1.ABSTRACT

Movies and TV Shows are currently a very popular means of entertainment among adults and children alike. In this era of the Internet, movies and TV shows are available easily online, often for free. Since everyone only has a limited amount of free time, they would like to be able to know the rating, reviews and other details of a movie or a TV show before they watch it. This creates a demand for a system where information about movies and shows should be available easily.

In this project we have designed a movie database which will gives a wide variety of information about movies & TV shows in various languages. A user can log in to rate, review movies and create a list of movies they want to watch etc.

This report contains details on all the important activities taken up during the course of the project. Starting from the requirement gathering , modelling the ER Diagram, Obtaining relation tables, Normalization followed by implementation details. We have also included important queries we have used to satisfy the functional requirements of the system and relevant statistical data on various tables in the database.

2. ACTORS

Actors refers to the people who will interact with the websiteThe actors of this website will primarily be

- Professional movie critics employed by newspapers and magazines
- Administrators who can add/edit/remove information
- General public

3. SAMPLE QUERIES

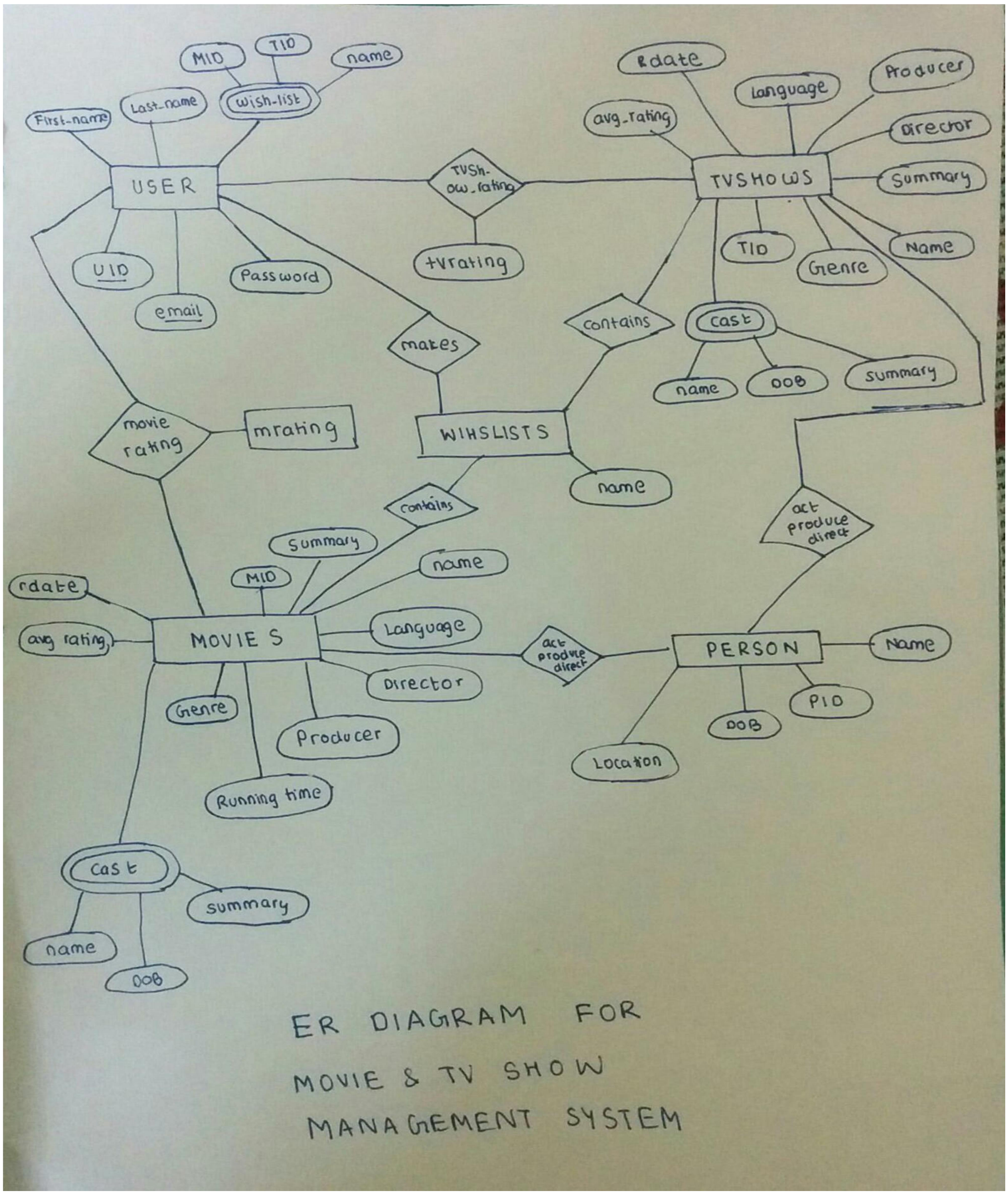
A user can use our website backed by the database to perform various functions as described below:

A user can

- Login to the website using his email id and password
- Signup to the website, filling up a form containing all his details
- Search for any movie/TV show using a search string
- Visit the website without login or signup
- Click on a movie title to view its details
- Click on a TV Show to view its details
- Click on an Actor to view his/her details
- Rate a movie/TV show
- Write a review for a movie/TV Show
- Create a wish list of movies to watch
- View the top 10 movies in any language or genre
- See the latest movies
- See the movies which are to be released soon
- Get movie/TV show recommendations based on their existing ratings/reviews
- See the success predictions of new movies

4. ER DIAGRAM

The following image contains the ER Diagram that was constructed keeping the functional and non-functional requirements of the system in mind.



5. TRANSFORMATION FROM ER TO RELATIONAL MODEL

From the ER Diagram made, the following relations were created.

Movies

MID	M_Title	M_Summary	M_poster_link	Director	PID	Cast_Details	Rating	Runtime	Date	Genre	Reviews

TV_Shows

TID	TV_Title	TV_Summary	TV_poster_link	Director	Cast_Details	Rating	Runtime	Date	Genre	No_episodes	Reviews

Users

UID	email	Password	First_Name	Last_Name	WishLists

6. ASSUMPTIONS

We have assumed the following details while implementing the project.

1. Each user can rate a particular movie or TV Show only once
2. Each user can review a particular movie or TV Show only once
3. The e-mail IDs of each user has to be unique. So, two or more users cannot have the same e-mail ID.
4. A user rates/reviews a movie/TV show only after watching it. So, these can be considered as appropriate parameters to make recommendations.

7.NORMALIZATION

Normalization is a systematic, step by step process carried out in order to reduce redundancies in the database. This is necessary because redundancies can cause various anomalies apart from wasting storage space.

The following anomalies can be noted in the above relations :

1.Insert Anomaly

Consider the situation where we have a new actor who has not yet acted in any movie or TV show. Then we cannot insert any data about this actor into any of the relations in the database because to insert into the movies table we need to know the Movie ID, MID as it is the primary key of the movies table and to insert into the TV_Shows table we need to know the TV Show ID, TID as it is the primary key of the TV_Shows table.Both Mid and TID are unavailable as the actor has not yet acted in any movie or TV Show. Thus, we come across an insert anamoly.

2.Delete Anomaly

Consider the situation where we delete all the movies from the movies table in which a particular actor has acted in. Then all the data about the actor is also lost, although this was not the intention. The same argument can be extended to the TV_Shows table as well. Thus, we have a delete anomaly.

3.Update Anomaly

Consider the situation where in we want to update the image of a particular actor in the movies table. Then we will have to update the image of the actor in multiple tuples of the movies relations as the same actor can act in multiple movies. If we miss out , to update the image of the actor for any one movie, then there will be inconsistency due to the update operation. Thus, we have an update anomaly.

When we normalize the relations, we end up with more relations which contain only related data. However, as the number of relations increase, we will have to perform more join operations and join is an extremely expensive operation. So, increased number of relations can hamper the performance of the database. Hence, we have restricted ourselves till the 3rd normal form which is industry standard as well.

Normalization of the relations can eliminate the above specified anomalies.

7.1 1st NORMAL FORM

Consider the movies relation. A movie can have multiple actors and actresses in it. So, cast_details is a multivalued attribute. Furthermore, cast_details has attributes like cast_name, caste_description, cast_DOB, cast_image etc. So, it is a composite attribute as well.So, we create a separate relation for storing the Cast of each movie. Additionally, each movie can have multiple ratings and reviews given by different users. So, we have to create separate relations to store these values as well. The Movies table stores the average rating of the movie given by users. We chose to treat the average rating as a stored attribute , instead of a derived attribute because it is used very frequently and computing it each time may affect the performance of the database.

