

데이터베이스설계 #프로젝트

제 목

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 12일

학부 정보통신공학과

학년 3

성명 서혜빈

학번 1218178



1. **개요**

Movies database 의 주어진 ERD 를 가정에 맞게 수정한다.

Movies database 에 테이블을 만들고, Key 를 수정한 ERD 및 가정에 맞게 생성한다. 또한 이를 Web application 을 통해 CURD 가 가능하도록 구현한다.

1. **상세 개요**
   * + - 1. **ERD에 대한 답(T/F)**
2. T

Actor와 Movie의 관계가 전체참여이므로, 이 데이터베이스에는 영화에 출여하지 않은 배우가 없다.

1. Maybe

있을 수 있지만, 필수가 아니다. Actor와 Movie의 관계는 Performs\_in 이다. 해당 릴레이션은 M : N 관계로, 제한이 없다. 따라서 일부 배우들은 10편 이상의 영화에 출연했을 수 있다.

따라서 False 로 가정한다.

한 배우는 10편 이상의 영화에 출연할 수 없다. 그러므로 M : N 관계를 유지하지만, trigger를 통해 한 배우가 10편 이상의 영화에 출연하려고 하는 경우, INSERT가 불가능하도록 제한한다.

1. T

True이지만 이 또한, 2번 가정에 의해 한 배우는 10편 이상의 영화에서 주연을 맡을 수 없다.

1. T

Lead\_role 이 2 : N 관계이므로, True이다. 이는 trigger를 통해 한 영화의 주연 배우가 2명 이상 INSERT 될 경우, 제한한다.

1. F
2. F
3. F
4. Maybe

3번의 이유와 같다. Performs\_in 은 M : N 관계이고, 이는 제한이 없다.

False로, 한 영화에는 10명 이상의 배우가 출연하지 않는다고 가정한다. 이것 또한 trigger를 통해 한 영화에 10명 이상의 배우가 INSERT 될 경우, 제한한다.

1. T
2. Maybe

영화에는 감독 1명, 제작자 1명만이 있다고 가정한다. 따라서 True로 가정하는 것이다. 또한 영화에는 감독과 제작자가 무조건 있다고 가정한다.

즉, Movie와 Producer 사이의 Produce 릴레이션을 1 : 1 관계로 수정한다. 또한, 전체 참여로 수정한다.

1. T

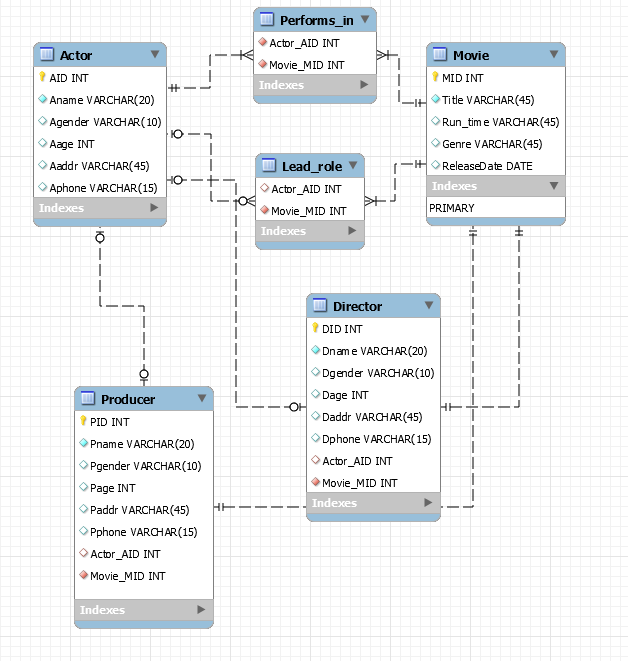
True 이지만, 10번에서 감독 1명, 제작자 1명으로 가정했기 때문에 False로 수정한다.

따라서 Director와 Movie 관계가 1 : 1이고, 전체참여로 수정한다.

