

UE21CS351A: Database Management System

MINI PROJECT USER REQUIREMENT SPECIFICATION

TOPIC: MOVIE DATABASE

CARRIED OUT BY: RAHUL C KALEKAR PES1UG21CS471

R SUMEDH PES1UG21CS465

1.Introduction

Purpose of the Project

The primary goal of this project is to create a Movie Database web application, that will allow users to explore information about movies, actors, directors, and other related content offering a comprehensive platform for movie enthusiasts. The objective is to deliver an immersive, user-friendly experience that enables users to discover, critique, and discuss their beloved films.

Scope of the Project

The Movie Database project contains the following key components:

- User Accounts
- Movie Information
- User Ratings and Reviews
- Search and Filtering
- Watchlists
- User Recommendations

2. Project Description

Project Overview

The Movie Database project aspires to construct an intuitive and feature-rich platform tailored to the needs of movie enthusiasts. It promises users the ability to create accounts, embark on comprehensive searches for movies, explore intricate details about movies, rate and review content, and interact with other users regarding their reviews. Employing contemporary design principles, the project aims to deliver a visually appealing and user-friendly interface.

Major Project Functionalities

The functionalities of the Movie Database web application are:

- **Browse and Search**
Users possess the capability to execute searches for movies, actors, directors, and genres. They can also navigate content through designated categories.

- **Detailed Movie Information**

In-depth information about movies is readily accessible, including cast and crew details, plot summaries, release dates, and more.

- **User Ratings and Reviews**

Users can express their opinions by assigning ratings to movies and composing text-based reviews. They can also view ratings and reviews contributed by fellow users.

- **Watchlists**

Users can efficiently craft and maintain watchlists to keep a close watch on their preferred movies.

- **Recommendation System**

The system will autonomously generate movie recommendations, leveraging user preferences and viewing history.

USER REQUIREMENT SPECIFICATIONS

1. **SEARCH FOR INFORMATION ABOUT A MOVIE**

Allows for searching information about a particular movie such as the name of the movie, Date of release, Director name, ratings etc.

Entities involved: Movie

2. **SEARCH FOR INFORMATION ABOUT A DIRECTOR**

Allows for searching more info about a director corresponding to a particular movie or allows for exclusively searching info about a director

Entities involved: Movie, Director

3. SEARCH FOR INFORMATION ABOUT AN ACTOR

Allows for searching more info about an actor corresponding to a particular movie or allows for exclusively searching info about an actor

Entities involved: Movie, Actor

4. SEARCH FOR INFORMATION ABOUT A WRITER

Allows for searching more info about a writer corresponding to a particular movie or allows for exclusively searching info about a writer

Entities involved: Movie, Writer

5. GO THROUGH REVIEWS OF A MOVIE

Allows for viewing different reviews corresponding to a movie

Entities involved: Movie, Ratings

6. SEARCH FOR TOP 20 MOVIES BASED ON A CERTAIN CATEGORY

Entities involved: Movies

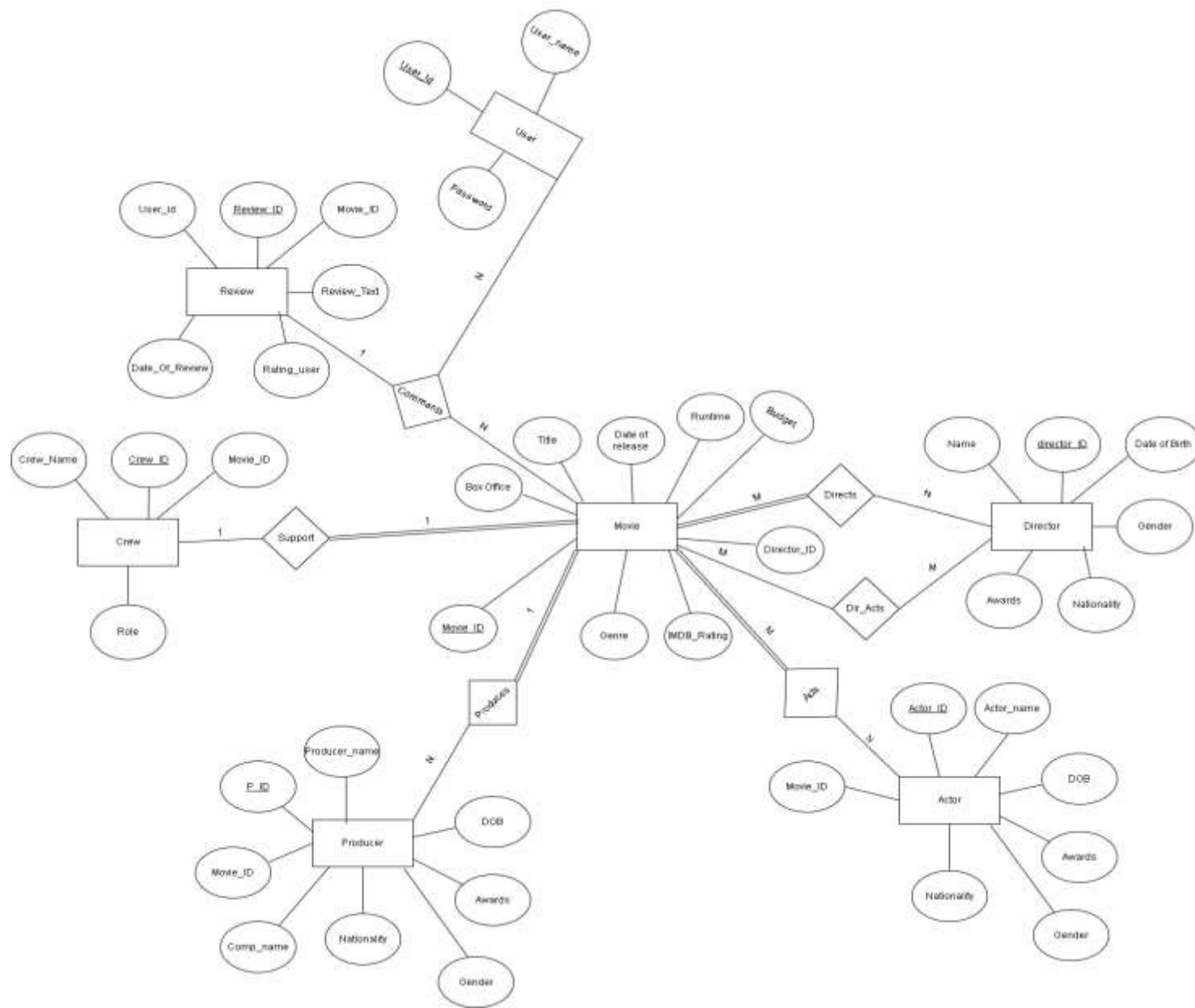
There are supposed to be several categories like Box Office collection, ratings etc. and hence a wide range of features.

