**DBMS – Mini Project**

**Netflix Database**

Submitted By:

Name: Shashank Varma

SRN: PES1UG20CS395

V Semester: Section G

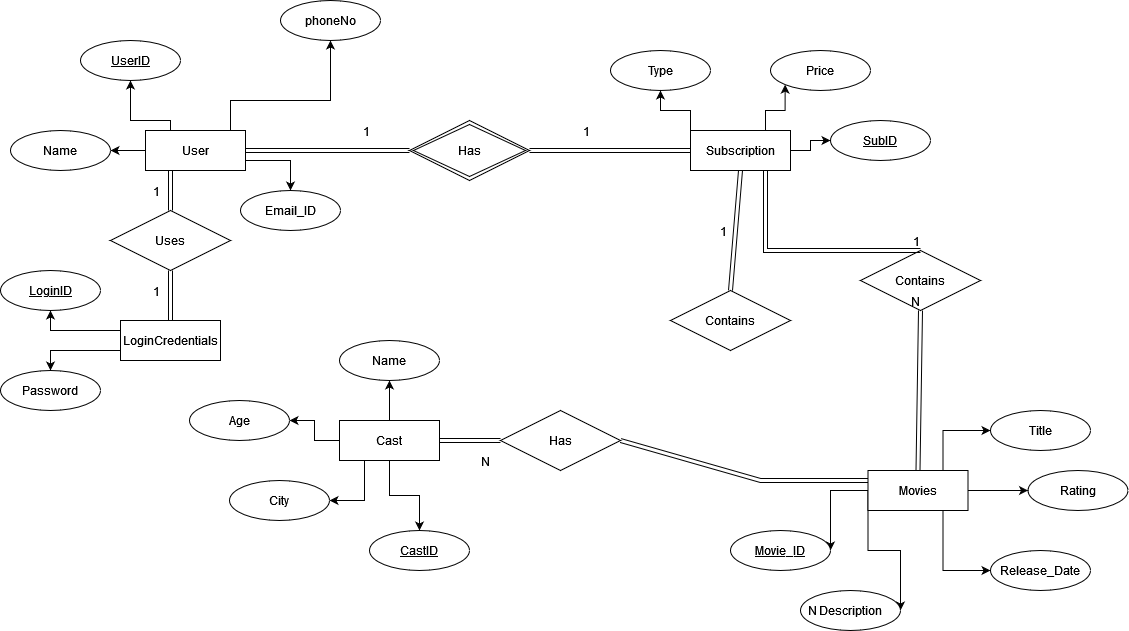
**Short Description and Scope of Project**

This is a clone of the Netflix Database done using mySQL and Streamlit.

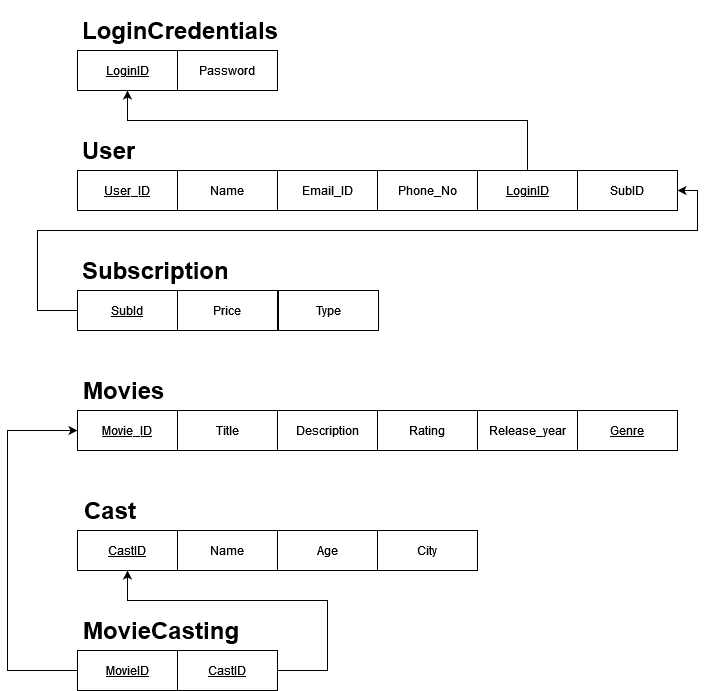
It consists of 5 tables – User, Login, Movies, Cast and MovieCasting relation. The user has to first login using his credentials. If he is a new user, he can register first and then login. He can then use the home page where all list of movies available are shown along with the average movie rating. After selecting one of them, he can view the full information of the movie in the movie page along with the cast, short summary and ratings. There is also an option to rate the movie. On rating, the rating gets updated to the average of the new and old rating.

The admin has some functions available at backend which can be used to get information regarding the users, their subscriptions, cast info and so on.