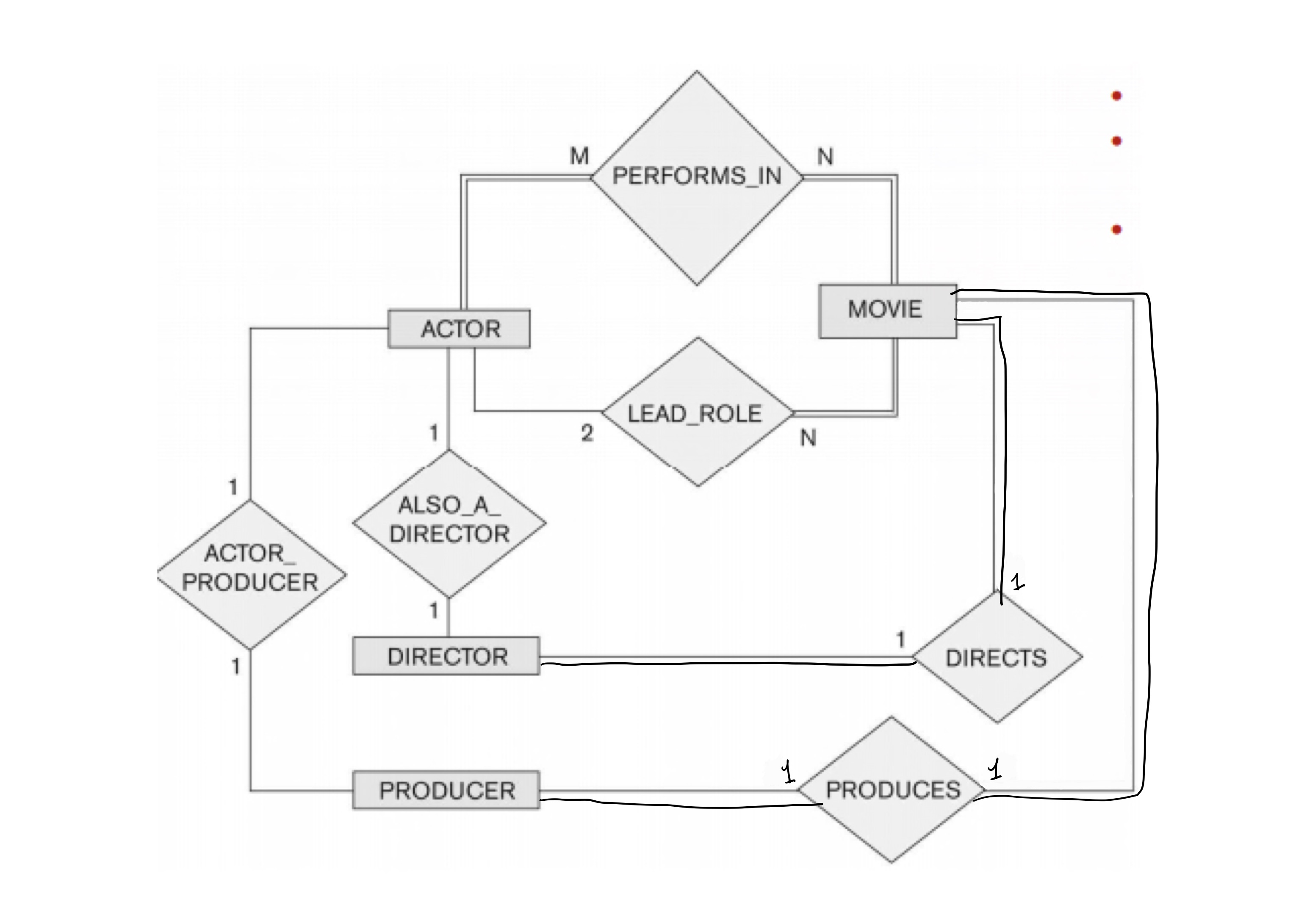
12) T

13) F

* + - * 1. **가정을 바탕으로 수정한 ERD**



* 부분참여의 경우 mandotory를 주어, 참여가 0이 가능하도록 해주었다.
* Lead\_role 릴레이션의 2 : N 관계는 우선 M : N 으로 주었고, mysql 에서 trigger로 제한할 예정이다. Performs\_in 또한 마찬가지이다.
* Identifying relation이 없기 때문에 모두 실선으로 해주었다.



