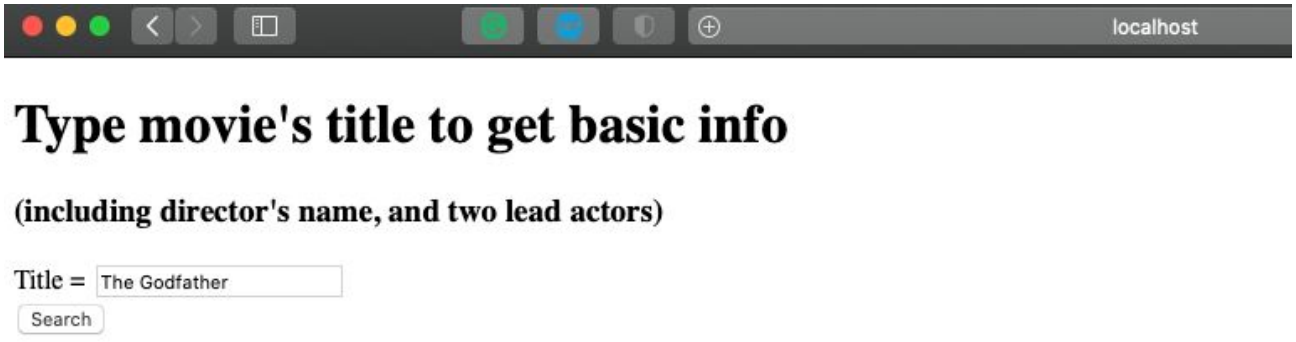


Create Movie 선택 및 값 입력 모습

(여기서는 모두 새로운 사람들을 입력했지만 기존에 있던 사람들을 입력해도 상관 없음)


```
mysql> select * from MOVIE_INFO where TITLE='The Godfather';
+---------------+----------------+----------------+-------------+-------------+
| TITLE         | DIRECTOR_FNAME | DIRECTOR_LNAME | ACTOR_FNAME | ACTOR_LNAME |
+---------------+----------------+----------------+-------------+-------------+
| The Godfather | Karwai         | Wang           | Marlon      | Brando      |
+---------------+----------------+----------------+-------------+-------------+
```

입력 완료 모습과 데이터베이스에도 입력된 모습. 해당 배우, 감독, 프로듀서 또한 모두 잘 입력되었음.



View Movie선택 및 값 입력 모습.

Info of The Godfather

Director's name : Karwai Wang
Actor1's name : Marlon Brando
Actor2's name : Al Pacino

영화에대한 기본정보가 잘 보이는 모습

Type movie's title to know all the actors in that piece.

Title = Green Flash

Search

View Actors 선택 및 값 입력 모습

Actors in Green Flash

Actor's name : Brad Pitt

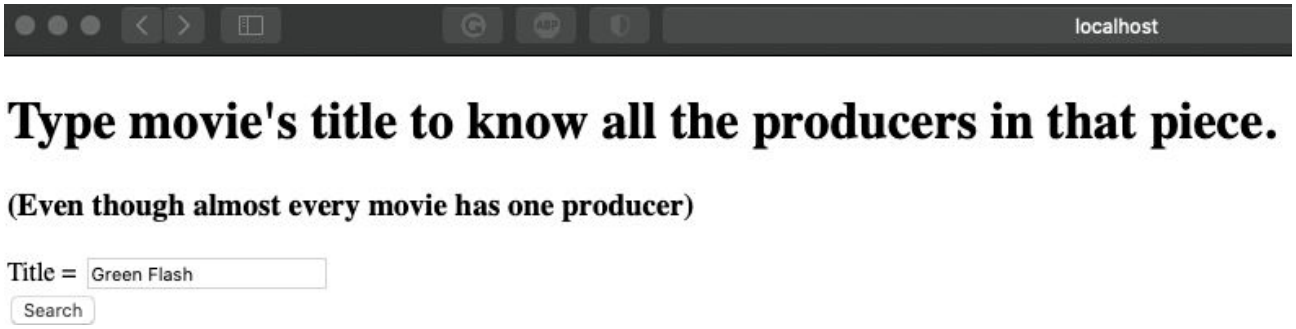
Actor's name : George Clooney

Actor's name : Matt Damon

Actor's name : Matthew McConaughey

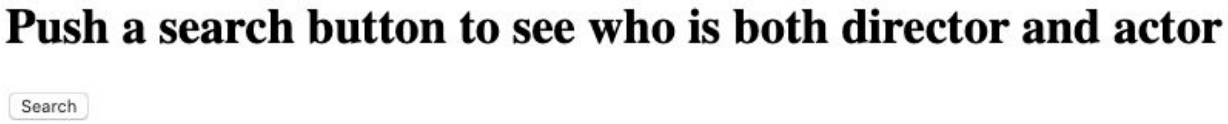
```
mysql> select * from ACTORS_IN_MOVIE where TITLE='Green Flash';
+-------------+---------+-------------+
| TITLE       | FNAME   | LNAME       |
+-------------+---------+-------------+
| Green Flash | Brad    | Pitt        |
| Green Flash | George  | Clooney     |
| Green Flash | Matt    | Damon       |
| Green Flash | Matthew | McConaughey |
+-------------+---------+-------------+
4 rows in set (0.01 sec)
```

영화에 나오는 모든 배우들의 이름을 다 볼 수 있는 모습(데이터베이스에서 확인해도 동일)

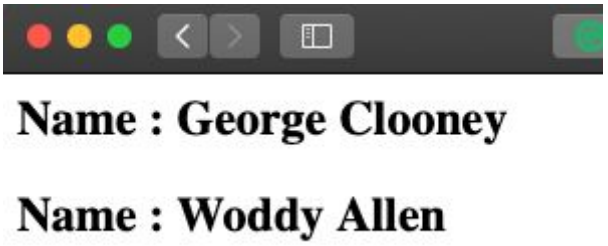


View Producer 선택 및 값 입력 모습

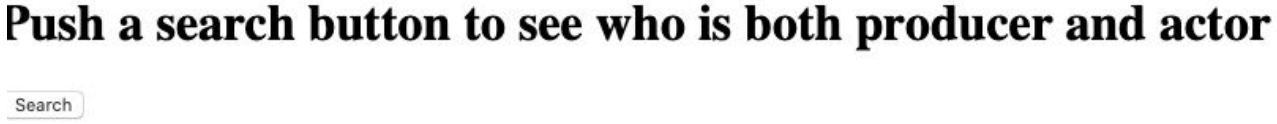
참여 프로듀서들의 목록을 잘 볼 수 있는 모습. View Table을 잘 읽어옴을 알 수 있다.



View Director&Actor 선택모습

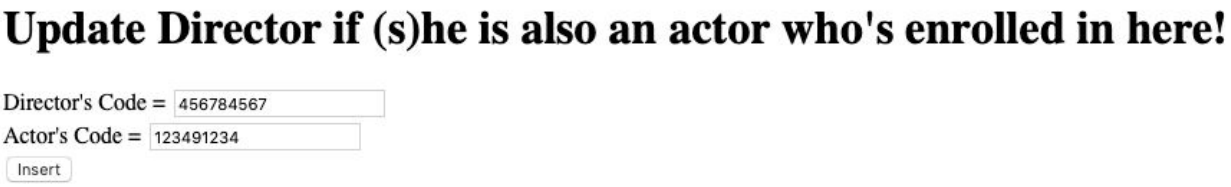


버튼 누르면 목록을 볼 수 있음. 이 두명이 감독이자 배우를 둘 다 하는 사람들이다.



