Assignment 3

Project name: IMDUH

by Filip Mylläri and Fredrik Strömbäck

Idea

Our idea that we came up with is a site with users, movies and grade on the movies. The reasoning behind us choosing this project is to create a simple way for people to give their reviews of a movie. It could make it easier for users who are choosing between different movies to make a choice based on the reviews from other users. The main users for this application would be anyone who are movie interested or someone who just wants to find a movie with good reviews. Our idea solves this by letting the everyday user do a review on the movie which could help out others when it comes to choosing which movie to watch next.

Features

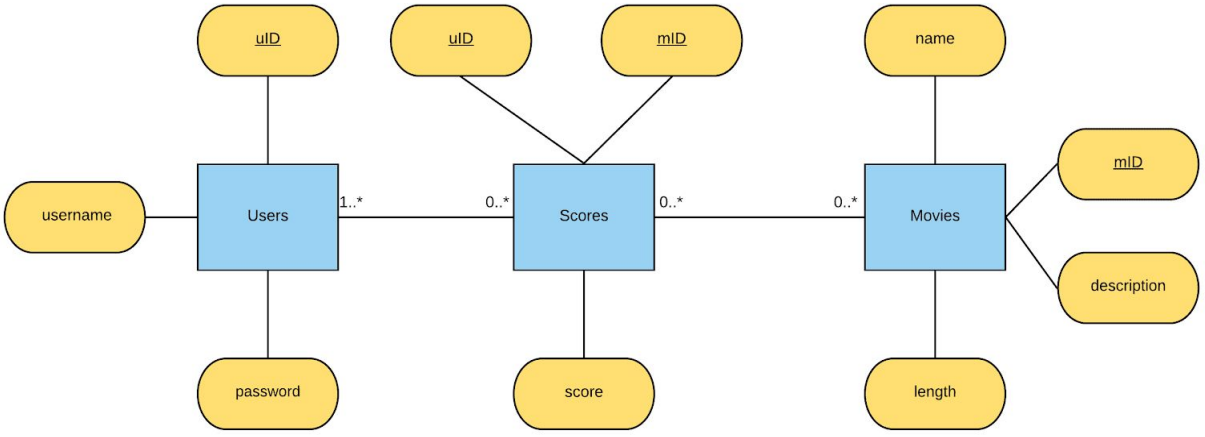
- Login system
- Grade a movie
- Be able to see grades of movies
- See the movies the user has graded and the score.
- Hours of movies watched per user

Logical model

Motivation/explanation of design:

This is a simple E/R diagram over a database that stores users, scores and movies. We have chosen to user userID as the primary key for users and movieID as the primary key for movies. In scores make use of both the userID and movieID to create the primary key. Users has a one to many relationship to score since there has to be a user to be able to view the score for that user. Score has a many to many relationship to users. Movie to score is a many to many relationship and the other way around as well. The user table contains of the users and matching passwords we have created for our application. Movies contain all the different movies we have added, information about the movies like description and length. In the score table we make use of both the movies table and users table to calculate the score and average score of a certain movie.

E/R diagram



Design in SQL

Movies: movieID, name, year, length, description
Movies has a name, length, year and description. Year is added since there could be multiple movies with the same name so the year gives a easier way to separate them. Each movie also has a movieID which also is the primary key.

```
CREATE TABLE `movies` (
  `movieID` int NOT NULL AUTO_INCREMENT,
  `name` varchar(100) NOT NULL,
  `year` int NOT NULL,
  `description` text,
  `length` int NOT NULL,
  PRIMARY KEY (`movieID`),
  UNIQUE KEY `movieID_UNIQUE` (`movieID`)
) ENGINE=InnoDB AUTO_INCREMENT=9 DEFAULT
CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Scores: movieID, userID, score
Score has both the movieID and userID to be able to calculate the score/grade of the different movies. Both the movieID and userID is the primary key in this case.

```
CREATE TABLE `scores` (
  `mID` int NOT NULL,
  `uID` int NOT NULL,
  `score` int NOT NULL,
  PRIMARY KEY (`mID`,`uID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci
```

Users: userID, username, password
The user need to have a username and password to be able to register. The users primary key is the userID.

```
CREATE TABLE `users` (
  `userID` int NOT NULL AUTO_INCREMENT,
  `username` varchar(45) NOT NULL,
  `password` varchar(255) NOT NULL,
  PRIMARY KEY (`userID`,`username`),
  UNIQUE KEY `username_UNIQUE` (`username`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT
CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

SQL queries

Query 1

This query is used in our login system. Checks if there is a user with a specific username. The query picks out the user with the specified username from the table users.

```
SELECT * FROM users WHERE username = ?
```

Query 2

This query is used to find out which movies the logged in user has voted for. The query JOINS the movies and scores tables where the movieID and mID matches AND where the uID matches the logged in userID.

```
SELECT scores.mID, scores.score, movies.name
FROM movies
JOIN scores
ON movies.movieID = scores.mID
AND scores.uID=?;
```

Query 3

This query is used create a view for keeping track on the average score of a specific movie. First of all we create a VIEW called average_score with the columns name, average and movieID, that are grouped by their name and movieID. The data picked from a JOIN of the scores and movies tables where the movieID and mID matches.

```
CREATE VIEW average_score AS SELECT name,
ROUND(AVG(T.score), 1) AS average, movieID
FROM (SELECT *
FROM imduh.scores
JOIN imduh.movies
ON imduh.movies.movieID = imduh.scores.mID) AS T
GROUP BY name, movieID;
```

Query 4

This query is used to find a specific movie from a movieID. This is so that we can present data about a specific movie for a logged in user.

```
SELECT * FROM movies WHERE movieID = ?
```

Query 5

This query is to find out the average score of a movie that is stored in a VIEW which makes it possible to present it to a user. To be able to pick out a specific movie we use the movieID.

```
SELECT average FROM average_score WHERE movieID = ?
```

Query 6

This query creates a VIEW called watch_time which we use to get a users total watch time of all the movies they have graded. So we check which movies a certain users has graded, then take the length of that movie and add it to the total when then is presented on the page in hours.

```
CREATE VIEW watch_time AS SELECT
ROUND(SUM(length)/ 60,1) AS hours, userID FROM
(SELECT *
FROM imduh.users
INNER JOIN imduh.scores
ON imduh.users.userID = imduh.scores.uID
INNER JOIN imduh.movies
ON imduh.movies.movieID = imduh.scores.mID) AS T
GROUP BY userID;
```

Query 7

This query allows the user to grade a specific movie on a scale from 1-5. We insert it into the scores table by using the VALUES needed which are movieID, userID and score. You can also change your original rating on a movie. For example you might have graded 3 at first but want to change to 5 later, you can do that.

```
INSERT INTO imduh.scores (mID, uID, score) VALUES (?)
ON DUPLICATE KEY UPDATE score =${values[2]}
```

Supplemental video

https://www.youtube.com/watch?v=fibMlaOulJU