7. RECOMMENDATION SYSTEM

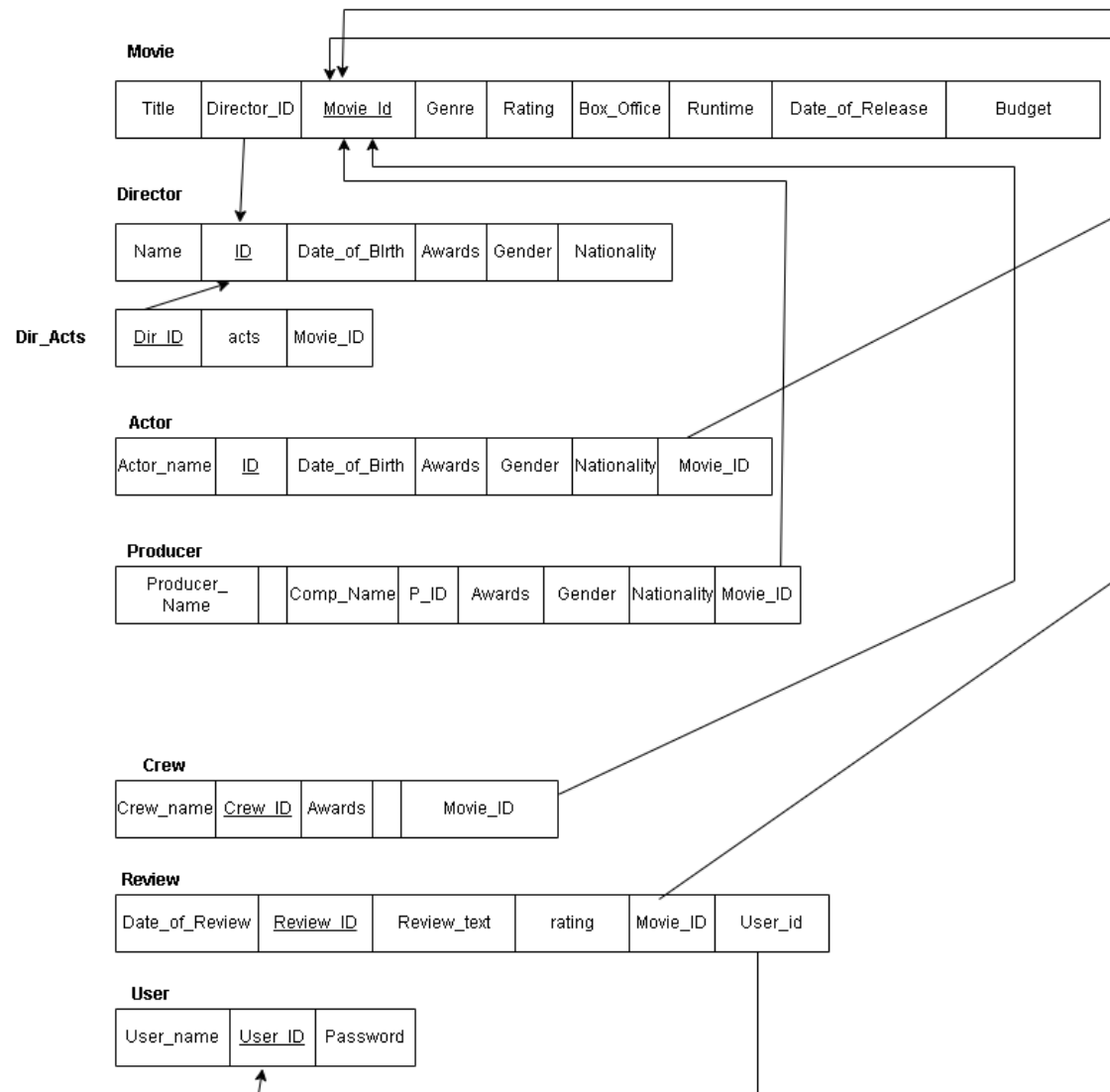
Filters out Movies based on the user interests.

Entities involved: Movies, Actors, Directors

ER DIAGRAM



RELATIONAL SCHEMA



DDL SQL COMMANDS

```
CREATE DATABASE IF NOT EXISTS `movie_database` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016
DEFAULT ENCRYPTION='N' */;
```

```
USE `movie_database`;
```

```
DROP TABLE IF EXISTS `actors`;CREATE TABLE `actors` (
  `actor_id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(255) NOT NULL,
  `date_of_birth` date NOT NULL,
  `awards` varchar(255) DEFAULT NULL,
  `gender` varchar(255) DEFAULT NULL,
  `nationality` varchar(255) DEFAULT NULL,
  `movie_id` int NOT NULL,
  PRIMARY KEY (`actor_id`),
  KEY `movie_id` (`movie_id`),
  CONSTRAINT `actors_ibfk_1` FOREIGN KEY (`movie_id`) REFERENCES `movies` (`movie_id`)
)
```

```
INSERT INTO `actors` VALUES (1,'Sam Neill','1947-09-14','0 Academy Awards','Male','New Zealander',1),(2,'John Travolta','1954-02-18','0
Academy Awards','Male','American',2),(3,'Leonardo DiCaprio','1974-11-11','1 Academy Award','Male','American',3),(4,'Saoirse Ronan','1994-
04-12','0 Academy Awards','Female','Irish',4),(5,'Song Kang-ho','1967-01-17','0 Academy Awards','Male','South Korean',5),(6,'David
Oyelowo','1976-04-01','0 Academy Awards','Male','British',6),(7,'Robert De Niro','1943-08-17','2 Academy
Awards','Male','American',7),(8,'Yalitza Aparicio','1993-12-11','0 Academy Awards','Female','Mexican',8),(9,'Matt Damon','1970-10-08','1
Academy Award','Male','American',9),(10,'Jeremy Renner','1971-01-07','0 Academy Awards','Male','American',10),(11,'Edward Norton','1969-
```