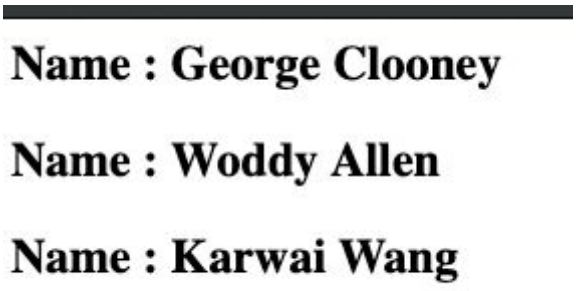
View Producer&Actor 선택모습.

버튼을 누르면 목록을 볼 수 있음. 이 두명이 프로듀서이자 배우를 모두 하는 사람들이다.

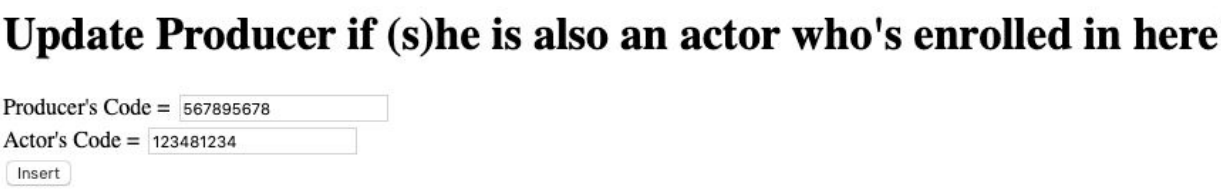


update Director&Actor 선택 및 값 입력 모습 (create Actor로 가서 Karwai Wang을 배우로도 하나 만들었음)

연결 잘 되었다는 메세지



아까 View Director&Actor 로 봐도 잘 되었다는게 보임.



Update Producer&Actor 선택 및 값 입력모습. (create producer에서 Marlon Brando 프로듀서로도 하나 만들고옴)

123481234 is now connected with 567895678 !

버튼 누르면 완료되었다는 메시지.

Name : Brad Pitt

Name : Matt Damon

Name : Marlon Brando

아까 view Producer&Actor로 다시봐도 연결 잘 되었다는걸 알 수 있음.

Delete Actor who's not doing a lead role

(He also should not be a director or producer)

Actor's code = 123456789

Delete

Delete Actor 선택 및 값 (처음에 만든 Jay Chou 코드임) 입력모습

123456789 is succesfully deleted!

```
mysql> select * from ACTOR;
+------------+---------+-------------+
| actor_code | fname   | lname       |
+------------+---------+-------------+
| 111111111  | Brad    | Pitt        |
| 111111112  | George  | Clooney     |
| 111111113  | Matt    | Damon       |
| 111111114  | Matthew | McConaughey |
| 111111115  | Tom     | Cruise      |
| 111111116  | Woddy   | Allen       |
| 123481234  | Marlon  | Brando      |
| 123491234  | Karwai  | Wang        |
+------------+---------+-------------+
8 rows in set (0.00 sec)
```

입력완료 메세지와 데이터베이스에서도 삭제 완료되어 더이상 Jay Chou볼 수 없음.

Delete Producer

(But he should not be the only producer in movies, and should not be an actor at the same time)

Producer's code = 234562345

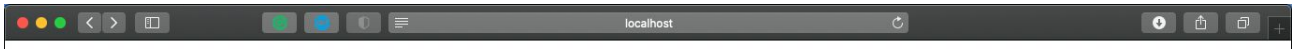
Delete

Delete Producer 선택모습과 값 (처음에 입력했던 Adam Smith 코드) 입력모습

234562345 is succesfully deleted!

```
mysql> select * from PRODUCER;
+---------------+-----------+---------+-------------+
| producer_code | fname     | lname   | actor_code  |
+---------------+-----------+---------+-------------+
| 444444441     | Yosep     | Kim     | NULL        |
| 444444442     | Brad      | Pitt    | 111111111   |
| 444444443     | Coco      | Chanel  | NULL        |
| 444444444     | Matt      | Damon   | 111111113   |
| 444444445     | Seunghyup | Na      | NULL        |
| 543215432     | John      | Kaynes  | NULL        |
| 567895678     | Marlon    | Brando  | 123481234   |
+---------------+-----------+---------+-------------+
7 rows in set (0.00 sec)
```

삭제 완료모습. 데이터베이스에서도 더이상 Adam Smith 볼 수 없음.



Designed By Duckhyung Ryu(12161725)

Create

create actor : You can insert actor who's not in a lead_role
create producer : You can insert extra producer to the movie
create movie : You can insert movie with every information of it

Read

Using View Tables

view movie : You can see the basic info of a specific movie
view actors : You can see all the actors in a specific movie
view producers : You can see all the producers in a specific movie
view Director&Actor : You can see people who's both director and actor at the same time
view Producer&Actor : You can see people who's both producer and actor at the same time

Update

You may need to know the codes that is 9digit

connect director&Actor : You can connect them if you think they are same person.
connect producer&Actor : You can connect them if you think they are same person.

Delete

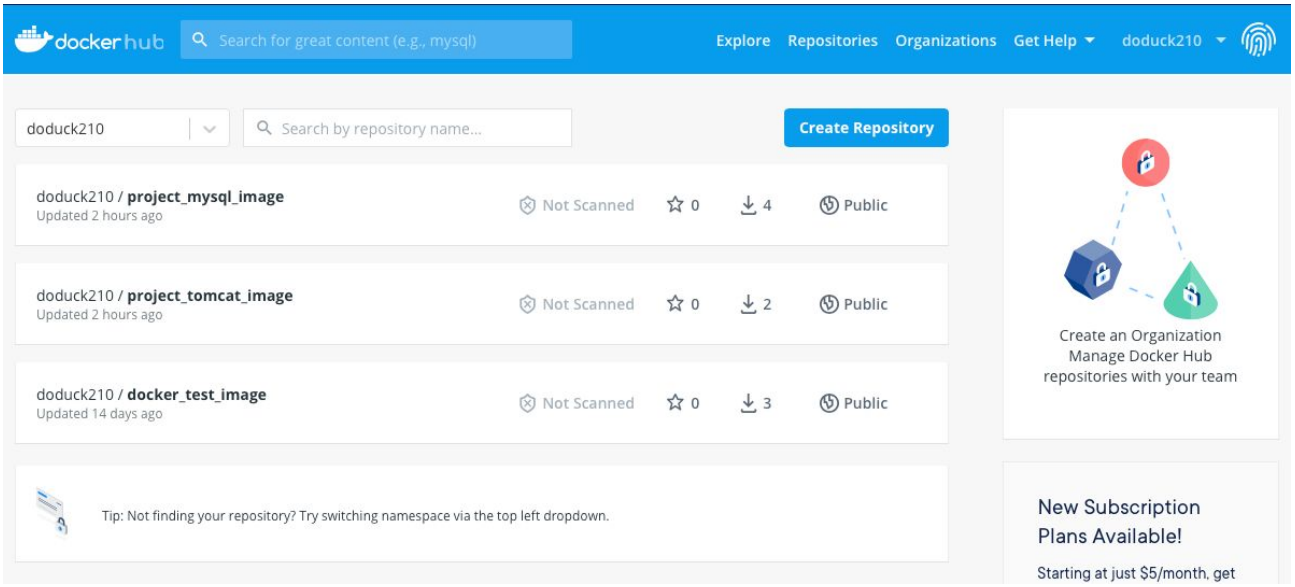
You may need to know the codes that is 9digit

delete actor : You can delete actor if he's not playing lead_role
delete producer : You can delete producer if he's not the only one producer in movies

모든 기능들을 다 이용해봤고 모두 정상작동함을 볼 수 있었다.