* Producer 와 Movie의 관계는 전체참여이면서, 1:1이다.
* Director 와 Movie의 관계는 전체참여이면서, 1:1이다.
* Performs\_in 과 Lead\_role 은 trigger로 Insert를 제한한다.
  + - * 1. **MOVIES Database 생성**

mysql> create database movies;

Query OK, 1 row affected (0.00 sec)

mysql> create user 'movieuser'@'localhost' identified by 'movie1234';

Query OK, 0 rows affected (0.02 sec)

mysql> grant all privileges on movies.\* to movieuser@'localhost';

Query OK, 0 rows affected (0.01 sec)

mysql> flush privileges;

Query OK, 0 rows affected (0.01 sec)

mysql> set global time\_zone = '+9:00';

Query OK, 0 rows affected (0.01 sec)

mysql> set time\_zone = '+9:00';

Query OK, 0 rows affected (0.00 sec)

(더 자세한 코드는 txt 파일 참조.)

* + - * 1. **Table 생성**

mysql> CREATE TABLE Actor (AID INT NOT NULL,

-> Aname VARCHAR(20) NOT NULL,

-> Agender VARCHAR(10),

-> Aage INT,

-> Aaddr VARCHAR(45),

-> Aphone VARCHAR(15),

-> PRIMARY KEY (AID));

Query OK, 0 rows affected (0.04 sec)

mysql> CREATE TABLE Movie (MID INT NOT NULL,

-> Title VARCHAR(45) NOT NULL,

-> Run\_time VARCHAR(45),

-> Genre VARCHAR(45),

-> ReleaseDate DATE,

-> PRIMARY KEY (MID));

Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE Director (DID INT NOT NULL,

-> Dname VARCHAR(20) NOT NULL,

-> Dgender VARCHAR(10),

-> Dage INT,

-> Daddr VARCHAR(45),

-> Dphone VARCHAR(15),

-> AID INT,

-> MID INT NOT NULL,

-> PRIMARY KEY(DID),

-> FOREIGN KEY (AID) REFERENCES Actor (AID)

-> ON DELETE CASCADE

-> ON UPDATE CASCADE,

-> FOREIGN KEY (MID) REFERENCES Movie (MID)

-> ON DELETE CASCADE

-> ON UPDATE CASCADE);

Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE Producer (PID INT NOT NULL,

-> Pname VARCHAR(20) NOT NULL,

-> Pgender VARCHAR(10),

-> Page INT,

-> Paddr VARCHAR(45),

-> Pphone VARCHAR(15),

-> AID INT,

-> MID INT NOT NULL,

-> PRIMARY KEY (PID),

-> FOREIGN KEY (AID) REFERENCES Actor (AID)

-> ON DELETE CASCADE

-> ON UPDATE CASCADE,

-> FOREIGN KEY (MID) REFERENCES Movie (MID)

-> ON DELETE CASCADE

-> ON UPDATE CASCADE);

Query OK, 0 rows affected (0.04 sec)

mysql> CREATE TABLE Performs\_in (AID INT,

-> MID INT,

-> FOREIGN KEY (AID) REFERENCES Actor (AID)

-> ON DELETE CASCADE

-> ON UPDATE CASCADE,

-> FOREIGN KEY (MID) REFERENCES Movie (MID)

-> ON DELETE CASCADE

-> ON UPDATE CASCADE);

Query OK, 0 rows affected (0.16 sec)

mysql> CREATE TABLE Lead\_role (AID INT,

-> MID INT NOT NULL,

-> FOREIGN KEY (AID) REFERENCES Actor (AID)

-> ON DELETE CASCADE

-> ON UPDATE CASCADE,

-> FOREIGN KEY (MID) REFERENCES Movie (MID)

-> ON DELETE CASCADE

-> ON UPDATE CASCADE);

Query OK, 0 rows affected (0.05 sec)

* FK 에 CASCADE 조건을 추가해주었다. 삭제 및 갱신 시 관련 애트리뷰트를 모두 삭제 가능하다. 예를 들어, Actor 테이블에서 AID 가 1인 배우를 삭제할 경우 Lead\_role, Perfomrs\_in 등 AID 를 참조하는 모든 테이블에서 삭제 가능하다.

(더 자세한 코드는 txt 파일 참조.)