Movies

MID	M_Title	M_Summary	M_poster_link	Director	Rating	Runtime	Date	Genre

MovieCast

MID	PID	Cast_name	Cast_image	Caste_DOB	Cast_description	Cast_Location

Movie_rating

UID	MID	Rating

MReviews

UID	MID	Review

Similar arguments can be raised for TV-Shows as well. So, we include another relation called TVCast to store information on the cast of each TV Show. We also add relations TV-rating and TVReviews to store the ratings and reviews for each TV show given by different users.

TV_Shows

TID	TV_Title	TV_Summary	TV_poster_link	Director	Rating	Runtime	Start_Date	Genre	No_episodes

TvShowCast

TID	PID	Cast_name	Cast_image	Caste_DOB	Cast_description	Cast_Location

TVShow_rating

UID	MID	Rating

TVReviews

UID	MID	Review

The user table contains the attribute wish lists. Each user can have multiple wish lists for Movies as well as TV shows. Wish lists is further a composite attribute. So, we break down the Users relation as shown below.

Users

UID	Username	Password	First_Name	Last_Name

MWishList

UID	ListName	MID

TVWishList

UID	ListName	TID

Now, all the relations are in 1st Normal form as all the relations satisfy the following consitions :

1. All the relations have primary keys which can be used to uniquely identify a tuple in a relation
2. No multi-valued attributes
3. No composite attributes

7.2 2nd NORMAL FORM

Consider the relation MovieCast with MID and PID as primary keys of the relation. So, MID and PID are prime attributes and the remaining attributes are non-prime attributes.This relation has the following functional dependancies :

MID PID -> Cast_name, Cast_image, Cast_DOB, Cast_description, Cast_Location

PID -> Cast_name, Cast_image, Cast_DOB, Cast_description, Cast_Location

In the second FD, the non-prime attributes are dependent on a subset of the primary key, namely PID. So, the non-prime attributes are not fully functionally dependant on the key of the relation. So, we break the MovieCast relation to bring it into 2nd Normal Form. After normalization we get a Person relation which stores data about the actors and another relation MCast that maps Movie IDs to Actor IDs.

Person

<u>MID</u>	<u>PID</u>	Cast_name	Cast_image	Caste_DOB	Cast_description	Cast_Location

MCast

<u>MID</u>	<u>PID</u>

A similar argument can be extended to the TVShowCast relation as well as it has the following FDs :

TID PID -> Cast_name, Cast_image, Cast_DOB, Cast_description, Cast_Location

PID -> Cast_name, Cast_image, Cast_DOB, Cast_description, Cast_Location

So, we break the relation TVShowCast into two relations as well. The first one is Person (the same relation obtained above for movie actors can be used here as well). So, we get one new relation TVCast.

TVCast

<u>TID</u>	<u>PID</u>

The relations obtained till now, are in 2nd Normal Form as they satisfy the following conditions :

1. All the relations are in 1st Normal Form
2. All non-prime attributes of a relation are fully functionally dependant on the key of the relation.

7.3 3rd NORMAL FORM

To bring the relations to 3rd normal form, we must ensure that a non-prime attribute is not dependant on another non-prime attribute. There are no such FDs in the existing relations. The only FDs remaining are the non-prime attributes dependant on the primary key of the relation. Hence, all the relations are already in 3rd normal form.

The relations obtained till now, are in 3rd Normal Form as they satisfy the following conditions :

1. All the relations are in 2nd normal form.
2. No non-prime attribute any relation is transitively dependant on the key of the relation.

We also have a recommendation system and a success prediction system. To implement these features we wanted to associate each user with actors/actresses he/she would mostly like. Each time a user rates a movie, we give points to the actor/actress in that movie, depending on the rating. We have a relation called FavActor to store the points of actors . Initially, all the actors have 0 points.

FavActor

UID	PID	Points

8. GLOBAL SCHEMA

The final set of relations implemented and used in the database are as follows :

1. Movies(MID, name,summary, poster_link, rdate, producer, director, composer, language, run_time, rating, genre)
2. TvShows(TID, name, summary,poster_link, director, start_date, seasons, run_time, rating, genre)
3. Person(PID, name, description, place, link, DOB)
4. MCast (MID, TID)
5. TVCast(TID, PID)
6. Users(UID, Username, first_name, last_name, password)

7. MovieRating(<u>UID</u>, <u>MID</u>, rating)
8. TVRating(<u>UID</u>, <u>TID</u>, rating)
9. MovieWishList (<u>UID</u>, <u>listname</u>, <u>MID</u>)
10. TVShowWishList(<u>UID</u>, <u>listname</u>, <u>TID</u>)
11. MovieReview (<u>UID</u>, <u>MID</u>, Review)
12. TVReview (<u>UID</u>, <u>TID</u>, Review)
13. FavActor(<u>UID</u>, <u>PID</u>, points)

9. TECHNOLOGIES USED

1. **Front-End**

   The entire GUI for the website has been built using HTML, CSS and Twitter Bootstrap. The Bootstrap framework was used to design the elements and increase the aesthetic beauty of the website. JavaScript was used as the client side scripting language. It was used mainly for log in and sign up.