Docker Upload

실습 11, 12에서 배운 Docker tomcat-server를 사용해 작업할 때는 DatabaseUtil을 사용해야 database에 접속이 되어서 기존 DAO코드에서 데이터베이스에 접근하는 부분을 util.DatabaseUtil을 import해 여기에있는 getConnection을 이용하게 했다. 여기서 Database에 접근은 “docker inspect mysql’ 명령어로 볼 수 있는 mysql의 IP 주소(172.17.0.2 였음)를 사용했다. 그리고 web.xml에서 servlet이 실행될 수 있게 설정도 해주고, html의 action도 여기에 맞춰서 변경해주니 잘 동작함을 볼 수 있었다. 그 후 mysql과 tomcat-server컨테이너를 저장하고 이미지를 docker hub에 업로드하였다.



위에 두개가 각각 이번 프로젝트의 mysql, tomcat-server

4. 고찰

한학기동안 이론 및 실습시간에 배운 모든 내용들을 총 망라해 데이터베이스를 만들고 MVC모델을 준수해서 이 데이터베이스와 연동되는 (CRUD모두 가능한) 웹을 만들어 보았다.

ERD를 바탕으로 데이터베이스를 만들때는 True or False에 답하며 ERD에 대한 해석을 구체화했고 이를 토대로 다시 ERD를 그려보고 나중에 데이터베이스에 문제가 생기지 않도록 설정을 고려해서 Workbench로 이를 구현해보았다. Foreign키들의 관계에 유의해 순서를 신경쓰며 값들을 넣어보고 사용자들이 원할 만한 값들을 편하고 안전하게 볼 수 있게 하기 위해 View Table을 만들었다. DAO, DTO, Servlet을 사용해 Back End를, html로 Front End를 구성했고, 데이터베이스에서 강제할 수 없었던 것 (주연배우 2명 조건)은 여기서 강제할 수 있었다. 이로써 사용자와 엔지니어 단을 잘 구분해주는 MVC 디자인패턴이 잘 적용된 데이터베이스-웹을 만들 수 있었다.

CRUD를 하나씩 만들다 보니 이런기능이 있으면 더 좋겠다, 어떤 오류는 피해야겠다 싶은게 계속 생겨 CRUD각각 하나씩이 아닌 총 12개의 기능을 만들었고 DAO함수도 길어졌는데, 이러한 경험을 통해 사용자의

needs에 맞춰 어떤 기능이라도 만들 수 있겠다는 자신감이 생겼다. 또한 웹에서 기능을 구현하면 구현할 수록 처음에 데이터베이스를 잘 만들어 놓는게 굉장히 중요함을 깨달았다. 이론시간에 배운 많은 내용이 실제로 설계에서 중요했기에 이를 정리해보며 실용적으로 다뤄볼 수 있어 좋았고, 앞으로 데이터베이스 혹은 웹을 설계할 때 이번 프로젝트를 잊지 않으며 사용자와 엔지니어를 잘 구분하는 (MVC모델을 준수하는) 좋은 엔지니어가 되도록 하겠다.

5. Appendix

Movies Database 생성 스크립트 (Index 포함)

```
-- 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,ERROR_FOR_DIVISION_
BY_ZERO,NO_ENGINE_SUBSTITUTION';



-- -----------------------------------------------------

-- Schema MOVIES

-- -----------------------------------------------------



-- -----------------------------------------------------

-- Schema MOVIES

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `MOVIES` DEFAULT CHARACTER SET utf8 ;

USE `MOVIES` ;



-- -----------------------------------------------------

-- Table `MOVIES`.`ACTOR`

-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `MOVIES`.`ACTOR` (

 `actor_code` CHAR(9) NOT NULL,

 `fname` VARCHAR(45) NOT NULL,

 `lname` VARCHAR(45) NOT NULL,

 PRIMARY KEY (`actor_code`),

 UNIQUE INDEX `actor_code_UNIQUE` (`actor_code` ASC) VISIBLE)

ENGINE = InnoDB;



-- -----------------------------------------------------

-- Table `MOVIES`.`DIRECTOR`

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

```
CREATE TABLE IF NOT EXISTS `MOVIES`.`DIRECTOR` (
 `Director_Code` CHAR(9) NOT NULL,
 `fname` VARCHAR(45) NOT NULL,
 `lname` VARCHAR(45) NOT NULL,
 `actor_code` CHAR(9) NULL,
 PRIMARY KEY (`Director_Code`),
 UNIQUE INDEX `Director_Code_UNIQUE` (`Director_Code` ASC) VISIBLE,
 INDEX `actor` (`actor_code` ASC) VISIBLE,
 CONSTRAINT `fk_DIRECTOR_ACTOR1`
   FOREIGN KEY (`actor_code`)
   REFERENCES `MOVIES`.`ACTOR` (`actor_code`)
   ON DELETE NO ACTION
   ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `MOVIES`.`MOVIE`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `MOVIES`.`MOVIE` (
 `MOVIE_CODE` CHAR(9) NOT NULL,
 `title` VARCHAR(45) NOT NULL,
 `Director_Code` CHAR(9) NOT NULL,
 PRIMARY KEY (`MOVIE_CODE`),
 UNIQUE INDEX `MOVIE_CODE_UNIQUE` (`MOVIE_CODE` ASC) VISIBLE,
 INDEX `director` (`Director_Code` ASC) VISIBLE,
 CONSTRAINT `fk_MOVIE_DIRECTOR1`
   FOREIGN KEY (`Director_Code`)
   REFERENCES `MOVIES`.`DIRECTOR` (`Director_Code`)
   ON DELETE NO ACTION
   ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `MOVIES`.`PERFORMS_IN`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `MOVIES`.`PERFORMS_IN` (
 `movie_code` CHAR(9) NOT NULL,
 `actor_code` CHAR(9) NOT NULL,
 PRIMARY KEY (`movie_code`, `actor_code`),
 INDEX `actor` (`actor_code` ASC) VISIBLE,
 INDEX `movie` (`movie_code` ASC) VISIBLE,
```

```
 CONSTRAINT `fk_MOVIE_has_ACTOR_MOVIE`
   FOREIGN KEY (`movie_code`)
   REFERENCES `MOVIES`.`MOVIE` (`MOVIE_CODE`)
   ON DELETE CASCADE
   ON UPDATE NO ACTION,
 CONSTRAINT `fk_MOVIE_has_ACTOR_ACTOR1`
   FOREIGN KEY (`actor_code`)
   REFERENCES `MOVIES`.`ACTOR` (`actor_code`)
   ON DELETE CASCADE
   ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `MOVIES`.`PRODUCER`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `MOVIES`.`PRODUCER` (
 `producer_code` CHAR(9) NOT NULL,
 `fname` VARCHAR(45) NOT NULL,
 `lname` VARCHAR(45) NOT NULL,
 `actor_code` CHAR(9) NULL,
 PRIMARY KEY (`producer_code`),
 UNIQUE INDEX `producer_code_UNIQUE` (`producer_code` ASC) VISIBLE,
 INDEX `actor` (`actor_code` ASC) VISIBLE,
 CONSTRAINT `fk_PRODUCER_ACTOR1`
   FOREIGN KEY (`actor_code`)
   REFERENCES `MOVIES`.`ACTOR` (`actor_code`)
   ON DELETE NO ACTION
   ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `MOVIES`.`LEAD_ROLE`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `MOVIES`.`LEAD_ROLE` (
 `MOVIE_CODE` CHAR(9) NOT NULL,
 `actor_code` CHAR(9) NOT NULL,
 PRIMARY KEY (`MOVIE_CODE`, `actor_code`),
 INDEX `actor` (`actor_code` ASC) VISIBLE,
 INDEX `movie` (`MOVIE_CODE` ASC) VISIBLE,
 CONSTRAINT `fk_MOVIE_has_ACTOR_MOVIE1`
   FOREIGN KEY (`MOVIE_CODE`)
```