**ER Diagram**



**Relational Schema**



**DDL Statements – Building Database**

CREATE TABLE loginCredentials (LoginID varchar(20) NOT NULL, password varchar(20), PRIMARY KEY(LoginId));

CREATE TABLE user (userId varchar(20) NOT NULL, name varchar(20), emailId varchar(20), phoneNo BIGINT(10),LoginId varchar(20), PRIMARY KEY(UserId), FOREIGN KEY(LoginId) REFERENCES loginCredentials(LoginId));

CREATE TABLE movie (movieId int(10) NOT NULL, title varchar(20), description varchar(20), rating float(10), releaseYear YEAR, genre varchar(20), PRIMARY KEY(movieId));

CREATE TABLE webSeries (seriesId int(10) NOT NULL, title varchar(20), description varchar(20), rating float(10), releaseYear YEAR, genre varchar(20), noOfEpisodes INT(10), PRIMARY KEY(seriesId));

CREATE TABLE subscription (subId INT(10) NOT NULL, type varchar(20), price float(10), PRIMARY KEY(subId));

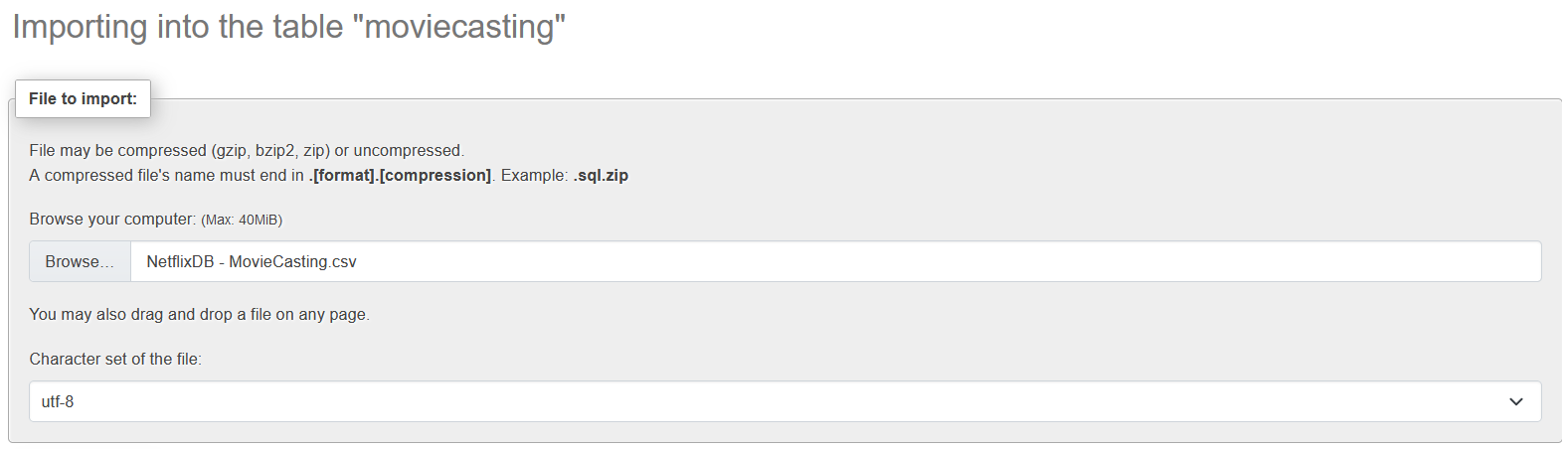
CREATE TABLE castInfo (castId INT(10) NOT NULL, name VARCHAR(20), age INT(10), city VARCHAR(20), PRIMARY KEY(castId));

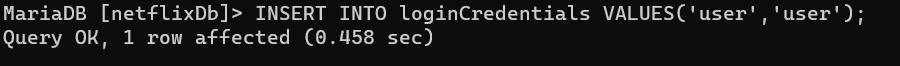
CREATE TABLE moviecasting (castId INT(10) NOT NULL, movieId int(10) NOT NULL, FOREIGN KEY(castId) REFERENCES castInfo(castId), FOREIGN KEY(movieId) REFERENCES movie(movieId));

ALTER TABLE user ADD COLUMN subId INT(10);

ALTER TABLE user ADD CONSTRAINT FOREIGN KEY(subId) REFERENCES subscription(subId);