* + - * 1. **Index 사용**
* Index 생성 전

mysql> SELECT m.title, a.AID

-> FROM Movie as m, Actor as a, Performs\_in as pi

-> WHERE (pi.AID = a.AID) AND (pi.MID = m.MID);

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

| title | AID |

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

| Movie-1 | 1 |

| Movie-1 | 2 |

| Movie-1 | 3 |

| Movie-3 | 4 |

| Movie-1 | 5 |

| Movie-5 | 6 |

| Movie-3 | 7 |

| Movie-2 | 8 |

| Movie-2 | 9 |

| Movie-5 | 10 |

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

10 rows in set (0.01 sec)

* 영화에 출연하는 배우 id 목록을 select 해보았다. Join 결과, 시간이 0.01 sec 소요된 것을 확인할 수 있다.
* Index 생성

mysql> CREATE INDEX index\_1 ON Performs\_in (AID);

Query OK, 0 rows affected (0.05 sec)

Records: 0 Duplicates: 0 Warnings: 0

* Index 생성 후

mysql> SELECT m.title, a.AID

-> FROM Movie as m, Actor as a, Performs\_in as pi

-> WHERE (pi.AID = a.AID) AND (pi.MID = m.MID);

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

| title | AID |

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

| Movie-1 | 1 |

| Movie-1 | 2 |

| Movie-1 | 3 |

| Movie-3 | 4 |

| Movie-1 | 5 |

| Movie-5 | 6 |

| Movie-3 | 7 |

| Movie-2 | 8 |

| Movie-2 | 9 |

| Movie-5 | 10 |

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

10 rows in set (0.00 sec)

* Index 생성 후, 조인을 해보니 0.00 sec 으로 시간이 단축 된 것을 확인할 수 있다.
* Index 가 잘 실행되고 있음을 알 수 있다.

(더 자세한 코드는 txt 파일 참조.)

* + - * 1. **Join view**
* Join View 1 생성

mysql> CREATE VIEW V1

-> AS

-> SELECT m.title, a.aname, a.agender

-> FROM Lead\_role lr, Movie m, Actor a

-> WHERE (lr.AID = a.AID) AND (lr.MID = m.MID);

Query OK, 0 rows affected (0.01 sec)

mysql> SELECT \* FROM V1;

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

| title | aname | agender |

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

| Movie-1 | Actor-1 | female |

| Movie-1 | Actor-3 | male |

| Movie-2 | Actor-5 | male |

| Movie-3 | Actor-10 | female |

| Movie-3 | Actor-4 | female |

| Movie-5 | Actor-2 | female |

| Movie-5 | Actor-6 | female |

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

7 rows in set (0.00 sec)

* 영화에 참여하는 주연배우의 이름과 성별을 조인하여 SELECT 해주었다.
* Join View 2 생성

mysql> CREATE VIEW V2

-> AS

-> SELECT m.title, d.dname, p.pname

-> FROM Movie m, Director d, Producer p

-> WHERE (d.MID = m.MID) AND (p.MID = m.MID);

Query OK, 0 rows affected (0.01 sec)

mysql> SELECT \* FROM V2;

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

| title | dname | pname |

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

| Movie-1 | Director-2 | Producer-1 |

| Movie-2 | Director-5 | Producer-3 |

| Movie-3 | Director-1 | Producer-5 |

| Movie-4 | Director-4 | Producer-2 |

| Movie-5 | Director-3 | Producer-4 |

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

5 rows in set (0.00 sec)

* 영화 제작에 참여하는 제작자와 프로듀서의 이름을 조인하여 SELECT 해주었다.

(더 자세한 코드는 Txt 파일 참조.)

* + - * 1. **Trigger**
* 주연 배우 2명 제한 trigger

mysql> delimiter //

mysql> create trigger Lead\_role\_trigger

-> before insert on Lead\_role

-> for each row

-> begin

-> set @sum = (select count(\*) from Lead\_role where MID = NEW.MID);

-> if @sum > 1 then

-> SIGNAL SQLSTATE '45000'

-> SET MESSAGE\_TEXT = 'Maximum';

-> end if;

-> end; //

Query OK, 0 rows affected (0.03 sec)

* Trigger 생성

mysql> delimiter ;

mysql> SELECT \* FROM Lead\_role;

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

| AID | MID |

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

| 1 | 1 |

| 3 | 1 |

| 10 | 3 |

| 4 | 3 |

| 6 | 5 |

| 2 | 5 |

| 5 | 2 |

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

7 rows in set (0.01 sec)

mysql> INSERT INTO Lead\_role VALUES ('9', '1');