```
  REFERENCES `MOVIES`.`MOVIE` (`MOVIE_CODE`)
  ON DELETE CASCADE
  ON UPDATE NO ACTION,
 CONSTRAINT `fk_MOVIE_has_ACTOR_ACTOR2`
  FOREIGN KEY (`actor_code`)
  REFERENCES `MOVIES`.`ACTOR` (`actor_code`)
  ON DELETE CASCADE
  ON UPDATE NO ACTION)
ENGINE = InnoDB;



-- -----------------------------------------------------
-- Table `MOVIES`.`PRODUCES`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `MOVIES`.`PRODUCES` (
 `MOVIE_CODE` CHAR(9) NOT NULL,
 `producer_code` CHAR(9) NOT NULL,
 PRIMARY KEY (`MOVIE_CODE`, `producer_code`),
 INDEX `producer` (`producer_code` ASC) VISIBLE,
 INDEX `movie` (`MOVIE_CODE` ASC) VISIBLE,
 CONSTRAINT `fk_MOVIE_has_PRODUCER_MOVIE1`
  FOREIGN KEY (`MOVIE_CODE`)
  REFERENCES `MOVIES`.`MOVIE` (`MOVIE_CODE`)
  ON DELETE CASCADE
  ON UPDATE NO ACTION,
 CONSTRAINT `fk_MOVIE_has_PRODUCER_PRODUCER1`
  FOREIGN KEY (`producer_code`)
  REFERENCES `MOVIES`.`PRODUCER` (`producer_code`)
  ON DELETE CASCADE
  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;
```

기본 Value Insert 문

```
use Movies;
```

```
INSERT INTO Actor
VALUES   ('111111111','Brad','Pitt')
       ,('111111112','George','Clooney')
       ,('111111113','Matt','Damon')
       ,('111111114','Matthew','McConaughey')
       ,('111111115','Tom','Cruise')
       ,('111111116','Woddy','Allen');

INSERT INTO Director
VALUES   ('222222221','Eirc','Rohmer',null)
       ,('222222222','George','Clooney','111111112')
       ,('222222223','David','Fincher',null)
       ,('222222224','Quentine','Tarantino',null)
       ,('222222225','Woddy','Allen','111111116');

INSERT INTO Movie
VALUES   ('333333331','Green Flash','222222221')
       ,('333333332','Good night and Good luck','222222222')
       ,('333333333','Fight Club','222222223')
       ,('333333334','Once upon a time in hollywood','222222224')
       ,('333333335','Manhatten','222222225');

INSERT INTO Producer
VALUES   ('444444441','Yosep','Kim',null)
       ,('444444442','Brad','Pitt','111111111')
       ,('444444443','Coco','Chanel',null)
       ,('444444444','Matt','Damon','111111113')
       ,('444444445','Seunghyup','Na',null);

INSERT INTO PERFORMS_IN
VALUES   ('333333332','111111112')
       ,('333333332','111111113')
       ,('333333332','111111111')
       ,('333333333','111111111')
       ,('333333333','111111114')
       ,('333333333','111111115')
       ,('333333334','111111111')
       ,('333333334','111111115')
       ,('333333335','111111116')
       ,('333333335','111111114')
       ,('333333331','111111114')
       ,('333333331','111111111')
       ,('333333331','111111112')
```

```
      ,('333333331','111111113');


INSERT INTO LEAD_ROLE
VALUES   ('333333332','111111112')
      ,('333333332','111111113')
      ,('333333333','111111111')
      ,('333333333','111111114')
      ,('333333334','111111111')
      ,('333333334','111111115')
      ,('333333335','111111116')
      ,('333333335','111111114')
      ,('333333331','111111114')
      ,('333333331','111111111');


INSERT INTO Produces
VALUES   ('333333331','444444444')
      ,('333333332','444444443')
      ,('333333333','444444441')
      ,('333333334','444444442')
      ,('333333335','444444445')
      ,('333333335','444444443');
```

View Table Create문

```
USE MOVIES;


CREATE VIEW MOVIE_INFO AS
SELECT   DISTINCT   TITLE,   DIRECTOR.FNAME   AS   DIRECTOR_FNAME,   DIRECTOR.LNAME   AS
DIRECTOR_LNAME,
   ACTOR.FNAME AS ACTOR_FNAME, ACTOR.LNAME AS ACTOR_LNAME
   FROM MOVIE, DIRECTOR, ACTOR, LEAD_ROLE
WHERE MOVIE.MOVIE_CODE=LEAD_ROLE.MOVIE_CODE AND LEAD_ROLE.ACTOR_CODE=ACTOR.ACTOR_CODE
   AND MOVIE.DIRECTOR_CODE=DIRECTOR.DIRECTOR_CODE;


CREATE VIEW ACTORS_IN_MOVIE AS
SELECT TITLE, ACTOR.FNAME, ACTOR.LNAME FROM MOVIE, PERFORMS_IN, ACTOR
WHERE                    ACTOR.ACTOR_CODE=PERFORMS_IN.ACTOR_CODE                    AND
PERFORMS_IN.MOVIE_CODE=MOVIE.MOVIE_CODE;


CREATE VIEW PRODUCERS_IN_MOVIE AS
SELECT TITLE, PRODUCER.FNAME, PRODUCER.LNAME FROM PRODUCER, PRODUCES, MOVIE
WHERE                  PRODUCER.PRODUCER_CODE=PRODUCES.PRODUCER_CODE                  AND
```

```
PRODUCES.MOVIE_CODE=MOVIE.MOVIE_CODE;


CREATE VIEW ACTOR_DIRECTOR AS
SELECT ACTOR.FNAME, ACTOR.LNAME FROM ACTOR, DIRECTOR
WHERE DIRECTOR.ACTOR_CODE=ACTOR.ACTOR_CODE;


CREATE VIEW ACTOR_PRODUCER AS
SELECT ACTOR.FNAME, ACTOR.LNAME FROM ACTOR, PRODUCER
WHERE PRODUCER.ACTOR_CODE=ACTOR.ACTOR_CODE;
```

Create Actor html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Create Actor</title>
</head>
<body>
   <h1>Insert an actor to movie who's not in lead_role!</h1>
   <form method='get' action='/myfirstweb/movie_servlet'>
       Actor's Code = <input type='text' name='actorCode' placeholder="9 numbers" required><br>
       First Name = <input type='text' name='fName' required><br>
       Last Name = <input type='text' name='lName' required><br>
       Movie Code = <input type='text' name='movieCode' placeholder="9 numbers" required><br>
       <input type='hidden' name='mode' value='Create_Actor'>
       <input type='submit' value='Insert'><br>
   </form>


</body>
</html>
```

Create Producer html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Create Producer</title>
</head>
<body>
```

```
   <h1>Insert producer to movie!</h1>
   <form method='get' action='/myfirstweb/movie_servlet'>
       Producer's Code = <input type='text' name='proCode' placeholder="9 numbers" required><br>
       First Name = <input type='text' name='fName' required><br>
       Last Name = <input type='text' name='lName' required><br>
       Movie Code = <input type='text' name='movieCode' placeholder="9 numbers" required><br>
       Actor's Code = <input type='text' name='actCode' placeholder="only if it is"><br>
       <input type='hidden' name='mode' value='Create_Producer'>
       <input type='submit' value='Insert'><br>
   </form>