**Populating the Database**



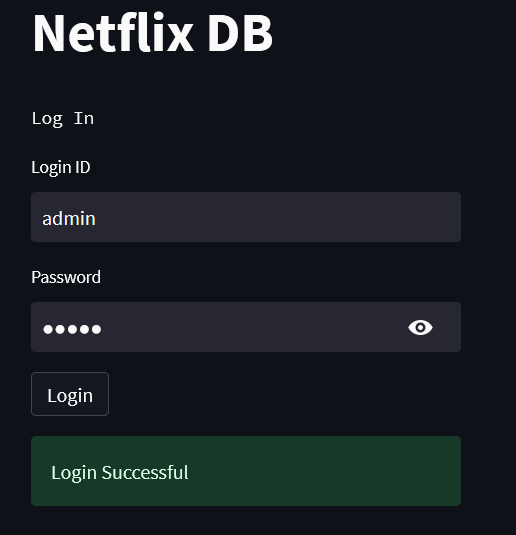
****

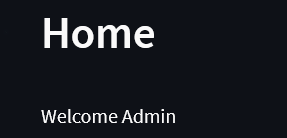


**Join Queries**

1. This performs JOIN on user and login credentials to fetch the user information as soon as he logs in.

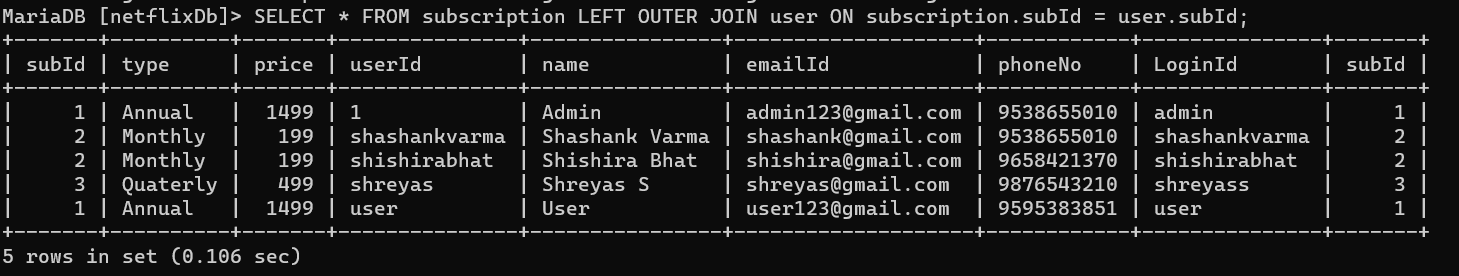
SELECT \* FROM logincredentials, user WHERE logincredentials.LoginId = %s AND logincredentials.password = %s AND logincredentials.LoginId = user.LoginId





2. This performs a LEFT OUTER JOIN on the user and subscription tables for the admin to see the users and the list of subscription he/she has made.

SELECT \* FROM subscription LEFT OUTER JOIN user ON subscription.subId = user.subId;



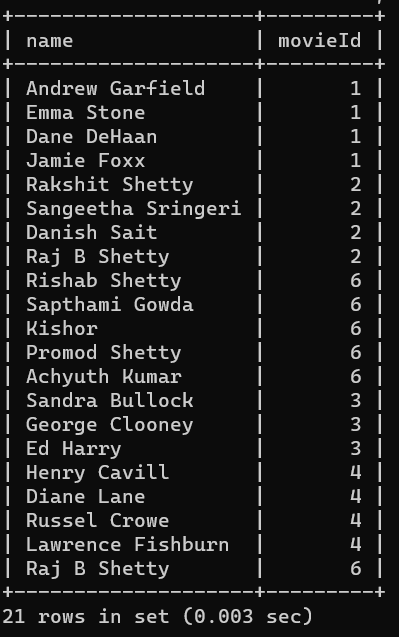
3. This performs a join on the movie and cast tables to retrieve the list of cast for a particular movie. This is displayed on the movie info page.

SELECT movieCast.name, movie.title FROM movieCast, movie WHERE movieCast.movieId=movie.movieId;



4. This join is FULL OUTER JOIN on the movieCasting relation and cast table to retrieve the list of all movies and all the cast.

SELECT \* FROM (SELECT castInfo.name,movieCasting.movieId FROM castInfo RIGHT OUTER JOIN movieCasting ON castInfo.castId=movieCasting.castId UNION SELECT castInfo.name,movieCasting.movieId FROM castInfo LEFT OUTER JOIN movieCasting ON castInfo.castId=movieCasting.castId) as c;



**Aggregate Function**

1. This is used to get a total count of movies available to watch.

SELECT COUNT(\*) FROM movie;



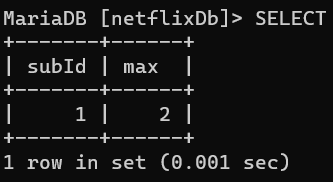
1. This command is used to get the average rating of the movies on Netflix. This shows the people’s liking towards current movies.