08-18','0 Academy Awards','Male','American',11),(12,'Scarlett Johansson','1984-11-22','0 Academy Awards','Female','American',12),(13,'Whoopi Goldberg','1955-11-13','1 Academy Award','Female','American',13),(14,'Suraj Sharma','1993-03-21','0 Academy Awards','Male','Indian',14),(15,'Spike Lee','1957-03-20','1 Academy Award','Male','American',15),(16,'Russell Crowe','1964-04-07','1 Academy Award','Male','New Zealander',16),(17,'Mahershala Ali','1974-02-16','2 Academy Awards','Male','American',17),(18,'Holly Hunter','1958-03-20','1 Academy Award','Female','American',18),(19,'Josh Brolin','1968-02-12','0 Academy Awards','Male','American',19),(20,'Frances McDormand','1957-06-23','2 Academy Awards','Female','American',20);

DROP TABLE IF EXISTS `crew`;

CREATE TABLE `crew` (

`crew_id` int NOT NULL AUTO_INCREMENT,

`name` varchar(255) NOT NULL,

`awards` varchar(255) DEFAULT NULL,

`movie_id` int NOT NULL,

PRIMARY KEY (`crew_id`),

KEY `movie_id` (`movie_id`),

CONSTRAINT `crew_ibfk_1` FOREIGN KEY (`movie_id`) REFERENCES `movies` (`movie_id`)

INSERT INTO `crew` VALUES (1,'Ang Lee','2 Academy Awards',13),(2,'Spike Lee','1 Academy Award',14),(3,'John Seale','1 Academy Award',15),(4,'James Laxton','0 Academy Awards',16),(5,'Janusz Kamiński','2 Academy Awards',17),(6,'Jan Chapman','0 Academy Awards',18),(7,'Ethan Coen','4 Academy Awards',19),(8,'Joel Coen','4 Academy Awards',20);

DROP TABLE IF EXISTS `dir_acts`;

CREATE TABLE `dir_acts` (


```
`director_id` int NOT NULL,  
  
`role` varchar(255) NOT NULL,  
  
PRIMARY KEY (`director_id`,`role`),  
  
CONSTRAINT `dir_acts_ibfk_1` FOREIGN KEY (`director_id`) REFERENCES `directors` (`director_id`)  
)
```

```
INSERT INTO `dir_acts` VALUES  
(1,'Director'),(2,'Director'),(3,'Director'),(4,'Director'),(5,'Director'),(6,'Director'),(7,'Director'),(8,'Director'),(9,'Director'),(10,'Director'),(11,'Director'),(12,'Director'),(13,'Director'),(14,'Director'),(15,'Director'),(16,'Director'),(17,'Director'),(18,'Director'),(19,'Director'),(20,'Director');
```

```
DROP TABLE IF EXISTS `directors`;
```

```
CREATE TABLE `directors` (  
  
  `director_id` int NOT NULL AUTO_INCREMENT,  
  
  `name` varchar(255) NOT NULL,  
  
  `date_of_birth` date NOT NULL,  
  
  `awards` varchar(255) DEFAULT NULL,  
  
  `gender` varchar(255) DEFAULT NULL,  
  
  `nationality` varchar(255) DEFAULT NULL,  
  
  PRIMARY KEY (`director_id`)  
)
```

```
INSERT INTO `directors` VALUES (1,'Steven Spielberg','1946-12-18','3 Academy Awards','Male','American'),(2,'Quentin Tarantino','1963-03-27','2 Academy Awards','Male','American'),(3,'Christopher Nolan','1970-07-30','1 Academy Award','Male','British'),(4,'Greta Gerwig','1983-08-04','1 Academy Award','Female','American'),(5,'Bong Joon-ho','1969-09-14','4 Academy Awards','Male','South Korean'),(6,'Ava DuVernay','1972-08-24','1 Academy Award','Female','American'),(7,'Martin Scorsese','1942-11-17','1 Academy Award','Male','American'),(8,'Alfonso Cuarón','1961-11-28','2 Academy Awards','Male','Mexican'),(9,'Gus Van Sant','1952-07-24','1 Academy Award','Male','American'),(10,'Kathryn Bigelow','1951-11-27','2 Academy Awards','Female','American'),(11,'David Fincher','1962-08-28','0 Academy Awards','Male','American'),(12,'Sofia Coppola','1971-05-14','1 Academy Award','Female','American'),(13,'Quincy Jones','1933-03-14','1 Academy Award','Male','American'),(14,'Ang Lee','1954-10-23','2 Academy Awards','Male','Taiwanese'),(15,'Spike Lee','1957-03-20','1 Academy Award','Male','American'),(16,'Ridley Scott','1937-11-30','1 Academy Award','Male','British'),(17,'Barry Jenkins','1979-11-19','1 Academy Award','Male','American'),(18,'Jane Campion','1954-04-30','1 Academy Award','Female','New Zealander'),(19,'Joel Coen','1954-11-29','4 Academy Awards','Male','American'),(20,'Ethan Coen','1957-09-21','4 Academy Awards','Male','American');
```

```
DROP TABLE IF EXISTS `movies`;
```

```
CREATE TABLE `movies` (
```

```
  `movie_id` int NOT NULL AUTO_INCREMENT,
```

```
  `title` varchar(255) NOT NULL,
```

```
  `release_date` date NOT NULL,
```

```
  `runtime` int NOT NULL,
```

```
  `genre` varchar(255) NOT NULL,
```

```
  `budget` int DEFAULT NULL,
```

```
  `revenue` int DEFAULT NULL,
```

```
  `rating` float NOT NULL,
```

```
  `director_id` int NOT NULL,
```

```

PRIMARY KEY (`movie_id`),

KEY `director_id` (`director_id`),

CONSTRAINT `movies_ibfk_1` FOREIGN KEY (`director_id`) REFERENCES `directors` (`director_id`)

)