</body>
</html>
```

Create Movie html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Create Movie</title>
</head>
<body>
   <h1>Insert Movie!</h1>
    <h2>Extra Actors(who's not doing lead role) and Producers can be added seperately, but not here</h2>
   <form method='get' action='/myfirstweb/movie_servlet'>
           Lead Actor1's Code = <input type='text' name='act1Code' placeholder="9 numbers" required><br>
       First Name = <input type='text' name='act1fname' required><br>
       Last Name = <input type='text' name='act1lname' required><br><br>
           Lead Actor2's Code = <input type='text' name='act2Code' placeholder="9 numbers" required><br>
       First Name = <input type='text' name='act2fname' required><br>
       Last Name = <input type='text' name='act2lname' required><br><br>
       Movie Code = <input type='text' name='movieCode' placeholder="9 numbers" required><br>
       Title = <input type='text' name='title' required><br><br>
       Director's Code = <input type='text' name='dirCode' placeholder="9 numbers" required><br>
       First Name = <input type='text' name='dirfname' required><br>
       Last Name = <input type='text' name='dirlname' required><br><br>
       Producer's Code = <input type='text' name='proCode' placeholder="9 numbers" required><br>
       First Name = <input type='text' name='profname' required><br>
       Last Name = <input type='text' name='prolname' required><br>
```

```
        <input type='hidden' name='mode' value='Create_Movie'>
        <input type='submit' value='Insert'><br>
    </form>


</body>
</html>
```

Read Movie html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Read Movie</title>
</head>
<body>
    <h1>Type movie's title to get basic info</h1>
    <h3>(including director's name, and two lead actors)</h3>
    <form method='get' action='/myfirstweb/movie_servlet'>
        Title = <input type='text' name='title'><br>
        <input type='hidden' name='mode' value='Read_Movie'>
        <input type='submit' value='Search'><br>
    </form>


</body>
</html>
```

Read Actors html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Read Actors</title>
</head>
<body>
    <h1>Type movie's title to know all the actors in that piece.</h1>
    <form method='get' action='/myfirstweb/movie_servlet'>
        Title = <input type='text' name='title'><br>
        <input type='hidden' name='mode' value='Read_Actor'>
        <input type='submit' value='Search'><br>
```

```
    </form>


</body>
</html>
```

Read Producer html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Read Producers</title>
</head>
<body>
    <h1>Type movie's title to know all the producers in that piece.</h1>
    <h3>(Even though almost every movie has one producer)</h3>
    <form method='get' action='/myfirstweb/movie_servlet'>
        Title = <input type='text' name='title'><br>
        <input type='hidden' name='mode' value='Read_Producers'>
        <input type='submit' value='Search'><br>
    </form>


</body>
</html>
```

Read Director&Actor html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Read Director&Actor</title>
</head>
<body>
    <h1>Push a search button to see who is both director and actor</h1>
    <form method='get' action='/myfirstweb/movie_servlet'>
        <input type='hidden' name='mode' value='Read_Diract'>
        <input type='submit' value='Search'><br>
    </form>


</body>
```

```
</html>
```

Read Producer&Actor html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Read Producer&Actor</title>
</head>
<body>
   <h1>Push a search button to see who is both producer and actor</h1>
   <form method='get' action='/myfirstweb/movie_servlet'>
       <input type='hidden' name='mode' value='Read_Proact'>
       <input type='submit' value='Search'><br>
   </form>

</body>
</html>
```

Update Director&Actor html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Update Director&Actor</title>
</head>
<body>
   <h1>Update Director if (s)he is also an actor who's enrolled in here!</h1>
   <form method='get' action='/myfirstweb/movie_servlet'>
       Director's Code = <input type='text' name='dirCode' placeholder="9 numbers" required><br>
       Actor's Code = <input type='text' name='actCode' placeholder="9 numbers" required><br>
       <input type='hidden' name='mode' value='Update_Diract'>
       <input type='submit' value='Insert'><br>
   </form>

</body>
</html>
```

Update Producer&Actor html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Update Producer&Actor</title>
</head>
<body>
   <h1>Update Producer if (s)he is also an actor who's enrolled in here!</h1>
   <form method='get' action='/myfirstweb/movie_servlet'>
       Producer's Code = <input type='text' name='proCode' placeholder="9 numbers" required><br>
       Actor's Code = <input type='text' name='actCode' placeholder="9 numbers" required><br>
       <input type='hidden' name='mode' value='Update_Proact'>
       <input type='submit' value='Insert'><br>
   </form>

</body>
</html>
```

Delete Actor html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Delete Actor</title>
</head>
<body>
   <h1>Delete Actor who's not doing a lead role</h1>
   <h2>(He also should not be a director or producer)</h2>
   <form method='get' action='/myfirstweb/movie_servlet'>
       Actor's code = <input type='text' name='actCode' placeholder="9 numbers" required><br>
       <input type='hidden' name='mode' value='Delete_Actor'>
       <input type='submit' value='Delete'><br>
   </form>

</body>
</html>
```

Delete Producer html

```
<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Delete Producer</title>

</head>

<body>

   <h1>Delete Producer</h1>

    <h2>(But he should not be the only producer in movies, and should not be an actor at the same time)</h2>

   <form method='get' action='/myfirstweb/movie_servlet'>

       Producer's code = <input type='text' name='proCode' placeholder="9 numbers" required><br>

       <input type='hidden' name='mode' value='Delete_Producer'>

       <input type='submit' value='Delete'><br>

   </form>


</body>

</html>
```

Index html

```
<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Movie Database Web</title>

</head>

<body>

   <h3>Designed By Duckhyung Ryu(12161725)</h3>

   <h1>Create</h1>

   <p>

     <a href='/myfirstweb/movie_create_actor.html'>create actor</a> : You can insert actor who's not in a lead_role <br>

     <a href='/myfirstweb/movie_create_producer.html'>create producer</a> : You can insert extra producer to the movie<br>

     <a href='/myfirstweb/movie_create_movie.html'>create movie</a> : You can insert movie with every information of it <br>

   </p>

   <h1>Read</h1>

   <h3>Using View Tables</h3>

   <p>

     <a href='/myfirstweb/movie_read_movie.html'>view movie</a> : You can see the basic info of a specific movie<br>
```

```
    <a href='/myfirstweb/movie_read_actors.html'>view actors</a> : You can see all the actors in a specific movie<br>

     <a href='/myfirstweb/movie_read_producers.html'>view producers</a> : You can see all the producers in a specific movie <br>

     <a href='/myfirstweb/movie_read_diract.html'>view Director&Actor</a> : You can see people who's both director and actor at the same time<br>

     <a href='/myfirstweb/movie_read_proact.html'>view Producer&Actor</a> : You can see people who's both producer and actor at the same time<br>

   </p>

   <h1>Update</h1>

   <h3>You may need to know the codes that is 9digit</h3>

   <p>

     <a href='/myfirstweb/movie_update_diract.html'>connect director&Actor</a> : You can connect them if you think they are same person.<br>

     <a href='/myfirstweb/movie_update_proact.html'>connect producer&Actor</a> : You can connect them if you think they are same person.<br>

   </p>

   <h1>Delete</h1>

   <h3>You may need to know the codes that is 9digit</h3>

   <p>

    <a href='/myfirstweb/movie_delete_actor.html'>delete actor</a> : You can delete actor if he's not playing lead_role<br>

       <a href='/myfirstweb/movie_delete_producer.html'>delete producer</a> : You can delete producer if he's not the only one producer in movies<br>

   </p>

</body>

</html>
```