SELECT AVG(movie.rating) FROM movie;

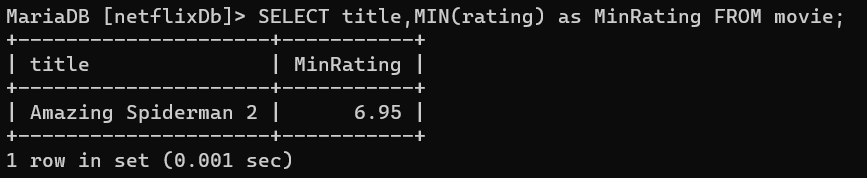


1. This command is used to get the subscription with the most users.

SELECT subId,MAX(subCount) AS max FROM (SELECT subId, COUNT(\*) AS subCount FROM user GROUP BY subId) AS a;

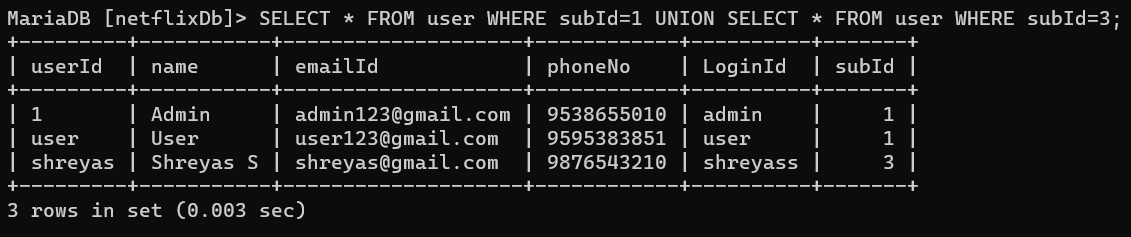


1. This command is used to find the movie with the least rating.

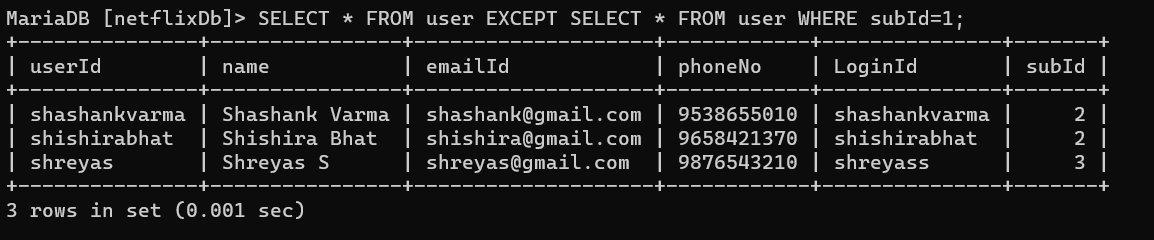
SELECT title,MIN(rating) as MinRating FROM movie;

**Set Operations**

1. This command might be used to get list of all users with the annual and quarterly subscription in order to provide them some offer.

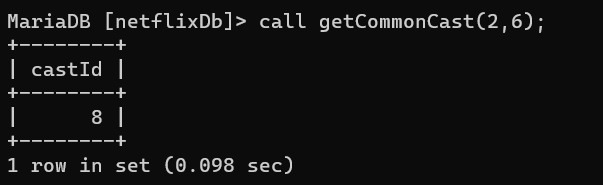
SELECT \* FROM user WHERE subId=1 UNION SELECT \* FROM user WHERE subId=3;

1. This can be used to select all users who haven’t subscribed for annual to remind them to subscribe or to provide them with some offer.

SELECT \* FROM user EXCEPT SELECT \* FROM user WHERE subId=1;

1. This command is used to get common cast between 2 movies to display in the “Also acted in” section.

SELECT castId FROM movieCasting where movieId=movie1 INTERSECT SELECT castId FROM movieCasting where movieId=movie2;



We can see that cast with id=8 has acted in both movie 2 and movie 6.

**Functions And Procedures**

1. This function can be used to get the total number of actors in the movie.

DELIMITER $$

CREATE FUNCTION totalCast(movId INT(10))

RETURNS INT(10)

DETERMINISTIC

BEGIN

DECLARE res INT(10);

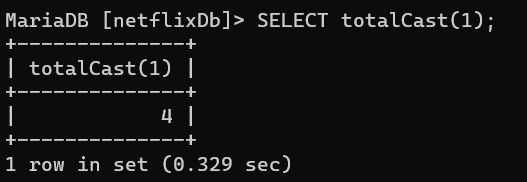
SELECT COUNT(\*) INTO res FROM movieCasting GROUP BY movieId HAVING movieCasting.movieId=movId;

RETURN res;

END;$$

DELIMITER ;

SELECT totalCast(1);



We can see that a total of 4 actors are present in the movie with id 1.