```

```

INSERT INTO `movies` VALUES (1,'Jurassic Park','1993-06-11',127,'Science Fiction',63000000,1049723641,8.1,1),(2,'Pulp Fiction','1994-10-14',154,'Crime',8000000,214179088,8.9,2),(3,'Inception','2010-07-16',148,'Science Fiction',160000000,829895144,8.8,3),(4,'Lady Bird','2017-11-03',94,'Drama',10000000,79180947,7.4,4),(5,'Parasite','2019-10-11',132,'Thriller',11400000,258817992,8.6,5),(6,'Selma','2014-12-25',128,'Drama',20000000,66821036,7.5,6),(7,'The Irishman','2019-11-01',209,'Crime',159000000,113188378,7.8,7),(8,'Roma','2018-12-21',135,'Drama',15000000,20493409,7.7,8),(9,'Good Will Hunting','1997-01-09',126,'Drama',10000000,225933435,8.3,9),(10,'The Hurt Locker','2008-09-04',131,'War',15000000,49623450,7.5,10),(11,'Fight Club','1999-10-15',139,'Drama',63000000,100853753,8.8,11),(12,'Lost in Translation','2003-09-12',102,'Drama',4000000,119723856,7.7,12),(13,'The Color Purple','1985-02-07',154,'Drama',15000000,14629229,7.8,13),(14,'Life of Pi','2012-11-21',127,'Adventure',120000000,609016565,7.9,14),(15,'Do the Right Thing','1989-06-30',120,'Drama',6000000,37536959,7.9,15),(16,'Gladiator','2000-05-05',155,'Action',103000000,457640427,8.5,16),(17,'Moonlight','2016-10-21',111,'Drama',1500000,65245512,7.4,17),(18,'The Piano','1993-05-15',121,'Drama',7000000,14009306,7.6,18),(19,'No Country for Old Men','2007-11-21',122,'Crime',25000000,171627166,8.1,19),(20,' Fargo','1996-03-08',98,'Crime',7000000,60586611,8.1,20);

```

```

DROP TABLE IF EXISTS `producers`;

CREATE TABLE `producers` (

`producer_id` int NOT NULL AUTO_INCREMENT,

`name` varchar(255) NOT NULL,

`date_of_birth` date NOT NULL,

`awards` varchar(255) DEFAULT NULL,

`gender` varchar(255) DEFAULT NULL,

`nationality` varchar(255) DEFAULT NULL,

```

```

`movie_id` int NOT NULL,

PRIMARY KEY (`producer_id`),

KEY `movie_id` (`movie_id`),

CONSTRAINT `producers_ibfk_1` FOREIGN KEY (`movie_id`) REFERENCES `movies` (`movie_id`)

)

```

```

INSERT INTO `producers` VALUES (1,'Kathleen Kennedy','1953-06-05','2 Academy Awards','Female','American',1),(2,'Lawrence Bender','1957-10-17','0 Academy Awards','Male','American',2),(3,'Emma Thomas','1971-12-14','1 Academy Award','Female','British',3),(4,'Eli Bush','1984-06-02','0 Academy Awards','Male','American',4),(5,'Bong Joon-ho','1969-09-14','4 Academy Awards','Male','South Korean',5),(6,'Ava DuVernay','1972-08-24','1 Academy Award','Female','American',6),(7,'Martin Scorsese','1942-11-17','1 Academy Award','Male','American',7),(8,'Alfonso Cuarón','1961-11-28','2 Academy Awards','Male','Mexican',8),(9,'Gus Van Sant','1952-07-24','1 Academy Award','Male','American',9),(10,'Kathryn Bigelow','1951-11-27','2 Academy Awards','Female','American',10),(11,'David Fincher','1962-08-28','0 Academy Awards','Male','American',11),(12,'Sofia Coppola','1971-05-14','1 Academy Award','Female','American',12),(13,'Quincy Jones','1933-03-14','1 Academy Award','Male','American',13),(14,'Ang Lee','1954-10-23','2 Academy Awards','Male','Taiwanese',14),(15,'Spike Lee','1957-03-20','1 Academy Award','Male','American',15),(16,'Ridley Scott','1937-11-30','1 Academy Award','Male','British',16),(17,'Barry Jenkins','1979-11-19','1 Academy Award','Male','American',17),(18,'Jane Campion','1954-04-30','1 Academy Award','Female','New Zealander',18),(19,'Joel Coen','1954-11-29','4 Academy Awards','Male','American',19),(20,'Ethan Coen','1957-09-21','4 Academy Awards','Male','American',20);

```

```

DROP TABLE IF EXISTS `reviews`;

```

```

CREATE TABLE `reviews` (

`review_id` int NOT NULL AUTO_INCREMENT,

`movie_id` int NOT NULL,

`user_id` int NOT NULL,

`rating` float NOT NULL,

```

```

`review_text` text,

`date_of_review` date NOT NULL,

PRIMARY KEY (`review_id`),

KEY `movie_id` (`movie_id`),

KEY `user_id` (`user_id`),

CONSTRAINT `reviews_ibfk_1` FOREIGN KEY (`movie_id`) REFERENCES `movies` (`movie_id`),

CONSTRAINT `reviews_ibfk_2` FOREIGN KEY (`user_id`) REFERENCES `users` (`user_id`)

)

