SCOPE

The project “Movie Chronicle Database” (MCDB) consist of a convincing replica of one of the most popular movie database site called “IMDB” (Internet Movie Database). MCDB should provide users the facility to search through a central database of movies (T.V shows included) and find information related to a specific movie (or T.V show). This specific information shall provide users a complete synopsis of the movie including the official trailers and the ability to rate and comment on the movie.

The MCDB Web App shall have a versatile list of ways to interact with the database in order to fulfil the never ending users’ requirements. Such ways include searches of movies based on their names, genres, name of directors, actors, rating etc... This project includes the storage of users’ details for the sake of feedback and comments. Any person who uses this site is a guest but guests who have an account are users which is a specialized form of guest having an email and id along with the ability to rate (which a guest doesn’t).

The Web App is supposed be responsive, interactive, clean and visually minimal to provide a simple yet appealing interface to the users. It should be fast enough to deal with the unpredictable and variable load of the users.

The name of the website is “Movie Chronicle Database” (MCDB).

Technologies to be used are:

* HTML (Frontend)
* Pure CSS (Frontend)
* Pure Javascript (Frontend)
* PHP (Backend)
* MySQL (Backend)

Data is to be gathered from TMDB api.

The target delivery time for this Web App is around one month.

SCENARIO

The following information is needed to be stored for the project “MCDB”:

* *Movie Information*

This includes movie ID, movie Name, path to its cover image, path to its thumbnail, Synopsis, Release Date, movie Length, Rating, Popularity, a list of similar movies and an attribute that determines whether it’s a movie or a TV Show.

Information about movie genres, its actors, directors and trailers is also stored.

* *Episode Information*

This information is only stored for TV shows. It includes the episode ID, Name, path to thumbnail and number of Seasons.

* *Guest Information*

It includes information about the visitors of *IMDB Clone.* Guest ID and his/her Name along with any comments posted by them needs to be stored here.

* *User Information*

A user is a specialized guest for which we store an E-mail and password.

*Note that the difference between a guest and a user is that only a User is allowed to rate a movie while any guest can comment on any movie as many times as he/she wants.*

RELATIONS

From the EER we can bring up the following relations:

(Junction tables are not shown)

* ***Movie***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| id | name | cover | thumbnail | synopsis | releaseDate | length | rating | popularity | isTvSeries |
| 4900 | Harry Potter and The Deathly Hallows | address of cover | address of thumbnail | …………….. | 7-7-11 | 2:10:00 | 8 | 108 | 0 |
| 2305 | The Walking Dead | address of cover | address of thumbnail | …………….. | 31-10-10 | 00:44:00 | 8 | 8 | 1 |

***Functional Dependencies:***

Following important functional dependencies can be found in this relation:

fd1: id → name, cover, thumbnail, synopsis, parentsGuide, releaseDate, length, rating, popularity, isTvSeries

(Primary Key/Candidate Key)

fd2: cover→ id, name, thumbnail, synopsis, parentsGuide, releaseDate, length, rating, popularity, isTvSeries

(Candidate Key)

fd3: thumbnail→ id, name, cover, synopsis, parentsGuide, releaseDate, length, rating, popularity, isTvSeries

(Candidate Key)

***Normalization:***

**1st Normal Form:**

Since there are no repeating groups in this relation, therefore it’s already in 1st normal form.

**2nd Normal Form:**

We can see clearly that the primary key attribute is not a composite key and hence there is no chance of partial dependency. So this relation fulfils the condition for 2nd normal form as well.

**3rd Normal Form:**

None of the non-key attributes can be determined by any other non-key attributes. So there is no transitive dependency and hence the relation is in 3rd normal form.

**Boyce-Codd Normal Form:**

All the determinants are candidate keys, therefore the relation is already in Boyce-Codd normal form.

***Assumptions:***

1. isTvSeries is a binary attribute to decide between a TV Show or a Movie.