ERROR 1644 (45000): Maximum

* 2명이 이미 있는데, 또 INSERT 하려고 시도하는 경우, ERROR 발생한다.
* Trigger 가 제대로 작동하고 있다.

mysql> INSERT INTO Lead\_role VALUES ('9', '4');

Query OK, 1 row affected (0.01 sec)

* 4번 영화에는 주연 배우가 아직 없으므로 INSERT 성공한다.
* 한 영화에 참여하는 배우 10명 제한 trigger

mysql> delimiter //

mysql> create trigger Performs\_in\_AID\_trigger

-> before insert on Performs\_in

-> for each row

-> begin

-> set @sum\_aid = (select count(\*) from Performs\_in where AID = NEW.AID);

-> if @sum\_aid > 8 then

-> SIGNAL SQLSTATE '45000'

-> SET MESSAGE\_TEXT = 'Maximum';

-> end if;

-> end; //

Query OK, 0 rows affected (0.03 sec)

mysql> delimiter ;

mysql> select \* from Performs\_in;

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

| AID | MID |

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

| 1 | 1 |

| 2 | 1 |

| 3 | 1 |

| 4 | 3 |

| 5 | 1 |

| 6 | 5 |

| 7 | 3 |

| 8 | 2 |

| 9 | 2 |

| 10 | 5 |

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

10 rows in set (0.01 sec)

* 한명의 배우는 영화 10개 이상에 참여 할 수 없도록 제한 trigger

mysql> delimiter //

mysql> create trigger Performs\_in\_MID\_trigger

-> before insert on Performs\_in

-> for each row

-> begin

-> set @sum\_mid = (select count(\*) from Performs\_in where MID = NEW.MID);

-> if @sum\_mid > 8 then

-> SIGNAL SQLSTATE '45000'

-> SET MESSAGE\_TEXT = 'Maximum';

-> end if;

-> end; //

Query OK, 0 rows affected (0.02 sec)

mysql> delimiter ;

* Trigger 생성

mysql> SELECT count(AID) FROM Performs\_in WHERE MID = 1;

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

| count(AID) |

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

| 9 |

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

1. row in set (0.00 sec)

* 현재 1번 영화에 참여하는 배우 수는 9명이다.

mysql> INSERT INTO Performs\_in VALUES ('10', '1');