```

```

INSERT INTO `reviews` VALUES (21,1,1,9,'Classic Spielberg film!','2023-10-15'),(22,2,2,8.5,'Tarantino is a masterpiece!','2023-10-20'),(23,3,3,9.2,'Mind bending Nolan flick!','2023-10-25'),(24,4,4,7.8,'Touching coming of age story.','2023-10-30'),(25,5,5,8.6,'Parasite is a must watch ','2023-11-05'),(26,6,1,7.4,'Powerful civil rights drama.','2023-11-10'),(27,7,2,7.8,'Epic mobster saga','2023-11-15'),(28,8,3,7.7,'Roma is a visual beauty!','2023-11-20'),(29,9,4,8.3,'A math genius story.','2023-11-25'),(30,10,5,7.5,'Intense war drama.','2023-11-30'),(31,11,1,8.8,'Fight Club is mind-blowing!','2023-12-05'),(32,12,2,7.7,'Lost in Translation','2023-12-10'),(33,13,3,7.8,'The Color Purple','2023-12-15'),(34,14,4,7.9,'Life of Pi adventure!','2023-12-20'),(35,15,5,7.9,'Do the Right Thing','2023-12-25'),(36,16,1,8.5,'Gladiator is epic!','2023-12-30'),(37,17,2,7.4,'Moonlight is powerful.','2024-01-05'),(38,18,3,7.6,'The Piano is a classic.','2024-01-10'),(39,19,4,8.1,'No Country for Old Men','2024-01-15'),(40,20,5,8.1,' Fargo is a dark gem.','2024-01-20');

```

```

DROP TABLE IF EXISTS `users`;

```

```

CREATE TABLE `users` (

`user_id` int NOT NULL AUTO_INCREMENT,

`name` varchar(255) NOT NULL,

`password` varchar(255) NOT NULL,

```

PRIMARY KEY (`user_id`)

)

INSERT INTO `users` VALUES

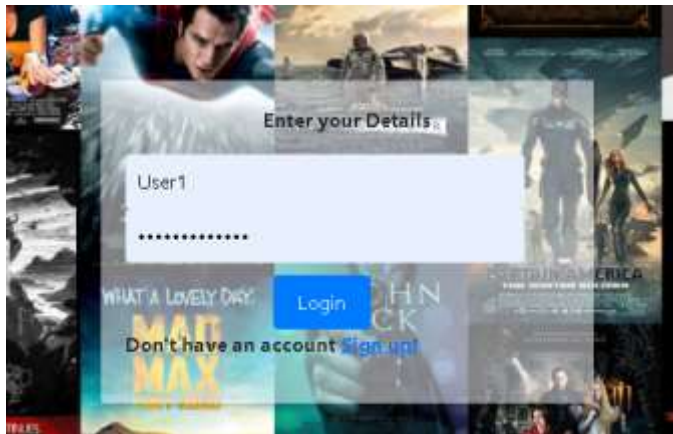
(1,'User1','user1password'),(2,'User2','user2password'),(3,'User3','user3password'),(4,'User4','user4password'),(5,'User5','user5password');

QUERIES AND SCREENSHOTS OF OUTPUT:

1. FOR USER LOGIN:

```
app.post('/api/submitFormData/', (req, res) => {  
  const { username, password } = req.body;  
  var query = connection.query('call user_login("${username}", "${password}")', (error, results) => {  
    if (error) {  
      console.error('Error executing MySQL query:', error);  
      res.status(500).json({ message: 'Server Error' });  
    } else {  
      res.json({ message: 'User added successfully' });  
    }  
  });  
});
```

OUTPUT:

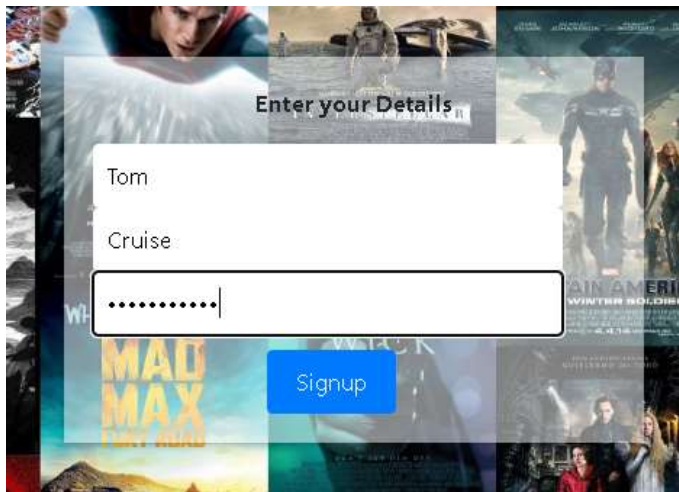


```
mysql> select * from users;  
+-----+-----+-----+  
| user_id | user_name | password |  
+-----+-----+-----+  
| 1 | User1 | user1password |  
| 2 | User2 | user2password |  
| 3 | User3 | user3password |  
| 4 | User4 | user4password |  
| 5 | User5 | user5password |
```

2. FOR USER SIGNUP:

```
app.post('/api/submitSignupData/', (req, res) => {  
  
  const { fname, lname, password } = req.body;  
  
  var query = connection.query('INSERT INTO users(user_name, password) VALUES(?,?)', [fname + " " + lname, password],  
    if (error) {  
      console.error('Error executing MySQL query:', error);  
      res.status(500).json({ message: 'Server Error' });  
    } else {  
      res.json({ message: 'User added successfully' });  
    }  
  );  
});
```