1. This procedure can be used to check how many movies each actor has acted in.

DROP PROCEDURE getCastMovies;

DELIMITER $$

CREATE PROCEDURE getCastMovies()

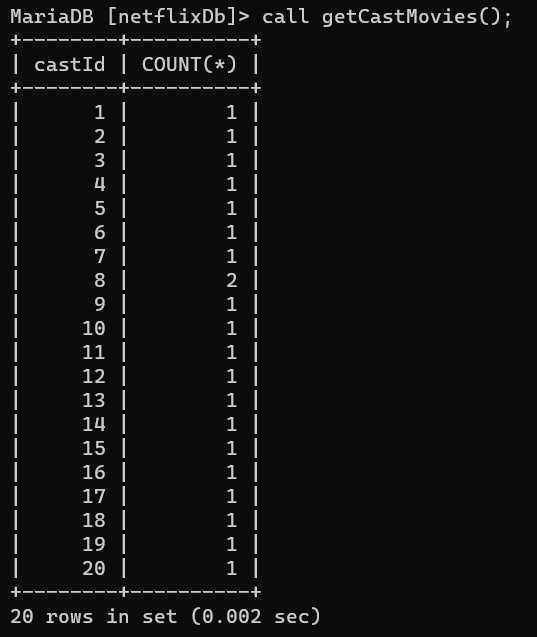
BEGIN

SELECT castId,COUNT(\*) from movieCasting GROUP BY castId;

END;$$

DELIMITER ;

call getCastMovies();



**Triggers and Cursors**

1. This trigger is used to make sure the rating stays between 0 and 10 while updating. If it is above 10, it made 10 and if it is below zero, it is made zero before updating.

DELIMITER $$

CREATE TRIGGER rating\_check BEFORE UPDATE ON movie

FOR EACH ROW

BEGIN

IF NEW.rating < 0 THEN

SET NEW.rating = 0;

ELSEIF NEW.rating > 10 THEN

SET NEW.rating = 10;

END IF;

END;$$

DELIMITER ;

1. This trigger is used to set the rating of a movie to be the average of the old rating and the new rating entered by the user.

DELIMITER $$

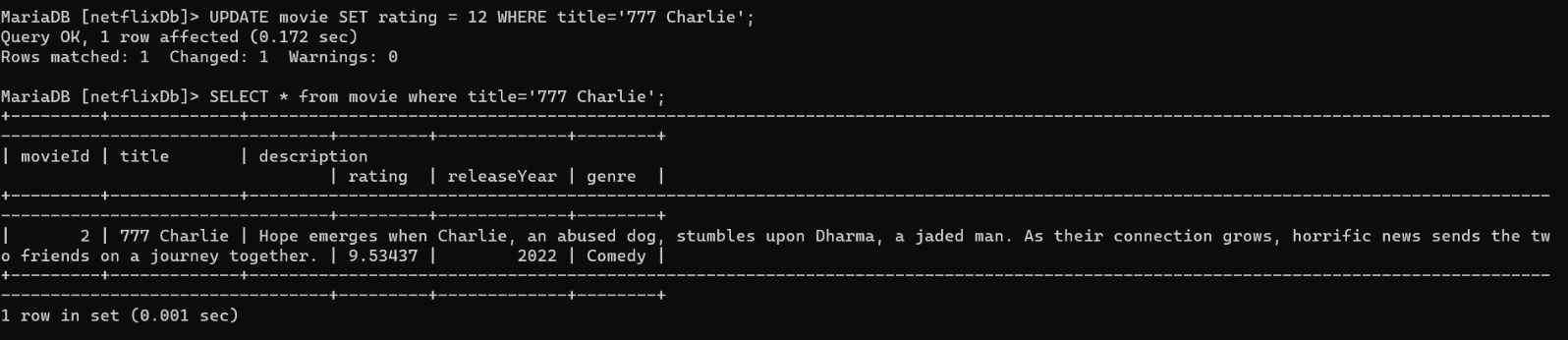
CREATE TRIGGER update\_rating BEFORE UPDATE ON movie

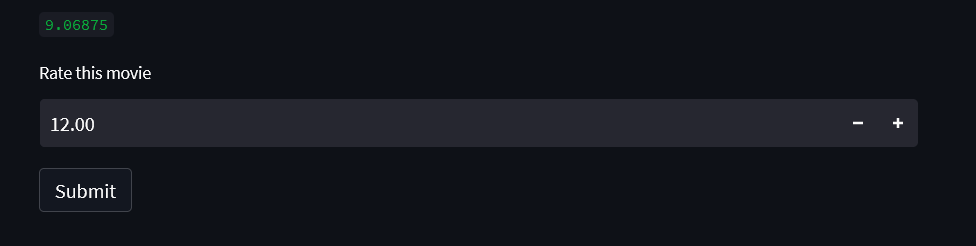
FOR EACH ROW

BEGIN

SET NEW.rating=(OLD.rating+NEW.rating)/2;

END;$$

DELIMITER ;

Before update:

After update:

1. This cursor is used to get all the names of users as list. This can be used by the admin to get the list of users.

DELIMITER $$

CREATE PROCEDURE list\_name(INOUT user\_list varchar(1000))

BEGIN

DECLARE is\_done INTEGER DEFAULT 0;

DECLARE uName varchar(20) DEFAULT "";

DECLARE getNames CURSOR FOR

SELECT name FROM user;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET is\_done=1;

OPEN getNames;

get\_names: LOOP

FETCH getNames INTO uName;

IF is\_done=1 THEN

LEAVE get\_names;

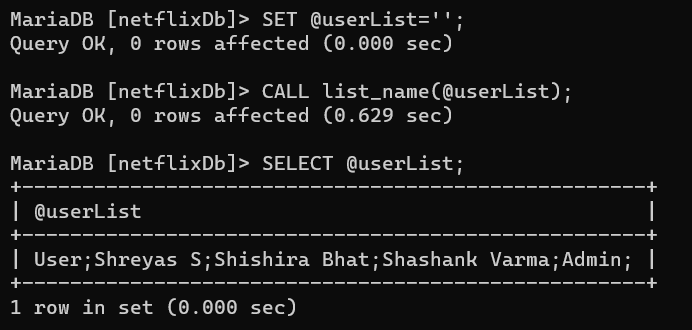
END IF;

SET user\_list=CONCAT(uName, ";",user\_list);

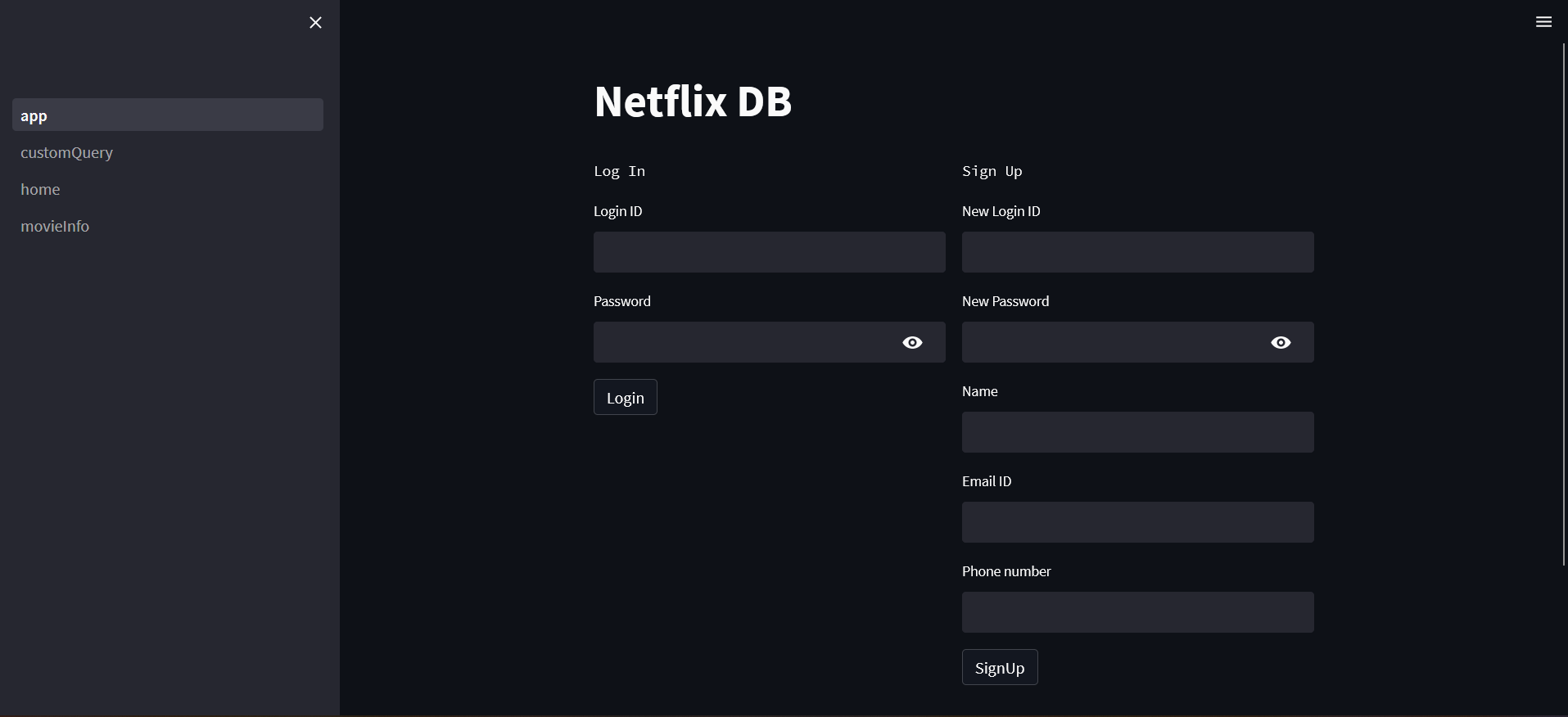
END LOOP get\_names;

