CLAIM

ABOUT

Nowadays, we use recommendation systems for almost every decision we take. They are in out musics, products to buy, books to read, movies to watch. It is an important service when you are building an app that communicates with the people. In this project, we wanted to create Web Application that can help people to watch movies based on their ratings. Its recommendation based on Collaborative Filtering algorithm that filters information by using the recommendations of other people. It is based on the idea that people who agreed in their evaluation of certain items in the past are likely to agree again in the future.

You can read more about CF in : recommender-systems.org/collaborative-filtering/

DATA AND REQUIREMENT ANALYSIS

Database will consist of tables such as Movie, MovieGenre, Occupation, Rating and User.

- Movie: Table identified by unique MovieID as Primary Key, Title as varchar and Genre as varchar.
- MovieGenre: Table has MovieID as Foreign Key and Genre as varchar.
- Occupation: Table identified by unique OccID as Primary Key and Occ_Desc as varchar.
- Rating: Table identified by MovieID and PersonID as Combined Primary Key and Rating as int.
- User: Table identified by unique PersonID as Primary Key, Gender as varchar, Age as int, ZipCode as bigint, AgeDesc as varchar and OccID as Foreign Key.

Every table has a relation one to another such as:

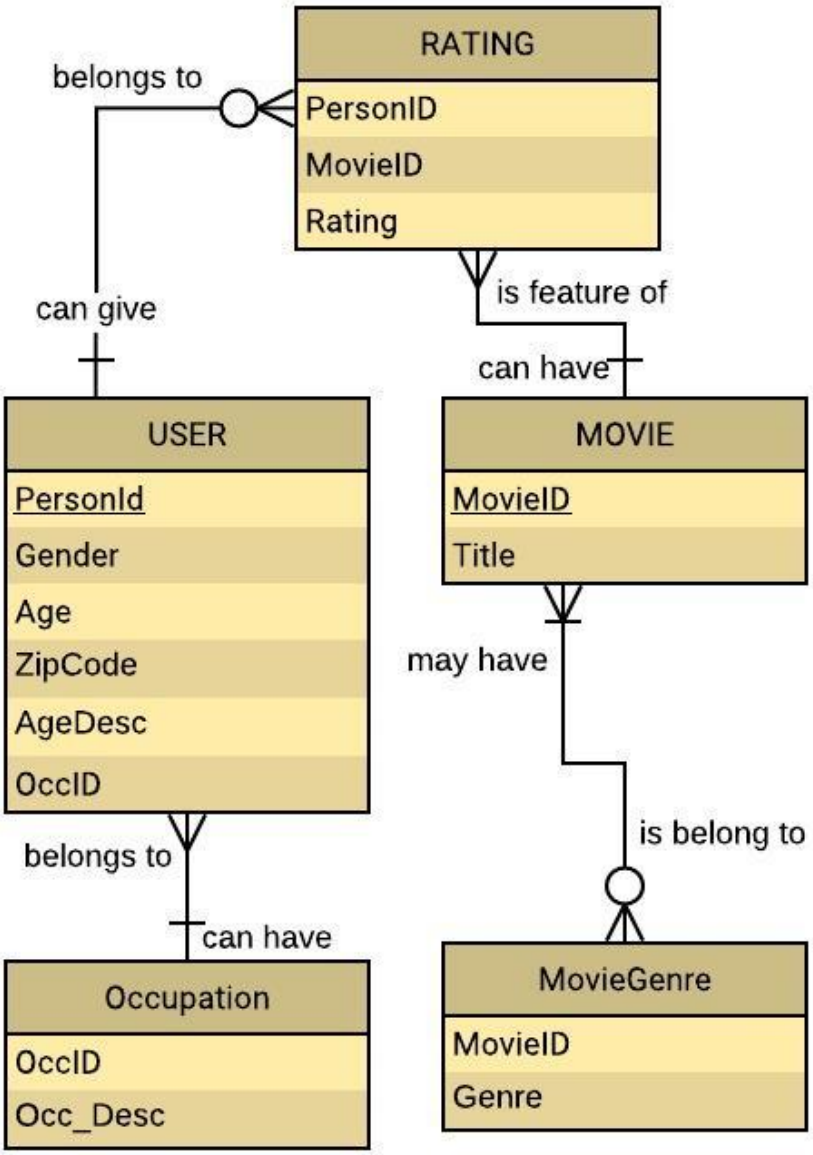
- Movie can have multiple MovieGenre but every MovieGenre has only one Movie.
- Movie can have multiple Rating, but each Rating only belongs to one Movie.
- User can rate multiple Rating but each Rating only belongs to one User.
- User can have only one Occupation but each Occupation can have multiple User.

This project contains the data of Movielens Dataset it has 1,000,000 ratings from 6000 users on 4000 movies.

You can check out the dataset: grouplens.org/datasets/movielens/

Dataset includes 4 different files.
We created and manipulated the data with Python with the help of the Pandas and NumPy modules.

E-R Diagram


RATING
PersonID
MovieID
Rating
belongs to
is feature of
can give
can have
USER
PersonId
Gender
Age
ZipCode
AgeDesc
OccID
MOVIE
MovieID
Title
may have
is belong to
belongs to
can have
Occupation
OccID
Occ_Desc
MovieGenre
MovieID
Genre

TABLES

- MOVIE
  - Movie table has MovieID and Title fields.
  - MovieID: It is the Primary Key that identifies each Movie from each other. It is also the key to reach the Rating table.
  - Title: This field has the name of the corresponding movie.

MovieID	Title
1	Toy Story (1995)
2	Jumanji (1995)
3	Grumpier Old Men (1995)
4	Waiting to Exhale (1995)
5	Father of the Bride Part II (1995)
6	Heat (1995)

- MOVIEGENRE
  - MovieGenre table has MovieID and Genre fields.
  - MovieID: It is a Foreign Key that can connect with the Movie table.
  - Genre: Genre information about the corresponding MovieID.

MovieID	Genre
1	Animation
1	Children's
1	Comedy
2	Adventure
2	Children's
2	Fantasy
3	Comedy

- OCCUPATION
  - Occupation table has OccID and Occ_Desc fields.
  - OccID: It is the Primary Key that can connect us with the User table.
  - Occ_Desc: It is the occupation for the corresponding OccID.

OccID	Occ_Desc
0	other or not specified
1	academic/educator
2	artist
3	clerical/admin
4	college/grad student

- RATING
  - Rating table has MovieID, PersonID and Rating as fields.
  - MovieID: It is a part of the Combined Primary Key with the PersonID. It can help for connection to Movie table.
  - PersonID: It is a part of the Combined Primary Key with the MovieID. It can help for connection to User table.
  - Rating: Corresponding rating for the specific rating action from a User to a Movie.

PersonID	MovieID	Rating
1	1	5
1	48	5
1	150	5
1	260	4
1	527	5
1	531	4
1	588	4
1	594	4

- USER
  - User table has PersonID, Gender, Age, Zipcode, AgeDesc and OccID as fields.
  - PersonID: It is the Primary Key that specifies a User. That can help to connect with the Rating table and Occupation table.
  - Gender: Gender value for the User.
  - Age: Age value for the User.
  - Zipcode: Zipcode value for the User.
  - AgeDesc: Age description value for the User. It is an aggregated column from Age column.
  - OccID: Occupation identifier that can help to connect with the Occupation table.

VIEWS

- **GetCustomerRatings:** It is designed to get the ratings for the Web App User. In the app, the user is in the index of 0. So we are getting all the ratings from the 0 index. It is showing the results in descending order for the Rating column in Rating table.

```
select `rating`.`PersonID` AS `PersonID`,`rating`.`MovieID` AS `MovieID`,`rating`.`Rating` AS `Rating`
from `rating`
where (`rating`.`PersonID` = 0)
order by `rating`.`Rating` desc
```

PersonID	MovieID	Rating
0	2	4
0	27	4
0	270	2
0	5	1

- **GetMostPopularMovies:** It is designed to get the most popular movies across the Rating table. It also computes the total ratings, average ratings and number of ratings for the corresponding MovieID. Orders by descending and limits them to 200. So it selects top 200 movies that gets the most number of ratings, biggest total value of ratings and biggest average ratings.

```
select `M`.`Title` AS `Title`,(sum(`R`.`Rating`) / count(`R`.`Rating`)) AS `Average_Rating`,count(`R`.`Rating`) AS `NumberOfRatings`,sum(`R`.`Rating`) AS `Total_Rating`
from (`rating` `R`
join `movie` `M`
on((`M`.`MovieID` = `R`.`MovieID`)))
group by `R`.`MovieID`
order by `NumberOfRatings` desc,`Total_Rating` desc,`Average_Rating` desc
limit 200
```

Title	Average_Rating	NumberOfRatings	Total_Rating
American Beauty (1999)	4.3174	3428	14800
Star Wars: Episode IV - A New Hope (1977)	4.4537	2991	13321
Star Wars: Episode V - The Empire Strikes Back (1980)	4.2930	2990	12836
Star Wars: Episode VI - Return of the Jedi (1983)	4.0229	2883	11598
Jurassic Park (1993)	3.7638	2672	10057
Saving Private Ryan (1998)	4.3374	2653	11507
Terminator 2: Judgment Day (1991)	4.0585	2649	10751
Matrix, The (1999)	4.3158	2590	11178

- **PopularGenresAcrossOccupations**

It is designed for getting knowledge about which Movie Genres are the most watched/rated across Occupations that every Users has. It first computes the summation of ratings for each Occupation and each Genre. Then It computes the biggest summations that every occupation has. Later on it joins that two table to get the Genres that has biggest summation across the each Occupation.

```
SELECT `occdata`.`occ_desc`  AS `Occ_Desc`,
       `occgenre`.`genre`    AS `Genre`,
       `occdata`.`maxrating` AS `MaxRating`
FROM   ((SELECT `t`.`occ_desc`          AS `Occ_Desc`,
                Max(`t`.`total_rating`) AS `MaxRating`
         FROM   (SELECT `O`.`occ_desc`    AS `Occ_Desc`,
                        `MG`.`genre`      AS `Genre`,
                        Sum(`R`.`rating`) AS `Total_Rating`
                 FROM   ((((`occupation` `O`
                            JOIN `user` `U`
                              ON(( `U`.`occid` = `O`.`occid` )))
                           JOIN `rating` `R`
                             ON(( `U`.`personid` = `R`.`personid` )))
                          JOIN `movie` `M`
                            ON(( `M`.`movieid` = `R`.`movieid` )))
                         JOIN `moviegenre` `MG`
                           ON(( `MG`.`movieid` = `M`.`movieid` )))
                 GROUP  BY `O`.`occ_desc`,
                           `MG`.`genre`
                 ORDER  BY `total_rating` DESC) `t`
         GROUP  BY `t`.`occ_desc`) `OccData`
        JOIN (SELECT `O`.`occ_desc`    AS `Occ_Desc`,
                     `MG`.`genre`      AS `Genre`,
                     Sum(`R`.`rating`) AS `Total_Rating`
              FROM   ((((`occupation` `O`
                         JOIN `user` `U`
                           ON(( `U`.`occid` = `O`.`occid` )))
                        JOIN `rating` `R`
                          ON(( `U`.`personid` = `R`.`personid` )))
                       JOIN `movie` `M`
                         ON(( `M`.`movieid` = `R`.`movieid` )))
                      JOIN `moviegenre` `MG`
                        ON(( `MG`.`movieid` = `M`.`movieid` )))
              GROUP  BY `O`.`occ_desc`,
                        `MG`.`genre`
              ORDER  BY `total_rating` DESC) `OccGenre`
          ON(( `occgenre`.`total_rating` = `occdata`.`maxrating` )))
```

Occ_Desc	Genre	MaxRating
executive/managerial	Drama	162238
college/grad student	Drama	159836
other or not specified	Drama	136578
technician/engineer	Drama	116737
academic/educator	Drama	114804
programmer	Drama	90051
sales/marketing	Comedy	67375
writer	Drama	59276
doctor/health care	Drama	58889
self-employed	Drama	58212
artist	Drama	57889

STORED PROCEDURES

- **DeleteRatings:** It deletes the rows from Rating table with the arguments of user_id and movie_id. If movie_id or user_id is given smaller than -1, it will ignore that field when it runs conditions.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `DeleteRatings`(IN user_id INT, IN movie_id INT)
BEGIN


  IF user_id <= -1 AND movie_id > -1 THEN
        DELETE FROM Rating WHERE MovieID=movie_id;
  ELSEIF user_id > -1 AND movie_id <= -1 THEN
      DELETE FROM Rating WHERE PersonID=user_id;
  ELSEIF user_id > -1 AND movie_id > -1 THEN
      DELETE FROM Rating WHERE MovieID=movie_id AND PersonID=user_id;
  END IF;
END
```

- **GetMoviesTable:** It will select the rows from Movie table with the argument of Limit. If Limit is given, it will limit the results (-in the app when page is home, it will query 20 results from Movie table.) If limit is not given, it will select all the rows from the table.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetMoviesTable`(IN limit_arg int)
BEGIN
  DECLARE query_string VARCHAR(60);
  CASE
    WHEN limit_arg <0 THEN
      SET @query_string = 'SELECT * FROM Movie';
    ELSE
      SET @query_string = CONCAT('SELECT * FROM Movie LIMIT ', limit_arg) ;
  END CASE;
  PREPARE myquery FROM @query_string;
  EXECUTE myquery;
END
```

- **GetNumberOfMovies:** It will return the number of movies from the Movie table. MovieID is not an index so we can't COUNT(*) them.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetNumberOfMovies`()
BEGIN
    SELECT MAX(MovieID) AS 'NumberOfMovies' FROM Movie;
 END
```

- **GetNumberOfUsers:** It will return the number of users from the User table. PersonID is not an index so we can't COUNT(*) them.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetNumberOfUsers`()
BEGIN
  SELECT MAX(PersonID)AS 'NumberOfUsers' FROM User;
END
```

- **GetRatingsTable:** It will select the rows from Rating table with the argument of Limit. If Limit is given, it will limit the results. If limit is not given, it will select all the rows from the table.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetRatingsTable`(IN limit_arg int)
BEGIN
  DECLARE query_string VARCHAR(60);
  CASE
    WHEN limit_arg <0 THEN
      SET @query_string = 'SELECT * FROM Rating';
    ELSE
      SET @query_string = CONCAT('SELECT * FROM Rating LIMIT ', limit_arg) ;
  END CASE;
  PREPARE myquery FROM @query_string;
  EXECUTE myquery;
END
```

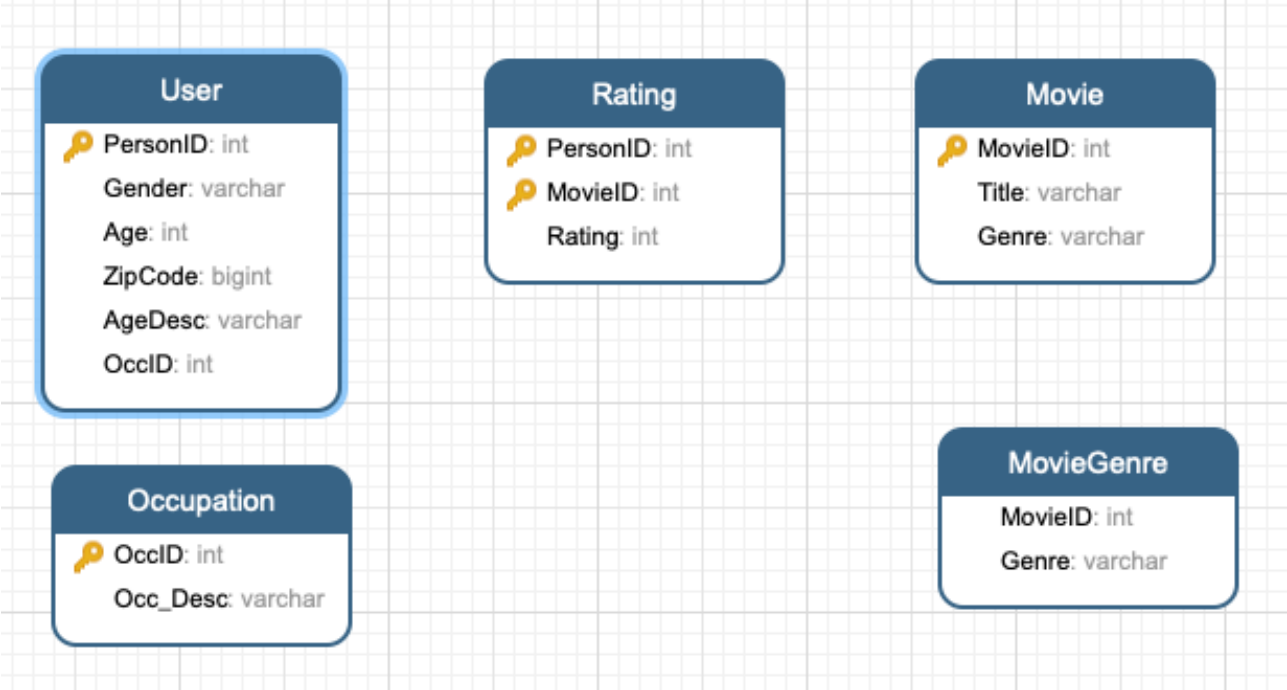
- **GetUsersTable:** It will select the rows from User table with the argument of Limit. If Limit is given, it will limit the results. If limit is not given, it will select all the rows from the table.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `GetUsersTable`( IN limit_arg int)
BEGIN
  DECLARE query_string VARCHAR(60);
  CASE
    WHEN limit_arg <0 THEN
      SET @query_string = 'SELECT * FROM User';
    ELSE
      SET @query_string = CONCAT('SELECT * FROM User LIMIT ', limit_arg) ;
  END CASE;
  PREPARE myquery FROM @query_string;
  EXECUTE myquery;

END
```

- **InsertRating:** It will insert a row to the Rating table with the movie_id, user_id and rating as arguments.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `InsertRating`(IN user_id int, IN movie_id int, IN rating int )
BEGIN
  INSERT INTO Rating VALUES (user_id, movie_id, rating);

END
```

CONSTRAINTS



- **User:**
  - PersonID can't be smaller than 0.
  - Gender is F or M.
  - Age can't be smaller than 0 or bigger than 106.
  - OccID have to be between 0 and 11.

- **Rating:**
  - PersonID can't be smaller than 0.
  - MovieID can't be smaller than 0.
  - Rating have to be between 0 and 5.

- **Movie:**
  - MovieID can't be smaller than 0.

- **MovieGenre:**
  - MovieID can't be smaller than 0.

- **Occupation:**
  - OccID have to be between 0 and 11.