OUTPUT:



```
mysql> select * from users;  
+-----+-----+-----+  
| user_id | user_name | password |  
+-----+-----+-----+  
| 1 | User1 | user1password |  
| 2 | User2 | user2password |  
| 3 | User3 | user3password |  
| 4 | User4 | user4password |  
| 5 | User5 | user5password |  
| 18 | User5 | user5password |  
| 19 | User5 | user5password |  
| 20 | User5 | user5password |  
| 21 | R Sumedhello | mf |  
| 22 | hello | mf |  
| 23 | R.sumedh | uchiha |  
| 24 | SumedhUzumaki | naruto |  
| 25 | NarutoUzumaki | hokage |  
| 26 | ViratKohli | anushka |  
| 27 | SasukeUchiha | Itachi |  
| 28 | ReehanShaveez | reehan129 |  
| 29 | Tom Cruise | Iamthegreat |  
+-----+-----+-----+  
17 rows in set (0.00 sec)
```


3. VIEWING MOVIE DETAILS

```
app.get('/api/movie/:id', (req, res) => {
  const { id } = req.params;
  const query = `CALL GetMovieDetails("${id}")`;
  connection.query(query, (error, result) => {
    if (error) {
      console.error('Error executing MySQL query:', error);
    } else {
      // res.render('<h1>My name is Khan</h1>')
      res.render('moviedetails', { movie: result[0] });
      console.log(result)
    }
  });
});
```

```
DELIMITER //

CREATE PROCEDURE GetMovieDetails(IN id VARCHAR(255))
BEGIN
  SELECT *
  FROM movies AS m
  JOIN directors AS d ON m.director_id = d.director_id
  JOIN actors AS a ON a.movie_id = m.movie_id
  JOIN producers AS p ON p.movie_id = m.movie_id
  WHERE m.title = id;
END //

DELIMITER ;
```

OUTPUT:

FOR EG: FOR id = 'Inception':

Movie Details

Title: Inception

Genre: Science Fiction

Release Date: Fri Jul 16 2010 00:00:00 GMT+0530 (India Standard Time)

Runtime: 148 minutes

Budget: 160000000

Revenue: 829895144

Rating: 9.2

Director: [Christopher Nolan](#)

Actors:

[Leonardo DiCaprio](#)

Producer: [Emma Thomas](#)

Reviews Given By: User3: Mind bending Nolan flick!

Crew: [Christa](#)

4. For displaying the details of a director:

Query:

```
app.get('/api/director/:id', (req, res) => {  
  const { id } = req.params;  
  const query = `SELECT * FROM DIRECTORS WHERE name = "${id}"`  
  connection.query(query, (error, result) => {  
    if (error) {  
      console.error('Error executing MySQL query:', error);  
      res.status(500).json({ message: 'Server Error' });  
    } else {  
      res.render('directordetails', { director: result[0] });  
    }  
  })  
})
```

OUTPUT: FOR id = "Quentin Tarantino"

Director Details

Name: Quentin Tarantino

DOB: Wed Mar 27 1963 00:00:00 GMT+0530 (India Standard Time)

Awards: 2 Academy Awards

Gender: Male

Nationality: American

5. For displaying details about actor:

```
app.get('/api/actor/:id', (req, res) => {  
  const { id } = req.params;  
  const query = `SELECT * FROM ACTORS WHERE actor_name = "${id}"`  
  connection.query(query, (error, result) => {  
    if (error) {  
      console.error('Error executing MySQL query:', error);  
      res.status(500).json({ message: 'Server Error' });  
    } else {  
      res.render('actordetails', { actor: result[0] });  
    }  
  })  
})
```

Output: For id = "John Travolta"

Actor Details

Name: John Travolta

DOB: Thu Feb 18 1954 00:00:00 GMT+0530 (India Standard Time)

Awards: 0 Academy Awards

Gender: Male

Nationality: American

6. FOR DISPLAYING DETAILS ABOUT PRODUCER:

```
app.get('/api/producer/:id', (req, res) => {  
  const { id } = req.params;  
  const query = `SELECT * FROM PRODUCERS WHERE producer_name = "${id}"`  
  connection.query(query, (error, result) => {  
    if (error) {  
      console.error('Error executing MySQL query:', error);  
      res.status(500).json({ message: 'Server Error' });  
    } else {  
      res.render('producerdetails', { producer: result[0] });  
    }  
  })  
})
```

Output: For id = "Lawrence Bender"

Producer Details

Name: Lawrence Bender

DOB: Thu Oct 17 1957 00:00:00 GMT+0530 (India Standard Time)

Awards: 0 Academy Awards

Gender: Male

Nationality: American

7. For displaying details about crew

```
app.get('/api/crew/:id', (req, res) => {  
  const { id } = req.params;  
  const query = `SELECT * FROM CREW WHERE crew_name = "${id}"`;   
  connection.query(query, (error, result) => {  
    if (error) {  
      console.error('Error executing MySQL query:', error);  
    } else {  
      // res.render('<h1>My name is Khan</h1>')  
      res.render('crewdetails', { crew: result[0] });  
      console.log(result)  
    }  
  });  
});
```

Output: for id = "Adam"

Crew Details

Name: Adam

Awards: 2 Academy Awards

8. For displaying the TOP 10 Movies in our database based on rating:

```
app.get('/api/options/:id', (req, res) => {
  const { id } = req.params;
  console.log(id);

  if (id == "TOP 10 MOVIES") {
    const query = 'SELECT * from movies order by rating desc limit 10';
    // const query = `SELECT * FROM movies as m,directors as d,actors as a, producers as p WHERE
    connection.query(query, (error, result) => {
      if (error) {
        console.error('Error executing MySQL query:', error);
        res.status(500).json({ message: 'Server Error' });
      } else {
        // res.render('<h1>My name is Khan</h1>')
        res.render('option1details', { option1: result });
      }
    });
  }
});
```

Output:

TOP 10 MOVIES

Title: Pulp Fiction

Genre: Crime

Release_date: Fri Oct 14 1994 00:00:00 GMT+0530 (India Standard Time)

Runtime: 154 minutes

Budget: 8000000

Revenue: 214179088

Title: Inception

Genre: Science Fiction

Release_date: Fri Jul 16 2010 00:00:00 GMT+0530 (India Standard Time)