* ***TvSeries***

|  |  |
| --- | --- |
| seriesID | noOfSeasons |
| 2305 | 9 |
| 2665 | 5 |

***Functional Dependencies:***

There is only one functional dependency in this relation:

fd1: seriesID→noOfSeasons (Primary Key / Candidate Key)

***Normalization:***

We can clearly see that this table is in Boyce-Codd normal form.

***Assumptions:***

1. seriesID is a foreign key which references id in movie relation

* ***Episode***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id | name | thumbnail | seriesID | seasonNumber |
| 20 | Days Gone Bye | address Of Thmbnail | 2305 | 1 |
| 21 | Guts | address Of Thmbnail | 2305 | 1 |

***Functional Dependencies:***

Following important functional dependencies can be found in this relation:

fd1: id→ name, thumbnail, length, seriesID, seasonNumber (Primary Key / Candidate Key)

fd2: thumbnail→ id, name, length, seriesID, seasonNumber (Candidate Key)

fd3: name, seriesID → id, thumbnail, length, seasonNumber (Candidate Key)

***Normalization:***

**1st Normal Form:**

Since there are no repeating groups in this relation, therefore it’s already in 1st normal form.

**2nd Normal Form:**

We can see clearly that the primary key attribute is not a composite key and hence there is no chance of partial dependency. So this relation fulfils the condition for 2nd normal form as well.

**3rd Normal Form:**

None of the non-key attributes can be determined by any other non-key attributes. So there is no transitive dependency and hence the relation is in 3rd normal form.

**Boyce-Codd Normal Form:**

All the determinants are candidate keys, therefore the relation is already in Boyce-Codd normal form.

***Assumptions:***

1. seriesID is a foreign key which references seriesID in TvSeries relation.

* ***Trailer***

|  |  |  |
| --- | --- | --- |
| id | trailer | movieID |
| 33 | https://www.youtube.com/watch?v=I\_kDb-pRCds | 4900 |
| 33 | https://www.youtube.com/watch?v=5NYt1qirBWg | 4900 |
| 49 | https://www.youtube.com/watch?v=R1v0uFms68U | 2305 |

***Functional Dependencies:***

There is only one functional dependency in this relation:

fd1: id→ trailer, movieID (Primary Key / Candidate Key)

***Normalization:***

We can clearly see that this table is in Boyce-Codd normal form.

* ***Genre***

|  |  |
| --- | --- |
| id | genreType |
| 1 | Action |
| 2 | Adventure |

***Functional Dependencies:***

Following functional dependencies are present in this relation:

fd1: id→ genreType (Primary Key / Candidate Key)

fd2: genreType→id (Candidate Key)

***Normalization:***

**1st Normal Form:**

Since there are no repeating groups in this relation, therefore it’s already in 1st normal form.

**2nd Normal Form:**

We can see clearly that the primary key attribute is not a composite key and hence there is no chance of partial dependency. So this relation fulfils the condition for 2nd normal form as well.

**3rd Normal Form:**

There is only one determined attribute in both the dependencies, therefore there are no such non-key attributes that determine the other.

**Boyce-Codd Normal Form:**

All the determinants are candidate keys, therefore the relation is already in Boyce-Codd normal form.

***Assumptions:***

1. Either of the attributes of this relation could be used alone for the purpose of the project. However, both are decided to be kept for use in future.

* ***Director***

|  |  |
| --- | --- |
| id | name |
| 25 | David Yates |
| 56 | Frank Darabont |
| 45 | Frank Darabont |

***Functional Dependencies:***

The only dependency in this relation is:

fd1: id→ name (Primary Key / Candidate Key)

***Normalization:***

Clearly, the relation is in Boyce-Codd normal form.

***Assumptions:***

1. There can be many directors with the same name.

* ***Actor***

|  |  |
| --- | --- |
| id | name |
| 25 | Daniel Radcliffe |
| 56 | Morgan Freeman |
| 67 | Daniel Radcliffe |

***Functional Dependencies:***

The only dependency in this relation is:

fd1: id→ name (Primary Key / Candidate Key)

***Normalization:***

Clearly, the relation is in Boyce-Codd normal form.