ERROR 1644 (45000): Maximum

* 따라서 INSERT 시 오류가 발생한다.
  + - * 1. **CRUD**
* DeleteActor.jsp
* request.setCharacterEncoding("EUC-KR");
* int aid = Integer.parseInt(request.getParameter("aid"));
* Connection conn=null;
* PreparedStatement pstmt=null;
* String sql = "DELETE FROM Actor WHERE AID = ?";
* conn=DatabaseUtil.getConnection();
* pstmt=conn.prepareStatement(sql);
* PrintWriter script=response.getWriter();
* pstmt.setInt(1, aid);
* pstmt.executeUpdate();
* pstmt.close();
* conn.close();
* 웹 페이지에서 입력받은 aid 를 sql 변수의 ‘?’ 자리에 넣어준다. aid 와 AID 가 같을 경우 해당 애트리뷰트를 삭제한다. 마지막에는 executeUpdate 로, 해당 내용을 mysql 에 갱신한다.
* Director, Movie, Producer 도 변수만 변경하여 동일하게 작성해주었다.
* InsertAction.jsp
* request.setCharacterEncoding("EUC-KR");
* int aid = Integer.parseInt(request.getParameter("aid"));
* String aname = request.getParameter("aname");
* String agender = request.getParameter("agender");
* int aage = Integer.parseInt(request.getParameter("aage"));
* String aaddr = request.getParameter("aaddr");
* String aphone = request.getParameter("aphone");
* String sql="INSERT INTO Actor VALUES (?, ?, ?, ?, ?, ?)";
* Connection conn=null;
* PreparedStatement pstmt=null;
* ResultSet rs=null;
* conn=DatabaseUtil.getConnection();
* pstmt=conn.prepareStatement(sql);
* PrintWriter script=response.getWriter();
* pstmt.setInt(1, aid);
* pstmt.setString(2, aname);
* pstmt.setString(3, agender);
* pstmt.setInt(4, aage);
* pstmt.setString(5, aaddr);
* pstmt.setString(6, aphone);
* pstmt.executeUpdate();
* pstmt.close();
* conn.close();
* Aid, aname, agender, aage, aaddr, aphone을 웹페이지에서 입력받은 뒤, sql 변수에 넣어준다. 해당 내용들이 Actor 테이블에 들어갈 수 있도록, executeUpdate를 해준다.
* Aid, aage 의 경우 request.getParameter 로 값을 가져온 후, Integer.parseInt를 통해 int 형으로 변환해준다.
* Movie도 변수만 변경하여 동일하게 작성해주었다.
* UpdateActor.jsp
* request.setCharacterEncoding("EUC-KR");
* int aid = Integer.parseInt(request.getParameter("aid"));
* String aname = request.getParameter("aname");
* Connection conn=null;
* PreparedStatement pstmt=null;
* String sql = "UPDATE Actor SET Aname = ? WHERE AID = ?";
* conn=DatabaseUtil.getConnection();
* pstmt = conn.prepareStatement(sql);
* PrintWriter script=response.getWriter();
* pstmt.setString(1, aname);
* pstmt.setInt(2, aid);
* pstmt.executeUpdate();
* pstmt.close();
* conn.close();
* 입력받은 aid 가 기존에 있던 AID 와 같을 경우, 이름을 수정할 수 있도록 작성하였다.
* Movie, Director, Producer 도 변수만 수정하여 동일하게 작성하였다.
* Select.jsp
* request.setCharacterEncoding("EUC-KR");
* String sql="select title, aname, agender from V1";
* Connection conn=null;
* PreparedStatement pstmt=null;
* ResultSet rs=null;
* conn=DatabaseUtil.getConnection();
* pstmt=conn.prepareStatement(sql);
* rs=pstmt.executeQuery();
* PrintWriter script=response.getWriter();
* while(rs.next()){
* String Title = rs.getString(1);
* String Aname = rs.getString(2);
* String Agender = rs.getString(3);
* script.println("<h3>" + "Moive title is " + Title + "</h3>"
* + "<h3>" + "Actor's name is " + Aname + "</h3>"
* + "<h3>" + "Actor's gender is " + Agender + "</h3>");
* }
* rs.close();
* pstmt.close();
* conn.close();
* 앞서 만들었던 View 중 V1 만을 SELECT 하도록 하였다. while문을 이용하여 출력을 해주고, 한 애트리뷰트를 출력하고 난 후 rs.next로 다음 애트리뷰트를 출력하도록 했다. 다음 애트리뷰트가 없을 때까지 출력한다.
* DatabaseUtil.java
* package util;
* import java.sql.Connection;
* import java.sql.DriverManager;
* public class DatabaseUtil {
* public static Connection getConnection() {
* try {
* String dbURL = "jdbc:mysql://172.17.0.2:3306/movies";
* String dbID ="root";
* String dbPassword="root";
* Class.forName("com.mysql.cj.jdbc.Driver");
* return DriverManager.getConnection(dbURL, dbID, dbPassword);
* } catch (Exception e) {
* e.printStackTrace();
* }
* return null;
* }
* }
* 데이터베이스와 연결하는 파일이다.
* Id 와 password 에 root 의 id 인 root, root 의 password 인 root를 주었다. Root 데이터베이스에 접속하여, movies database에 접속할 수 있도록 하였다.
  + - * 1. **Docker push**

C:\WINDOWS\system32>docker commit tomcat-server saved\_project

sha256:66d7e8fac24cf5ba2b9b34f86cffbe93d119b6b523b840660cff186bc367b2cc

C:\WINDOWS\system32>docker tag saved\_project idanbyi94/project\_fin

C:\WINDOWS\system32>docker push idanbyi94/project\_fin

The push refers to repository [docker.io/idanbyi94/project\_fin]

3f0ec0266c82: Pushed

e4edb7863185: Mounted from idanbyi94/12181785\_hyebeen

aa794496f8e9: Mounted from idanbyi94/12181785\_hyebeen

7a694df0ad6c: Mounted from idanbyi94/12181785\_hyebeen

3fd9df553184: Mounted from idanbyi94/12181785\_hyebeen

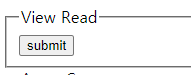
805802706667: Mounted from idanbyi94/12181785\_hyebeen