Runtime: 148 minutes

Budget: 160000000

Revenue: 829895144

Title: Fight Club

Genre: Drama

9. For searching information about a particular actor and director via a search bar

```
else if (id[id.length - 1] == "2") {
  const director = id.substring(0, id.length - 1);

  const query = `SELECT * from directors as d where d.name = "${director}"`;
  connection.query(query, (error, result) => {
    if (error) {
      console.error('Error executing MySQL query:', error);
      res.status(500).json({ message: 'Server Error' });
    } else {
      res.render('directordetails', { director: result[0] });
    }
  });
}
else if (id[id.length - 1] == "1") {
  const actor = id.substring(0, id.length - 1);
  const query = `SELECT * from actors where actors.actor_name = "${actor}"`;
  connection.query(query, (error, result) => {
    if (error) {
      console.error('Error executing MySQL query:', error);
      res.status(500).json({ message: 'Server Error' });
    } else {
      // res.render('<h1>My name is Khan</h1>')
      res.render('actordetails', { actor: result[0] });
    }
  });
}
```

OUTPUT: For actor = “Suraj Sharma” and for director = “Christopher Nolan”

Actor Details

Name: Suraj Sharma

DOB: Sun Mar 21 1993 00:00:00 GMT+0530 (India Standard Time)

Awards: 0 Academy Awards

Gender: Male

Nationality: Indian

Director Details

Name: Christopher Nolan

DOB: Thu Jul 30 1970 00:00:00 GMT+0530 (India Standard Time)

Awards: 1 Academy Award

Gender: Male

Nationality: British

10. For getting the total collection that a director has gained from his movies

```
app.get('/api/aggregate/:id', (req, res) => {  
  const { id } = req.params;  
  const query = `with dir_mov(name, title, revenue) as  
  .....(select name, title, revenue from movies  
  .....inner join directors on movies.director_id = directors.director_id)  
  .....select name, sum(revenue) as tot_revenue from dir_mov where name = "${id}"  
  .....group by name;`  
  connection.query(query, (error, result) => {  
    if (error) {  
      console.error('Error executing MySQL query:', error);  
      res.status(500).json({ message: 'Server Error' });  
    } else {  
      res.render('revenue', { revenue: result[0] });  
    }  
  })  
})
```

OUTPUT: for id = "Christopher Nolan"

Aggregate Details

Total collection of

Christopher Nolan

from his Movies: 2511159033

11. Update related queries

Updating details of movie and director:

```
app.post('/api/updateData/', (req, res) => {
  const { moviename, producename, actorname, directorname, replmoviename, replactorname, repldirectorname } = req.body;
  console.log(directorname);
  console.log(moviename);
  if (moviename !== '' && directorname !== '') {
    const query = `UPDATE movies AS m
    JOIN directors AS d ON m.director_id = d.director_id
    SET
      m.title = "${replmoviename}",
      d.name = "${repldirectorname}"
    WHERE
      m.title = "${moviename}"
      AND d.name = "${directorname}";`
  }
});
```

OUTPUT : Moviename = Alien ,replmoviename = 'ALIEN', directorname = 'Ridley Scott', repldirectorname = 'RIDLEY SCOTT'

Before and after Update

13	Quincy Jones
14	Ang Lee
15	Spike Lee
16	Ridley Scott
17	Barry Jenkins
18	Jane Campion
19	Joel Coen
14	Ang Lee
15	Spike Lee
16	RIDLEY SCOTT
17	Barry Jenkins
18	Jane Campion
19	Joel Coen

32	Hulk
33	Blade Runner
34	The Martian
35	Alien
31	Brokeback Mountain
32	Hulk
33	Blade Runner
34	The Martian
35	ALIEN

Updating details of director only:

```
else if (directorname != '') {
  const query = `update directors set name = "${repldirectorname}" where name = "${directorname}"`;
  connection.query(query, (error, result) => {
    if (error) {
      console.error('Error executing MySQL query:', error);
      res.status(500).json({ message: 'Server Error' });
    } else {
      res.json({ message: 'Updated successfully' });
    }
  });
}
```

BEFORE AND AFTER UPDATE:

13	Quincy Jones
14	Ang Lee
15	Spike Lee
16	Ridley Scott
17	Barry Jenkins
18	Jane Campion
19	Joel Coen

14	Ang Lee
15	Spike Lee
16	RIDLEY SCOTT
17	Barry Jenkins
18	Jane Campion
19	Joel Coen

Updating details of actor:

```
else if (actorname != '') {
  const query = `update actors set actor_name = "${replactorname}" where actor_name = "${actorname}"`;
  connection.query(query, (error, result) => {
    if (error) {
      console.error('Error executing MySQL query:', error);
      res.status(500).json({ message: 'Server Error' });
    } else {
      res.json({ message: 'Updated successfully' });
    }
  });
}
```

BEFORE AND AFTER UPDATE:

17	Mahershala Ali
18	Holly Hunter
19	Josh Brolin
23	Sumedh

18	Russell Crowe
17	Mahershala Ali
18	Holly Hunter
19	Josh Brolin
23	Emma Watson

12. Delete related queries:

```
app.post('/api/deleteData/', (req, res) => {
  const { moviename, producername, actorname, directorname, replmoviename, replactorname, repldirectorname } = req.body;
  if (moviename !== '') {
    const query = `call DeleteMovieAndAssociatedData("${moviename}")`;

    connection.query(query, (error, result) => {
      if (error) {
        console.error('Error executing MySQL query:', error);
        res.status(500).json({ message: 'Server Error' });
      } else {
        res.json({ message: 'Updated successfully' });
      }
    });
  }
});
```

```
root@localhost:~# mysql -u root -p
mysql> USE database;
mysql> DELIMITER //
mysql> CREATE PROCEDURE DeleteMovieAndAssociatedData (IN moviename VARCHAR(255))
BEGIN
  DECLARE movieid INT;
  SELECT movie_id INTO movieid FROM movies WHERE title = moviename;
  DELETE FROM actors WHERE movie_id = movieid;
  DELETE FROM producers WHERE movie_id = movieid;
  DELETE FROM reviews WHERE movie_id = movieid;
  DELETE FROM crew WHERE movie_id = movieid;
  DELETE FROM movies WHERE title = moviename;
END //
mysql> DELIMITER ;
```

```
else if (directorname != '') {
  const query = `call DeleteDirectorAndMovies("${directorname}")`;

  connection.query(query, (error, result) => {
    if (error) {
      console.error('Error executing MySQL query:', error);
      res.status(500).json({ message: 'Server Error' });
    } else {
      res.json({ message: 'Updated successfully' });
    }
  });
}

else if (actorname != '') {
  const query = `delete from actors where actor_name = "${actorname}"`;

  connection.query(query, (error, result) => {
    if (error) {
      console.error('Error executing MySQL query:', error);
      res.status(500).json({ message: 'Server Error' });
    } else {
      res.json({ message: 'Updated successfully' });
    }
  });
}
```



```

BEGIN
  DECLARE dir_id INT;

  SELECT director_id INTO dir_id FROM directors WHERE name = directorName;

  DELETE FROM dir_acts WHERE director_id = dir_id;

  DELETE FROM reviews WHERE movie_id IN (SELECT movie_id FROM movies WHERE director_id = dir_id);

  DELETE FROM producers WHERE movie_id IN (SELECT movie_id FROM movies WHERE director_id = dir_id);

  DELETE FROM crew WHERE movie_id IN (SELECT movie_id FROM movies WHERE director_id = dir_id);

  DELETE FROM actors WHERE movie_id IN (SELECT movie_id FROM movies WHERE director_id = dir_id);

  DELETE FROM movies WHERE director_id = dir_id;

  DELETE FROM directors WHERE director_id = dir_id;

```

BEFORE AND AFTER MOVIE DELETE

20	BlackKlansman
30	Crouching Tiger, Hidden Dragon
21	Brokeback Mountain
32	Hulk
23	Blade Runner
24	The Martian
25	Alien

20	BlackKlansman
30	Crouching Tiger, Hidden Dragon
21	Brokeback Mountain
32	Hulk
23	Blade Runner
24	The Martian

BEFORE AND AFTER DIRECTOR DELETE

15	Spike Lee
16	Ridley Scott
17	Barry Jenkins
18	Jane Campion
19	Joel Coen

15	Spike Lee
16	Ridley Scott
17	Barry Jenkins
19	Joel Coen

BEFORE AND AFTER ACTOR DELETE

15	Spike Lee
16	Russell Crowe
17	Mahershala Ali
19	Josh Brolin

14	Suraj Sharma
15	Spike Lee
16	Russell Crowe
17	Mahershala Ali

PROCEDURES AND THE CODE SNIPPETS FOR INVOKING THEM

1. User_login(user_name, password)

```
CREATE PROCEDURE `user_login`(IN username varchar(255), IN password varchar(255))
BEGIN
    DECLARE count int;
    SELECT COUNT(*) FROM USERS WHERE EXISTS(SELECT user_name, password from users where user_name = username and
password = password) into count;
    IF count = 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid username or password';
    END IF;
END
```

2. GetMovieDetails(movie_id)

```
PROCEDURE `GetMovieDetails`(IN id VARCHAR(255))
BEGIN
    SELECT *
    FROM movies AS m
    JOIN directors AS d ON m.director_id = d.director_id
    JOIN actors AS a ON a.movie_id = m.movie_id
    JOIN producers AS p ON p.movie_id = m.movie_id
    JOIN reviews AS r ON r.movie_id = m.movie_id
    JOIN crew AS c ON c.movie_id = m.movie_id
    JOIN users AS u on u.user_id = r.user_id
    WHERE title = id;
END
```

3. DeleteMovieandAssociatedData

```
PROCEDURE `DeleteMovieAndAssociatedData`(IN moviename VARCHAR(255))
```

```
BEGIN
```

```
    DECLARE movieid INT;
```

```
    SELECT movie_id INTO movieid FROM movies WHERE title = moviename;
```

```
    DELETE FROM actors WHERE movie_id = movieid;
```

```
    DELETE FROM producers WHERE movie_id = movieid;
```

```
    DELETE FROM reviews WHERE movie_id = movieid;
```

```
    DELETE FROM crew WHERE movie_id = movieid;
```

```
    DELETE FROM movies where title = moviename;
```

```
END
```

4. DeleteDirectorAndMovie

```
BEGIN
```

```
    DECLARE dir_id INT;
```

```
    SELECT director_id INTO dir_id FROM directors WHERE name = directorName;
```

```
    DELETE FROM dir_acts WHERE director_id = dir_id;
```

```
    DELETE FROM reviews WHERE movie_id IN (SELECT movie_id FROM movies WHERE director_id = dir_id);
```

```
    DELETE FROM producers WHERE movie_id IN (SELECT movie_id FROM movies WHERE director_id = dir_id);
```

```
    DELETE FROM crew WHERE movie_id IN (SELECT movie_id FROM movies WHERE director_id = dir_id);
```

```
    DELETE FROM actors WHERE movie_id IN (SELECT movie_id FROM movies WHERE director_id = dir_id);
```

```
    DELETE FROM movies WHERE director_id = dir_id;
```

```
    DELETE FROM directors WHERE director_id = dir_id;
```

Triggers:

1. Check_login(user_name)

```
check_login      | INSERT | users  | begin
```

```
DECLARE user_exists INT;
```

```
SELECT COUNT(*) INTO user_exists FROM users WHERE user_name = NEW.user_name;
```

```
IF user_exists > 0 THEN
```

```
SELECT COUNT(*) INTO user_exists FROM users WHERE user_name = NEW.user_name AND password = NEW.password;
```

```
IF user_exists = 0 THEN
```

```
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Invalid username or password';
```

```
END IF;
```

```
END IF;
```

```
END
```