DTO java

```
package movies;



public class moviesDTO {

   public moviesDTO(String actor_code, String actor_fname, String actor_lname, String movie_code, String movie_title,

                 String dir_code, String dir_fname, String dir_lname, String pro_code, String pro_fname, String pro_lname) {

       super();

       this.actor_code = actor_code;

       this.actor_fname = actor_fname;

       this.actor_lname = actor_lname;

       this.movie_code = movie_code;

       this.movie_title = movie_title;

       this.dir_code = dir_code;

       this.dir_fname = dir_fname;

       this.dir_lname = dir_lname;

       this.pro_code = pro_code;
```

```
        this.pro_fname = pro_fname;
        this.pro_lname = pro_lname;
    }
    private String actor_code;
    private String actor_fname;
    private String actor_lname;

    private String movie_code;
    private String movie_title;

    private String dir_code;
    private String dir_fname;
    private String dir_lname;

    private String pro_code;
    private String pro_fname;
    private String pro_lname;

    public String getActor_code() {
        return actor_code;
    }
    public void setActor_code(String actor_code) {
        this.actor_code = actor_code;
    }
    public String getActor_fname() {
        return actor_fname;
    }
    public void setActor_fname(String actor_fname) {
        this.actor_fname = actor_fname;
    }
    public String getActor_lname() {
        return actor_lname;
    }
    public void setActor_lname(String actor_lname) {
        this.actor_lname = actor_lname;
    }
    public String getMovie_code() {
        return movie_code;
    }
    public void setMovie_code(String movie_code) {
        this.movie_code = movie_code;
    }
    public String getMovie_title() {
```

```
        return movie_title;
    }
    public void setMovie_title(String movie_title) {
        this.movie_title = movie_title;
    }
    public String getDir_code() {
        return dir_code;
    }
    public void setDir_code(String dir_code) {
        this.dir_code = dir_code;
    }
    public String getDir_fname() {
        return dir_fname;
    }
    public void setDir_fname(String dir_fname) {
        this.dir_fname = dir_fname;
    }
    public String getDir_lname() {
        return dir_lname;
    }
    public void setDir_lname(String dir_lname) {
        this.dir_lname = dir_lname;
    }
    public String getPro_code() {
        return pro_code;
    }
    public void setPro_code(String pro_code) {
        this.pro_code = pro_code;
    }
    public String getPro_fname() {
        return pro_fname;
    }
    public void setPro_fname(String pro_fname) {
        this.pro_fname = pro_fname;
    }
    public String getPro_lname() {
        return pro_lname;
    }
    public void setPro_lname(String pro_lname) {
        this.pro_lname = pro_lname;
    }
    @Override
    public String toString() {
```

```
          return "moviesDTO [actor_code=" + actor_code + ", actor_fname=" + actor_fname + ",
actor_lname=" + actor_lname

                + ", movie_code=" + movie_code + ", movie_title=" + movie_title + ", dir_code=" +
dir_code

                    + ", dir_fname=" + dir_fname + ", dir_lname=" + dir_lname + ", pro_code=" +
pro_code + ", pro_fname="

               + pro_fname + ", pro_lname=" + pro_lname + "]";

   }

}
```