2. **Back-End**

   The server side scripting language used was PHP. MySQL was used for database management. We used the WAMP (Windows Apache MySQL PHP) software stack to manage the server and databases. WAMP allows you to create web applications with Apache2, PHP and a MySQL database. The website is hosted locally on the WAMP server itself.

3. **Data management**

   The data on movies, TV shows and actors was extracted from Wikipedia using python scripts. These scripts used Beautiful Soup : A HTML parser to extract relevant data from the web page. The data extracted by these scripts was fed into a CSV file. The data was cleaned in Excel (for example format extracted release date to MM-DD-YYYY format). Then a PHP script was used to insert data into MySQL database from the CSV files.

10.PHYSICAL DESIGN

Relation	No. of records	Size of record(in bytes)
Movies	242	1302
TvShows	41	1285
Actors	73	1236
MCast	672	8
TVCast	181	8
Users	7	94
MovieRating	10	12
TVShowRating	7	12
MovieWishList	2	38
TVShowWishlist	2	38
MovieReview	10	1008
TVReview	7	1008
FavActor	2	16

11. IMPORTANT QUERIES USED

11.1 Creating the tables for the database

create table Movies (mid int auto_increment, name varchar(30) not null,summary varchar(1000),posterlink varchar(100),rdate date,producer varchar(30),director varchar(30),composer varchar(30),language varchar(30),runtime int,rating float,genre varchar(20), primary key (mid));

create table TvShows (tvid int auto_increment, name varchar(30) not null,posterlink varchar(200),director varchar(20),rating float,runtime int,startdate date,genre varchar(15),seasons int,summary varchar(),primary key(tvid));

create table users (uid int auto_increment,email varchar(30),firstname varchar(20),secondname varchar(20),password varchar(20), primary key (uid));

create table movierating (uid int ,mid int, rating int, primary key (uid,mid));

create table tvshowrating (uid int, tvid int ,rating int, primary key (uid,tvid));

create table mwishlist(uid int, listname varchar(30),mid int, primary key (uid,mid,listname));

create table tvwishlist(uid int, listname varchar(30),tvid int, primary key (uid,tvid,listname));

create table mreview (uid int,mid int,review varchar(1000), primary key (uid,mid));

create table tvreview (uid int,tvid int,review varchar(1000), primary key (uid,tvid));

create table actors (name varchar(20),description varchar(1000),link varchar(1000),place varchar(100),dob date,pid int auto_increment, primary key(pid));

create table favactor (uid int,pid int,points float, primary key (uid,pid));

11.2.1 Recommending movies based on Favourite Actor

select actors.name from actors,mcast where mcast.pid=actors.pid and mcast.mid=3;

Selecting movies which include the user’s favorite actor in the cast:

select * from movies,mcast where movies.mid=mcast.mid and mcast.pid=(select pid from favactor where uid=$uid order by points desc limit 1);

11.3 Recommending movies based on genre

select * from movies mo where genre = (select genre from movies as m1,movierating as m2 where m2.uid=1 and m1.mid=m2.mid and m2.rating = (select max(d.rating) from movierating as d where d.uid=1)) order by mo.rating desc limit 5;

12. REFERENCES

1. Fundamentals of Database Systems, Elmasri Navathe, 3rd Edition, Pearson Education, 2006
2. Database System Concepts, Abraham Schilberschatz, Henry Korth, S Sudarshan, 6th Edition, 2007
3. http://www.w3schools.com/
4. http://tutorialspoint.com/
5. http://getbootstrap.com/getting-started/
6. http://php.net/manual/
7. https://www.sitepoint.com/php-amp-mysql-1-installation/
8. http://www.homeandlearn.co.uk/php/php1p3.html
9. http://stackoverflow.com/
10. http://tahirtaous.blogspot.in/2012/04/how-to-hodedisable-php-errors-in-wamp.html