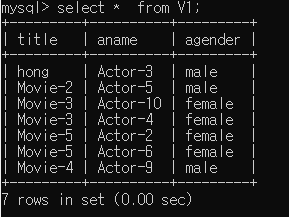
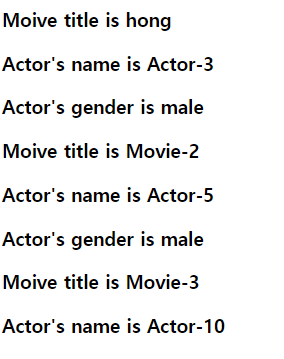
latest: digest: sha256:23100e82772c1bdd815cb6ea61d6bb8c5fa4e9c7df1cbcee1a0b08eb9cc53a4d size: 1581

* **Image 이름 : idanbyi94/project\_fin**

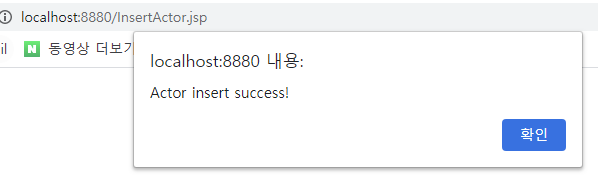
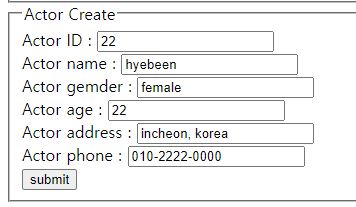
1. **실행 화면**
   * + - 1. CRUD 실행

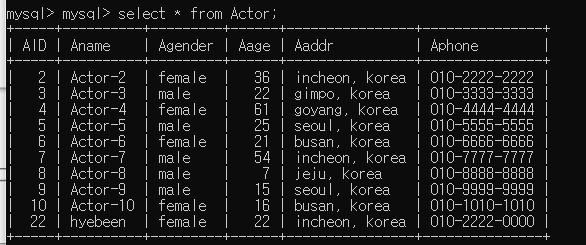
* SELECT VIEW



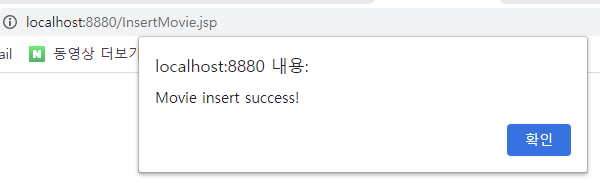
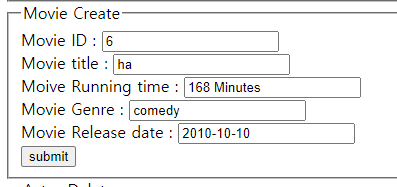


* UPDATE를 실행한 후 VIEW를 SELECT 한 것이라, Moive-1이 hong으로 바뀌었다. 처음 INSERT 하고, VIEW를 만들었을 때에는 Movie-1 이었다. UPDATE로 Movie 테이블의 title을 바꿔도 VIEW 에서도 FK로써 잘 바뀌는 것을 확인할 수 있다.
* 여러 테이블을 조인하여 만든 VIEW가 잘 SELECT 되는 것을 확인할 수 있다.
* INSERT



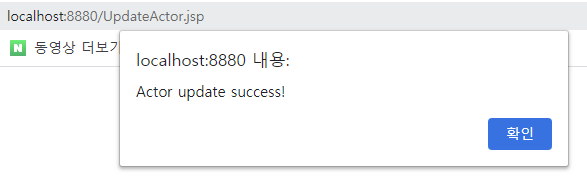
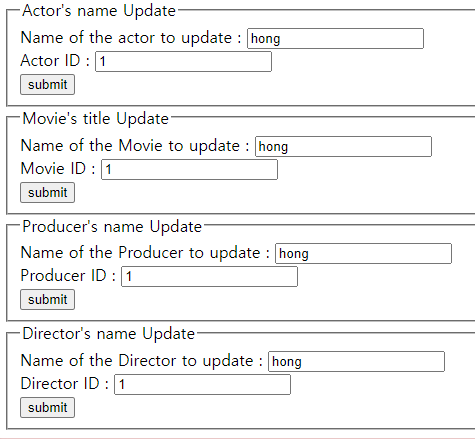


* Actor 테이블에 추가한 내용이 mysql 에서도 잘 확인이 되었다.