END;$$

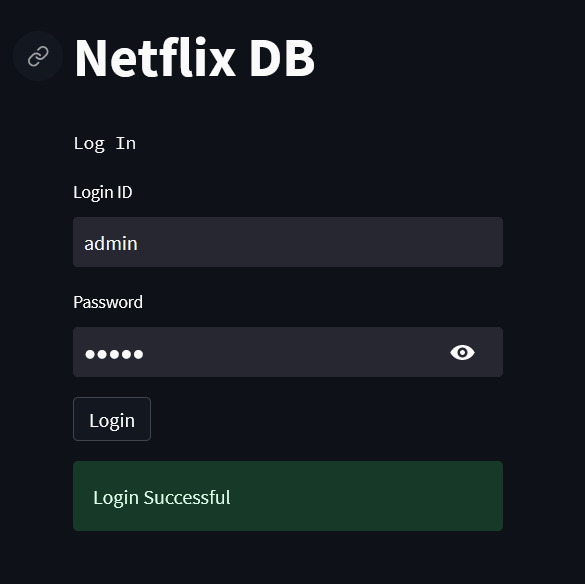
DELIMITER ;



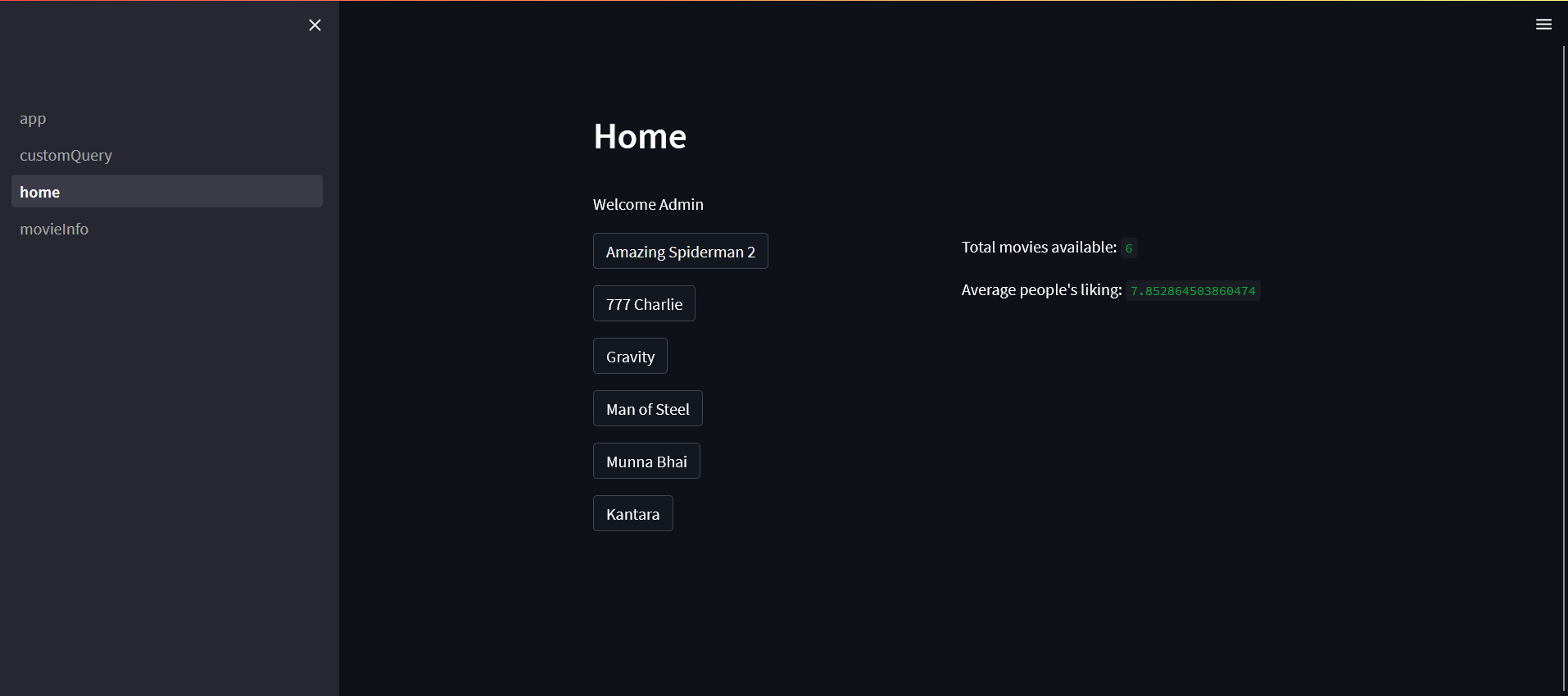
**Developing Frontend**

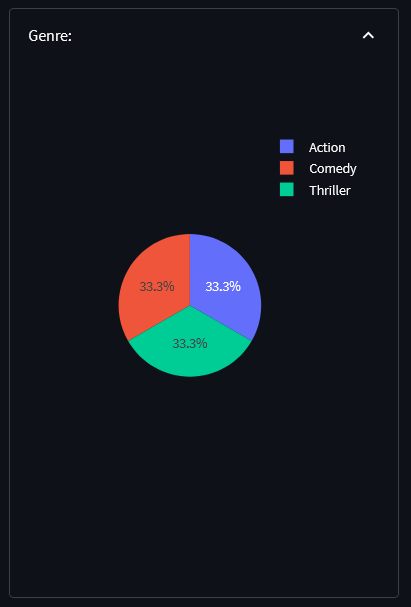