DAO java

```
package movies;



import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.ArrayList;

import java.util.List;


import movies.moviesDTO;


public class moviesDAO {

                                            private               static               String
dburl="jdbc:mysql://127.0.0.1:13306/MOVIES?allowPublicKeyRetrieval=true&serverTimezone=UTC";

   private static String dbUser="root";

   private static String dbpasswd="root";


   public List<moviesDTO> readMovie(String title) {

       List<moviesDTO> list = new ArrayList<>();

       try {

           Class.forName("com.mysql.cj.jdbc.Driver");

       } catch(ClassNotFoundException e) {

           e.printStackTrace();

       }


           String  sql  =  "Select  DIRECTOR_FNAME,  DIRECTOR_LNAME,  ACTOR_FNAME,  ACTOR_LNAME  from
MOVIE_INFO where TITLE = ? ";


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

           ps.setString(1,title);

           try(ResultSet rs= ps.executeQuery()){
```

```
               while (rs.next()) {
                   String dirfname = rs.getString(1);
                   String dirlname =rs.getString(2);
                   String actorfname=rs.getString(4);
                   String actorlname=rs.getString(4);
                            moviesDTO movie = new moviesDTO("1",actorfname,actorlname," "," "," 
",dirfname,dirlname," "," "," ");
                   list.add(movie);
               }
               } catch(Exception e) {
               e.printStackTrace();
               }

           } catch(Exception ex) {
               ex.printStackTrace();
           }
       return list;
       }

   public List<moviesDTO> readActor(String title) {
       List<moviesDTO> list = new ArrayList<>();
       try {
           Class.forName("com.mysql.cj.jdbc.Driver");
       } catch(ClassNotFoundException e) {
           e.printStackTrace();
       }

       String sql = "Select FNAME, LNAME from ACTORS_IN_MOVIE where TITLE = ? ";

         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement 
ps=conn.prepareStatement(sql)){
           ps.setString(1,title);
           try(ResultSet rs= ps.executeQuery()){
               while (rs.next()) {
                   String actorfname = rs.getString(1);
                   String actorlname =rs.getString(2);
                      moviesDTO movie = new moviesDTO("1",actorfname,actorlname," "," "," "," "," 
"," "," "," ");
                   list.add(movie);
               }
               } catch(Exception e) {
               e.printStackTrace();
               }
```

```
            } catch(Exception ex) {
                ex.printStackTrace();
            }
        return list;
        }

    public List<moviesDTO> readDiract() {
        List<moviesDTO> list = new ArrayList<>();
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch(ClassNotFoundException e) {
            e.printStackTrace();
        }

        String sql = "Select FNAME, LNAME from ACTOR_DIRECTOR";

          try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

            try(ResultSet rs= ps.executeQuery()){
                while (rs.next()) {
                    String actfname = rs.getString(1);
                    String actlname =rs.getString(2);
                       moviesDTO movie = new moviesDTO(" ",actfname,actlname," "," "," "," "," ","
"," "," ");
                    list.add(movie);
                }
                } catch(Exception e) {
                e.printStackTrace();
                }

            } catch(Exception ex) {
                ex.printStackTrace();
            }
        return list;
        }
    public List<moviesDTO> readProducers(String title) {
        List<moviesDTO> list = new ArrayList<>();
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
        } catch(ClassNotFoundException e) {
            e.printStackTrace();
        }
```

```
       String sql = "Select FNAME, LNAME from PRODUCERS_IN_MOVIE where TITLE = ? ";


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

           ps.setString(1,title);

           try(ResultSet rs= ps.executeQuery()){

               while (rs.next()) {

                   String profname = rs.getString(1);

                   String prolname =rs.getString(2);

                           moviesDTO movie = new moviesDTO("  "," "," "," "," "," "," "," "," "
",profname,prolname);

                   list.add(movie);

               }

               } catch(Exception e) {

               e.printStackTrace();

               }


           } catch(Exception ex) {

               ex.printStackTrace();

           }

       return list;

       }


   public List<moviesDTO> readProact() {

       List<moviesDTO> list = new ArrayList<>();

       try {

           Class.forName("com.mysql.cj.jdbc.Driver");

       } catch(ClassNotFoundException e) {

           e.printStackTrace();

       }


       String sql = "Select FNAME, LNAME from ACTOR_PRODUCER";


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


           try(ResultSet rs= ps.executeQuery()){

               while (rs.next()) {

                   String actfname = rs.getString(1);

                   String actlname =rs.getString(2);

                      moviesDTO movie = new moviesDTO(" ",actfname,actlname," "," "," "," "," ","
","  "," "," ");

                   list.add(movie);

               }

               } catch(Exception e) {
```

```
            e.printStackTrace();
            }

        } catch(Exception ex) {
            ex.printStackTrace();
        }
    return list;
    }

  public moviesDTO createActor(String actorcode,String fname,String lname,String moviecode) {
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
    } catch(ClassNotFoundException e) {
        e.printStackTrace();
    }

    String sql = "Insert into ACTOR values (?,?,?) ";

      try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

        ps.setString(1, actorcode);
        ps.setString(2,fname);
        ps.setString(3, lname);

        ps.executeUpdate();

        } catch(Exception ex) {
            ex.printStackTrace();
        }

    sql = "Insert into PERFORMS_IN values (?,?) ";

      try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

        ps.setString(1, moviecode);
        ps.setString(2, actorcode);

        ps.executeUpdate();

        } catch(Exception ex) {
            ex.printStackTrace();
            sql="delete from ACTOR where ACTOR_CODE=? ";
```

```
                                                                                try(Connection
conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

                    ps.setString(1,actorcode);

                    ps.executeUpdate();

                    moviesDTO blank = new moviesDTO(" ","No Movie Found","Error<!--"," "," "," ","
","  "," "," "," ");

                    return blank;

                }

                catch(Exception eex) {

                    eex.printStackTrace();

                }

            }


         moviesDTO movie = new moviesDTO(actorcode,fname,lname,moviecode," "," "," "," "," "," ","
");

        return movie;

        }


          public   moviesDTO   createProducer(String   procode,String   fname,String   lname,String
moviecode,String actcode) {

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

        } catch(ClassNotFoundException e) {

            e.printStackTrace();

        }


        String sql = "Insert into PRODUCER (PRODUCER_CODE,FNAME,LNAME,ACTOR_CODE) values (?,?,?,?)
";

        if(actcode.isEmpty()) {

            sql = "Insert into PRODUCER (PRODUCER_CODE,FNAME,LNAME) values (?,?,?) ";

        }

          try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


            ps.setString(1, procode);

            ps.setString(2,fname);

            ps.setString(3, lname);

            if(!actcode.isEmpty()) {

            ps.setNString(4, actcode);

            }


            ps.executeUpdate();


            } catch(Exception ex) {

                ex.printStackTrace();
```

```
                moviesDTO blank = new moviesDTO(" "," "," "," "," "," "," "," "," ","Wrong Actor
Code","Error<!--");

                return blank;

            }


       sql = "Insert into PRODUCES values (?,?) ";


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


            ps.setString(1, moviecode);

            ps.setString(2, procode);


            ps.executeUpdate();


            } catch(Exception ex) {

                ex.printStackTrace();

                sql="delete from PRODUCER where PRODUCER_CODE=? ";

                                                                                try(Connection
conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

                    ps.setString(1,procode);

                    ps.executeUpdate();

                     moviesDTO blank = new moviesDTO(" "," "," "," "," "," "," "," "," ","No Movie
Found","Error<!--");

                    return blank;

                }

                catch(Exception eex) {

                    eex.printStackTrace();

                }

            }


                moviesDTO  movie  =  new  moviesDTO("  ","  ","  ",moviecode,"  ","  ","  ","
",procode,fname,lname);

       return movie;

       }


     public  moviesDTO  createMovie(String  act1Code,String  act1fname,String  act1lname,  String
act2Code, String act2fname, String act2lname

                                   ,String movieCode,String title, String dirCode,String dirfname,
String dirlname, String proCode,

                                String profname, String prolname) {

       try {

            Class.forName("com.mysql.cj.jdbc.Driver");

       } catch(ClassNotFoundException e) {

            e.printStackTrace();
```

```
       }


       String sql = "Insert into ACTOR values (?,?,?) ";


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


           ps.setString(1, act1Code);

           ps.setString(2,act1fname);

           ps.setString(3, act1lname);

           ps.executeUpdate();


           } catch(Exception ex) {

               ex.printStackTrace();

           }


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


           ps.setString(1, act2Code);

           ps.setString(2,act2fname);

           ps.setString(3, act2lname);

           ps.executeUpdate();


           } catch(Exception ex) {

               ex.printStackTrace();

           }


       sql = "Insert into DIRECTOR (DIRECTOR_CODE,FNAME,LNAME) values (?,?,?) ";


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


           ps.setString(1, dirCode);

           ps.setString(2, dirfname);

           ps.setString(3, dirlname);

           ps.executeUpdate();


           } catch(Exception ex) {

               ex.printStackTrace();

           }


       sql = "Insert into PRODUCER (PRODUCER_CODE,FNAME,LNAME) values (?,?,?) ";
```

```
         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


           ps.setString(1, proCode);

           ps.setString(2, profname);

           ps.setString(3, prolname);

           ps.executeUpdate();


           } catch(Exception ex) {

               ex.printStackTrace();

           }


       sql = "Insert into MOVIE values (?,?,?) ";


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


           ps.setString(1, movieCode);

           ps.setString(2, title);

           ps.setString(3, dirCode);

           ps.executeUpdate();


           } catch(Exception ex) {

               ex.printStackTrace();

           }


       sql = "Insert into PERFORMS_IN values (?,?) ";


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


           ps.setString(1, movieCode);

           ps.setString(2, act1Code);

           ps.executeUpdate();


           } catch(Exception ex) {

               ex.printStackTrace();

           }


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


           ps.setString(1, movieCode);

           ps.setString(2, act2Code);

           ps.executeUpdate();
```

```
          } catch(Exception ex) {
              ex.printStackTrace();
          }

       sql = "Insert into LEAD_ROLE values (?,?) ";

         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

          ps.setString(1, movieCode);
          ps.setString(2, act1Code);
          ps.executeUpdate();

          } catch(Exception ex) {
              ex.printStackTrace();
          }
         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

          ps.setString(1, movieCode);
          ps.setString(2, act2Code);
          ps.executeUpdate();

          } catch(Exception ex) {
              ex.printStackTrace();
          }

       sql = "Insert into PRODUCES values (?,?) ";

         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

          ps.setString(1, movieCode);
          ps.setString(2, proCode);
          ps.executeUpdate();

          } catch(Exception ex) {
              ex.printStackTrace();
          }


       moviesDTO movie = new moviesDTO(" "," "," "," ",title," "," "," "," "," "," ");
       return movie;
```

```
    }


   public moviesDTO updateDiract(String dirCode,String actCode) {

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

        } catch(ClassNotFoundException e) {

            e.printStackTrace();

        }


        String sql = "Update DIRECTOR set ACTOR_Code = ?  where DIRECTOR_CODE = ?";


          try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


            ps.setString(1, actCode);

            ps.setString(2, dirCode);


            ps.executeUpdate();


            } catch(Exception ex) {

                ex.printStackTrace();

                  moviesDTO blank = new moviesDTO("Wrong Code Error<!--"," "," "," "," "," "," "
","," "," "," ");

                return blank;

            }


        moviesDTO movie = new moviesDTO(actCode," "," "," "," ",dirCode," "," "," "," "," ");

        return movie;

        }


   public moviesDTO updateProact(String proCode,String actCode) {

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

        } catch(ClassNotFoundException e) {

            e.printStackTrace();

        }


        String sql = "Update PRODUCER set ACTOR_Code = ?  where PRODUCER_CODE = ?";


          try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


            ps.setString(1, actCode);

            ps.setString(2, proCode);
```

```
            ps.executeUpdate();


            } catch(Exception ex) {

                ex.printStackTrace();

                  moviesDTO blank = new moviesDTO("Wrong Code Error<!--"," "," "," "," "," "," "," "," "," "," ");

                return blank;

            }


        moviesDTO movie = new moviesDTO(actCode," "," "," "," "," "," "," ",proCode," "," ");

        return movie;

        }


    public moviesDTO deleteActor(String actCode) {

        try {

            Class.forName("com.mysql.cj.jdbc.Driver");

        } catch(ClassNotFoundException e) {

            e.printStackTrace();

        }


        String sql = "select ACTOR_CODE from LEAD_ROLE where ACTOR_CODE = ?";


          try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement ps=conn.prepareStatement(sql)){

            ps.setString(1,actCode);

            try(ResultSet rs= ps.executeQuery()){

                    rs.next();

                    String actorfname = rs.getString(1);

                    if(!actorfname.isEmpty()) {

                          moviesDTO error = new moviesDTO("Hey (s)he has lead role!<!--"," "," "," "," "," "," "," "," "," "," ");

                        return error;

                    }

                } catch(Exception e) {

                e.printStackTrace();

                }


            } catch(Exception ex) {

                ex.printStackTrace();

            }


        sql = "Delete from ACTOR where ACTOR_CODE = ?";
```

```
         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){


           ps.setString(1, actCode);


           ps.executeUpdate();


           } catch(Exception ex) {

               ex.printStackTrace();

                 moviesDTO error = new moviesDTO("(S)he has another role!<!--"," "," "," "," ","
","," "," "," "," "," ");

               return error;

           }


       moviesDTO movie = new moviesDTO(actCode," "," "," "," "," "," "," "," "," "," ");

       return movie;

       }


   public moviesDTO deleteProducer(String proCode) {

       try {

           Class.forName("com.mysql.cj.jdbc.Driver");

       } catch(ClassNotFoundException e) {

           e.printStackTrace();

       }


       String sql = "select MOVIE_CODE from PRODUCES where PRODUCER_CODE=?;";


         try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

           ps.setString(1,proCode);

           try(ResultSet rs= ps.executeQuery()){

               while (rs.next()) {

                   String movCode = rs.getString(1);


                   String sql1 = "select count(*) from PRODUCES where MOVIE_CODE= ? ";


                   try(PreparedStatement pps=conn.prepareStatement(sql1)){


                       pps.setString(1, movCode);

                       try(ResultSet rrs= pps.executeQuery()){

                           rrs.next();

                               int count = rrs.getInt(1);

                               if(count<2) {

                                     moviesDTO error = new moviesDTO(""," "," "," "," "," "," ","
","he is the only producer! <!--"," "," ");
```

```
                                return error;
                            }
                        } catch(Exception e) {
                        e.printStackTrace();
                        }

                    } catch(Exception ex) {
                        ex.printStackTrace();
                    }

            }
            } catch(Exception e) {
            e.printStackTrace();
            }

        } catch(Exception ex) {
            ex.printStackTrace();
        }

    sql = "Delete from PRODUCER where PRODUCER_CODE = ? ";

      try(Connection conn=DriverManager.getConnection(dburl,dbUser,dbpasswd);PreparedStatement
ps=conn.prepareStatement(sql)){

        ps.setString(1, proCode);

        ps.executeUpdate();

        } catch(Exception ex) {
            ex.printStackTrace();
        }

    moviesDTO movie = new moviesDTO(" "," "," "," "," "," "," "," ",proCode," "," ");
    return movie;
    }
}
```