***Assumptions:***

1. There can be many actors with the same name.

* ***Producers***

|  |  |
| --- | --- |
| id | name |
| 25 | Niki Marvin |
| 56 | J.K. Rowling |
| 98 | Niki Marvin |

***Functional Dependencies:***

The only dependency in this relation is:

fd1: id→ name (Primary Key / Candidate Key)

***Normalization:***

Clearly, the relation is in Boyce-Codd normal form.

***Assumptions:***

1. There can be many producers with the same name.

* ***Guest***

|  |  |
| --- | --- |
| id | name |
| 49 | MGB |
| 58 | Hameez |
| 301 | Hameez |

***Functional Dependencies:***

The only dependency in this relation is:

fd1: id→ name (Primary Key / Candidate Key)

***Normalization:***

Clearly, the relation is in Boyce-Codd normal form.

***Assumptions:***

1. Any person who visits the site is a guest however this relation is only established when the guest comments on a movie or TV show.

* ***User***

|  |  |  |
| --- | --- | --- |
| userID | email | password |
| 301 | [raffay007@gmail.com](mailto:raffay007@gmail.com) | RPass123 |
| 49 | [ghayasbaig247@gmail.com](mailto:ghayasbaig247@gmail.com) | GPass123 |
| 58 | [hameez123@gmail.com](mailto:hameez123@gmail.com) | GPass123 |

***Functional Dependencies:***

Following functional dependencies can be found in this relation:

fd1: userID→ email, password (Primary Key / Candidate Key)

fd2: email→ userID, password (Candidate Key)

***Normalization:***

There are no repeating groups, no partial dependencies and no transitive dependency, therefore the relation is already in 3NF

Also, all the dependencies are candidate keys so the relation is also in BCNF (Boyce-Codd normal form).

***Assumptions:***

1. A user is a specialized guest who is registered to the site. Users have the privilege to rate a movie while a guest do not.
2. No user can have same email.

SQL QUERIES

Following important types of SQL queries were used in the project:

* SELECT \* FROM guest,user WHERE email=:userEmail AND password=:userPass AND guest.id=user.userID;
* SELECT \* FROM movie WHERE ID=:requestedMovie;
* SELECT name FROM actor INNER JOIN actormovie ON actormovie.actorid = actor.id AND actormovie.movieid=:requestedMovie
* SELECT DISTINCT movie.id, movie.name, movie.thumbnail FROM movie INNER JOIN moviegenre ON moviegenre.genreID IN(SELECT DISTINCT moviegenre.genreID FROM moviegenre INNER JOIN movie ON moviegenre.movieID=:requestedMovie) AND moviegenre.movieID=movie.id AND isTvSeries=0 AND movie.id<>:requestedMovie"
* UPDATE userRating SET rating=:userRating WHERE userRating.userID=:userID AND userRating.movieID=:requestedMovie;
* INSERT INTO userRating(userID,movieID,rating) VALUES(:userID,:requestedMovie,:userRating);
* SELECT COUNT(DISTINCT movie.id) countMovies FROM movie INNER JOIN moviegenre ON moviegenre.movieid=movie.id INNER JOIN genre ON moviegenre.genreID = (SELECT DISTINCT genre.id FROM genre WHERE genre.genreName LIKE :reqGenre);
* SELECT id,name,cover,thumbnail,synopsis,releaseDate,rating FROM movie ORDER BY releaseDate DESC LIMIT 20 OFFSET :reqpg;
* SELECT genreName FROM genre INNER JOIN moviegenre ON moviegenre.genreID=genre.id AND moviegenre.movieID=:currentID;
* SELECT COUNT(\*) userCount, SUM(rating) totalMovieRating FROM userrating WHERE movieid=:requestedMovie;