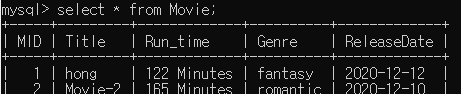




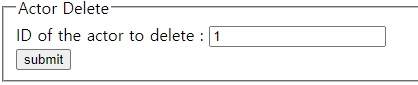
* Movie 테이블에 추가한 내용이 mysql 에서도 잘 확인이 되었다.
* INSERT 가 잘 실행됨을 확인할 수 있다.
* Update

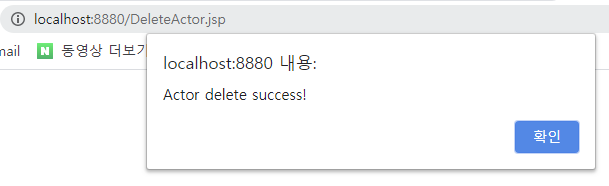


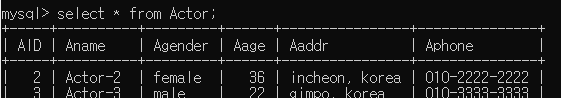




* AID, MID, PID, DID 가 1인 것들의 이름을 모두 hong으로 수정해주었다.
* Mysql 에서 확인 결과, Actor, Movie 뿐만 아니라 Producer, Director 에서도 잘 수정이 되었음을 확인할 수 있다.
* DELETE







* Actor 테이블에서 AID 가 1인 행을 삭제한다.
* Mysql 에서 확인 결과, 잘 삭제 되었다. Movie, Director, Producer 에서도 잘 삭제되는 것을 확인할 수 있다.
* index.jsp
* <%@ page language="java" contentType ="text/html; charset=EUC-KR"
* pageEncoding="EUC-KR"%>
* <!DOCTYPE html>
* <html>
* <head>
* <meta charset="Eng"/>
* <title></title>
* </head>
* <body>
* <form action="Select.jsp" method="post">
* <fieldset style = "width:150">
* <legend> View Read </legend>
* <input type="submit" value="submit">
* </fieldset>
* </form>
* </body>
* <body>
* <form action="InsertActor.jsp" method="post">
* <fieldset style = "width:150">
* <legend> Actor Create </legend>
* Actor ID : <input type = "text" name = "aid" required/><br>
* Actor name : <input type = "text" name = "aname" required/><br>
* Actor gender : <input type = "text" name = "agender"/><br>
* Actor age : <input type = "text" name = "aage"/><br>
* Actor address : <input type = "text" name = "aaddr"/><br>
* Actor phone : <input type = "text" name = "aphone"/><br>
* <input type = "submit" value = "submit"/><br>
* </fieldset>
* </form>
* </body>
* Jsp 를 통해 html 을 작성해주었다. 위에 작성된 SELECT, InsertActor 외에도 Movie Create, Directo Create 등과 Delete, Update 도 작성해주었는데, 너무 길어서 생략하였다.

1. **분석 및 결론**

우선 1명의 배우는 10편 이상의 영화에 참여할 수 없도록 하고, 1편의 영화는 10명 이상의 배우가 참여하지 못한다고 가정했다. 그리고 Director와 Movie의 관계, Producer와 Movie의 관계는 1:1, 전체참여로 가정했다. 이 관계는 Mysql 에서 trigger로 제한해 주었다.

가정을 바탕으로 Workbench를 이용하여 EERD 를 작성해주었고, 이 EERD 를 바탕으로 Mysql 에서 movies database를 생성해주었다. 각 테이블도 맞게 생성해주고, 각 키에는 가정에 맞는 올바른 PK, FK 를 주었다. FK의 경우 CASCADE를 주어 삭제 및 갱신이 용이하도록 하였다.

그리고 Index를 사용하여 조인 시, 시간이 단축되도록 해주었다. 또한 조인의 결과를 view 를 통해 확인했다.

마지막으로, Web application을 작성하였다. 웹 페이지에서 mysql 에 접근하여 정보를 CRUDE 할 수 있도록 한 것이다.