Servlet java

```
package movies;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.List;
```

```
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import movies.moviesDTO;
import movies.moviesDAO;



@WebServlet("/movie_servlet")
public class movie_servlet extends HttpServlet {
   private static final long serialVersionUID = 1L;

      protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
       System.out.println("DAO doget()호출!");
       response.setContentType("text/html;charset=UTF-8");
       PrintWriter out = response.getWriter();
       moviesDAO dao=new moviesDAO();

       String mode = request.getParameter("mode");

       switch(mode) {
           case "Read_Movie":
               List<moviesDTO> movieList=dao.readMovie(request.getParameter("title"));
               int count=0;
               for(moviesDTO listMovie : movieList) {
                   if(count==1) {
                               out.println("<br>Actor2's name : " + listMovie.getActor_fname()+ "
"+listMovie.getActor_lname()
                                + " </h2>");
                   }
                   else {
                       out.println("<h1>Info of " + request.getParameter("title") + "</h1><br>"
                                       + "<h2> Director's name : "+listMovie.getDir_fname()+" "+
listMovie.getDir_lname() +
                                        "<br> Actor1's name : " + listMovie.getActor_fname()+" "+
listMovie.getActor_lname());
                       count+=1;
                   }
```

```
                }
                    break;
            case "Read_Actor":
                List<moviesDTO> actorList=dao.readActor(request.getParameter("title"));
                out.println("<h1>Actors in " + request.getParameter("title") + "</h1><br>");
                for(moviesDTO listMovie : actorList) {

                                out.println("<h2>Actor's name : " + listMovie.getActor_fname()+ "
"+listMovie.getActor_lname()
                                + " </h2>");
                }
                    break;
            case "Read_Producers":
                List<moviesDTO> proList=dao.readProducers(request.getParameter("title"));
                out.println("<h1>Producer(s) in " + request.getParameter("title") + "</h1><br>");
                for(moviesDTO listMovie : proList) {

                               out.println("<h2>Producer's name : " + listMovie.getPro_fname()+ "
"+listMovie.getPro_lname()
                                + " </h2>");
                }
                    break;
            case "Read_Diract":
                List<moviesDTO> diractList=dao.readDiract();
                for(moviesDTO listMovie : diractList) {

                                      out.println("<h2>Name : " + listMovie.getActor_fname()+ "
"+listMovie.getActor_lname()
                                + " </h2>");
                }
                    break;

            case "Read_Proact":
                List<moviesDTO> proactList=dao.readProact();
                for(moviesDTO listMovie : proactList) {

                                      out.println("<h2>Name : " + listMovie.getActor_fname()+ "
"+listMovie.getActor_lname()
                                + " </h2>");
                }
                    break;
            case "Create_Actor":
                                                                                moviesDTO
cactor=dao.createActor(request.getParameter("actorCode"),request.getParameter("fName")
                    , request.getParameter("lName"), request.getParameter("movieCode"));
```

```
                out.println("<h2>" + cactor.getActor_fname()+ " "+cactor.getActor_lname()
                                + " has been completely inserted to database! </h2>");


                        break;
            case "Create_Producer":
                                                                                                moviesDTO
cpro=dao.createProducer(request.getParameter("proCode"),request.getParameter("fName")
                                                                ,   request.getParameter("lName"),
request.getParameter("movieCode"),request.getParameter("actCode"));


                out.println("<h2>" + cpro.getPro_fname()+ " "+cpro.getPro_lname()
                                + " has been completely inserted to database! </h2>");


                        break;
            case "Create_Movie":
                                moviesDTO  cmovie=dao.createMovie(request.getParameter("act1Code"),
request.getParameter("act1fname"),
                               request.getParameter("act1lname"), request.getParameter("act2Code"),
request.getParameter("act2fname")
                            , request.getParameter("act2lname"), request.getParameter("movieCode"),
request.getParameter("title"),
                                request.getParameter("dirCode"), request.getParameter("dirfname"),
request.getParameter("dirlname"),
                                request.getParameter("proCode"), request.getParameter("profname"),
request.getParameter("prolname"));


                out.println("<h2>" + cmovie.getMovie_title()
                                + " has been completely inserted to database! </h2>");


                        break;
            case "Update_Diract":
                               moviesDTO  udiract=dao.updateDiract(request.getParameter("dirCode"),
request.getParameter("actCode"));


                         out.println("<h2>"  +  udiract.getActor_code()+  "  is  now  connected  with
"+udiract.getDir_code()
                                + " ! </h2>");


                        break;
            case "Update_Proact":
                               moviesDTO  uproact=dao.updateProact(request.getParameter("proCode"),
request.getParameter("actCode"));


                         out.println("<h2>"  +  uproact.getActor_code()+  "  is  now  connected  with
"+uproact.getPro_code()
                                + " ! </h2>");
```

```
                        break;
            case "Delete_Actor":
                moviesDTO dactor=dao.deleteActor(request.getParameter("actCode"));
                out.println("<h2>" + dactor.getActor_code()+ " is succesfully deleted! </h2>");


                        break;
            case "Delete_Producer":
                moviesDTO dpro=dao.deleteProducer(request.getParameter("proCode"));
                out.println("<h2>" + dpro.getPro_code()+ " is succesfully deleted! </h2>");


                        break;
            default:
                break;
        }
        out.close();
    }
}
```