Login page

After successful login:

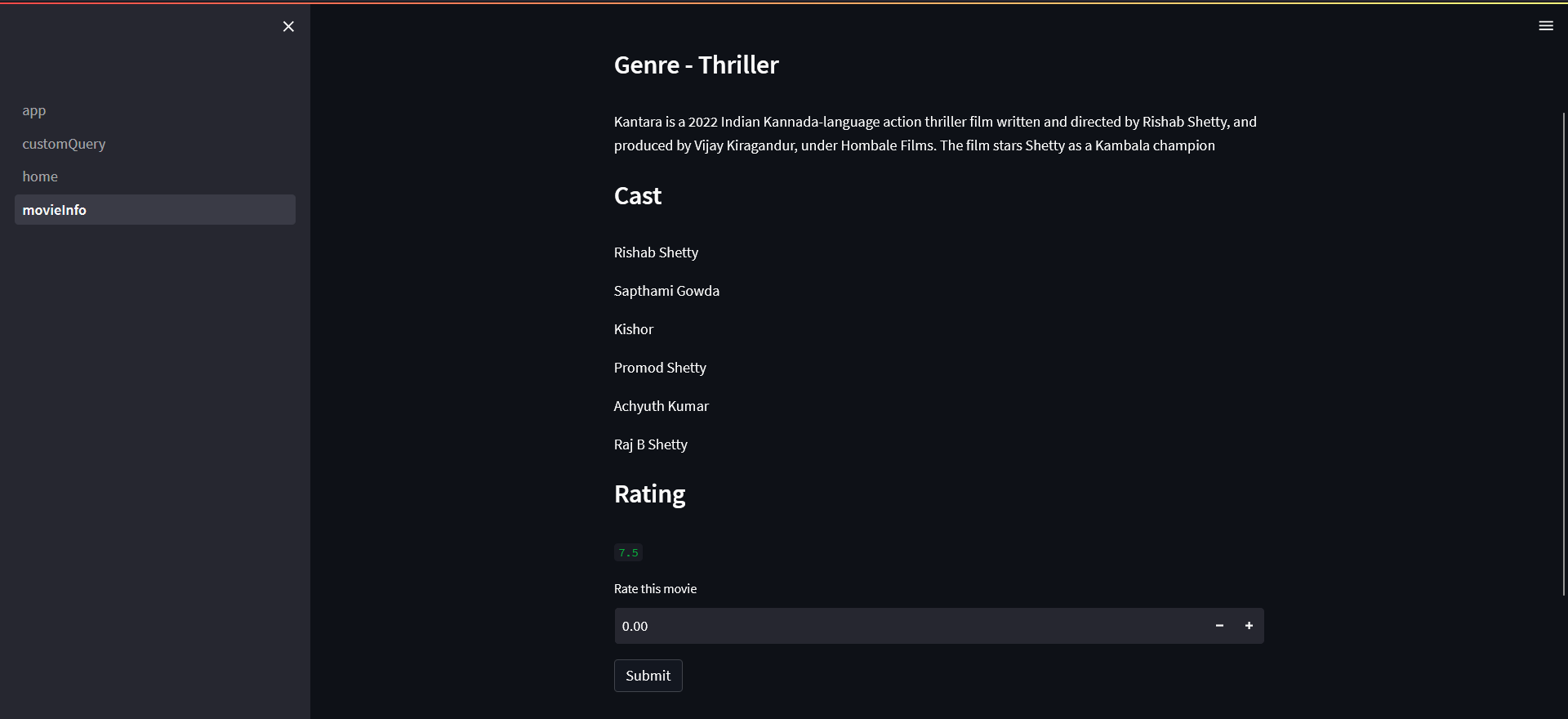


Home page:





Movie information page:



Custom query page:

