Matthew LaPorta SQL Database Project

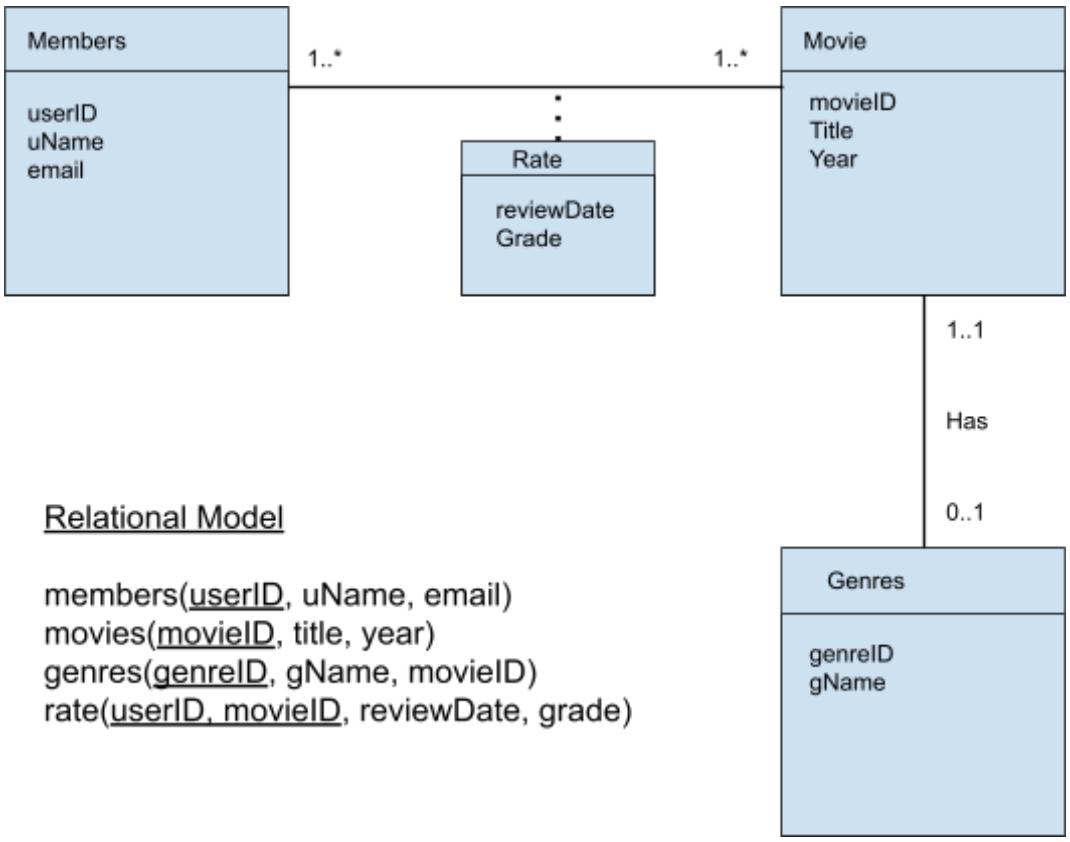
Title: Movies Recommendation Database

Introduction:

I wanted to create a database that tracks movies, their genres, and their ratings. The goal of the database is to recommend movies based on the ratings provided by members, which are in the form of letter grades. I went with letter grades instead of numerical ratings because the system works by assigning a grade, such as an 'A', to an action movie, for instance. Then, if another user has rated an action movie with an 'A' grade that you haven't seen, it would be recommended to you.

There are 4 tables: members, movie, genres, and rate. The members table tracks the user's ID, name, and email. The movie table tracks a movie's ID, title, and year it was released. The genre table tracks a genre's id and name. Lastly, the rate table tracks the grade a user gives a movie and when this grade was given. Maketables.sql has all table information, Populatetables.sql has all the data for the tables, and searchrecords.sql is where the queries were made.

UML Diagram



Relational Model

members(userID, uName, email)
movies(movieID, title, year)
genres(genreID, gName, movieID)
rate(userID, movieID, reviewDate, grade)

—----------

Maketables.sql

```
Drop TABLE members cascade constraints;

CREATE TABLE members (
userID VARCHAR(50) NOT NULL,
uName VARCHAR(50) NOT NULL,
email VARCHAR(50) NOT NULL,
PRIMARY KEY(userID));

TRUNCATE TABLE members;

Drop TABLE movies cascade constraints;

CREATE TABLE movies (
movieID VARCHAR(50) NOT NULL,
title VARCHAR(50) NOT NULL,
year NUMBER NOT NULL,
PRIMARY KEY(movieID));

TRUNCATE TABLE movies;

Drop TABLE genres cascade constraints;

CREATE TABLE genres (
genreID VARCHAR(50) NOT NULL,
gName VARCHAR(50) NOT NULL,
movieID VARCHAR(50) NOT NULL,
FOREIGN KEY (movieID) REFERENCES movies(movieID));

TRUNCATE TABLE genres;

Drop TABLE rate cascade constraints;

CREATE TABLE rate (
userID VARCHAR(50) NOT NULL,
movieID VARCHAR(50) NOT NULL,
reviewDate DATE NOT NULL,
grade VARCHAR(50) NOT NULL,
```

```
Primary KEY(userID, movieID),
FOREIGN KEY (userID) REFERENCES members(userID),
FOREIGN KEY (movieID) REFERENCES movies(movieID));

TRUNCATE TABLE rate;
```

Populatetables.sql

```
INSERT INTO members VALUES('u1','Bob','Bob@gmail.com');
INSERT INTO members VALUES('u2','Bill','Bill@gmail.com');
INSERT INTO members VALUES('u3','Ben','Ben@gmail.com');
INSERT INTO members VALUES('u4','Frank','Frank@gmail.com');
INSERT INTO members VALUES('u5','Matt','Matt@gmail.com');
INSERT INTO members VALUES('u6','Kyle','Kyle@gmail.com');
INSERT INTO members VALUES('u7','Ryan','Ryan@gmail.com');
INSERT INTO members VALUES('u8','Robert','Robert@gmail.com');
INSERT INTO members VALUES('u9','Kate','Kate@gmail.com');
INSERT INTO members VALUES('u10','Emily','Emily@gmail.com');

SELECT * FROM members;

INSERT INTO movies VALUES('m1','Wall-E',2008);
INSERT INTO movies VALUES('m2','Inside Out',2015);
INSERT INTO movies VALUES('m3','Ratatouille',2007);
INSERT INTO movies VALUES('m4','The Fall Guy',2024);
INSERT INTO movies VALUES('m5','The Princess Bride',1987);
INSERT INTO movies VALUES('m6','The Lego Batman Movie',2017);
INSERT INTO movies VALUES('m7','Drive',2011);
INSERT INTO movies VALUES('m8','Barbie',2023);
INSERT INTO movies VALUES('m9','Dune: Part Two',2024);
INSERT INTO movies VALUES('m10','Stardust',2007);
INSERT INTO movies VALUES('m11','The Other GUys',2010);
INSERT INTO movies VALUES('m12','John Wick: Chapter 4',2023);

SELECT * FROM movies;

INSERT INTO genres VALUES('g1','Science Fiction','m1');
INSERT INTO genres VALUES('g1','Science Fiction','m9');
INSERT INTO genres VALUES('g2','Adventure','m2');
```

```
INSERT INTO genres VALUES('g2','Adventure','m3');
INSERT INTO genres VALUES('g3','Fantasy','m5');
INSERT INTO genres VALUES('g3','Fantasy','m10');
INSERT INTO genres VALUES('g4','Action','m4');
INSERT INTO genres VALUES('g4','Action','m7');
INSERT INTO genres VALUES('g5','Comedy','m6');
INSERT INTO genres VALUES('g5','Comedy','m8');
INSERT INTO genres('g4','Action','m12');
INSERT INTO genres('g5','Comedy','m11');

SELECT * FROM genres;

INSERT INTO rate VALUES('u1','m1',DATE '2024-05-08','B');
INSERT INTO rate VALUES('u2','m2',DATE '2024-05-11','A');
INSERT INTO rate VALUES('u3','m3',DATE '2024-05-11','A');
INSERT INTO rate VALUES('u4','m4',DATE '2024-05-22','A');
INSERT INTO rate VALUES('u5','m7',DATE '2024-05-23','A');
INSERT INTO rate VALUES('u6','m9',DATE '2024-05-01','B');
INSERT INTO rate VALUES('u7','m10',DATE '2024-05-02','A');
INSERT INTO rate VALUES('u8','m5',DATE '2024-05-03','A');
INSERT INTO rate VALUES('u9','m8',DATE '2024-05-05','B');
INSERT INTO rate VALUES('u10','m6',DATE '2024-05-13','A');
INSERT INTO rate VALUES('u2','m3',DATE '2024-05-11','A');
INSERT INTO rate VALUES('u2','m4',DATE '2024-05-11','A');

SELECT * FROM rate;
```

Searchrecords.sql

```
/* Movies in the fantasy/adventure genre that have received an 'A' rating given from
5/11/2024-5/22/2024 */

SELECT DISTINCT m.movieID, m.title, m.year
FROM movies m, rate r, genres g
WHERE m.movieID = r.movieID
      AND r.movieID = g.movieID
      AND r.grade = 'A'
      AND g.gName IN('Fantasy','Adventure')
      AND r.reviewDATE >= DATE '2024-05-11'
      AND r.reviewDATE <= DATE '2024-05-22';
```

```
/* List the user who has rated the most movies (group by) */

SELECT userID, COUNT(DISTINCT movieID) AS num_rated_movies
FROM rate
GROUP BY userID
HAVING COUNT(DISTINCT movieID) = (
	SELECT MAX(movie_count)
	FROM (
		SELECT COUNT(DISTINCT movieID) AS movie_count
		FROM rate
		GROUP BY userID
	)max_counts
);

/* List movies that have not been rated (Minus) */

SELECT m.movieID, m.title, m.year
FROM movies m
MINUS
SELECT r.movieID, m.title, m.year
FROM rate r, movies m
WHERE r.movieID = m.movieID;

/* Comedy Movies that have been given an 'A' rating and have not been rated by user 'u9'
(Recommendation) */

SELECT m.movieID, m.title, m.year, 'Comedy' AS genre, 'A' AS grade
FROM movies m
JOIN genres g ON m.movieID = g.movieID AND g.gName = 'Comedy'
WHERE NOT EXISTS (
	SELECT *
	FROM rate r
	WHERE r.userID = 'u9'
	AND r.movieID = m.movieID
);
```

Conclusion:

Regarding the queries in searchrecords.sql: The first query lists fantasy or adventure movies rated 'A' between 5/11/2024 and 5/22/2024, relevant for users interested in sorting by

genre and high ratings. The second query identifies users who rated the most movies, which could be particularly helpful if friends became a feature in the database, enabling users to compare their movie preferences with one another and see who has watched more movies. The third query lists unrated movies, potentially useful for user recommendations or comparing movie collections with friends. Lastly, the fourth query recommends comedy movies unrated by user 9 that are highly rated by others, directly aligning with the database's goal of recommending movies based on user preferences.

The goal of my database was to recommend movies to users based on other movies they liked by matching genres and the scores given by other users. I think this goal was achieved, and it is kind of like a barebones Letterboxd. A couple of ways that my project could be improved would be to switch from letter grades to numerical grades. When coming up with the idea for the project, I imagined that it would be small and be meant more to be used among friends. If it were going to be bigger like a website such as letterboxd, I think numerical grades would be better since you could take the average of the ratings easier without having to go through extra steps. Overall, the process of making a simple database was interesting and it made me think about how apps or websites I use might function on a database level.