Erik F Garcia

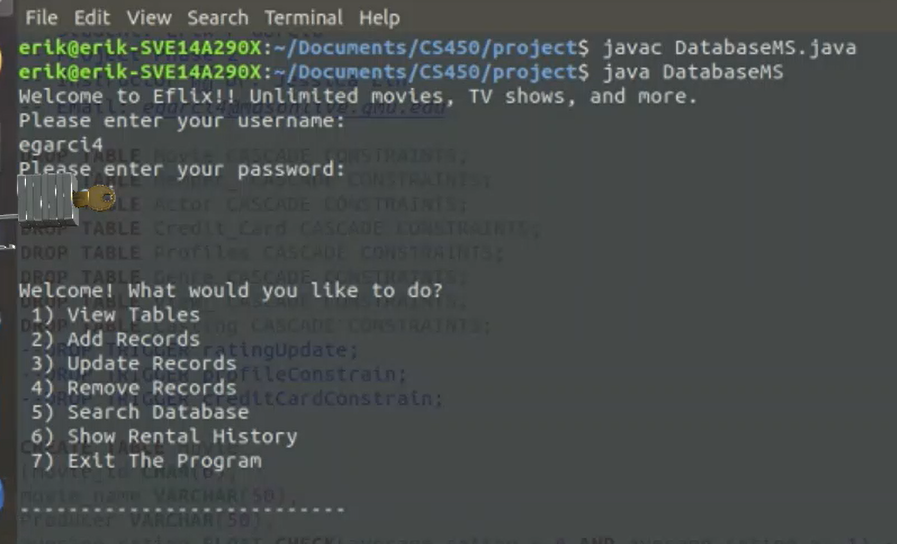
Project Phase 3

Instructor – Dr. Jessica Lin

Final Report

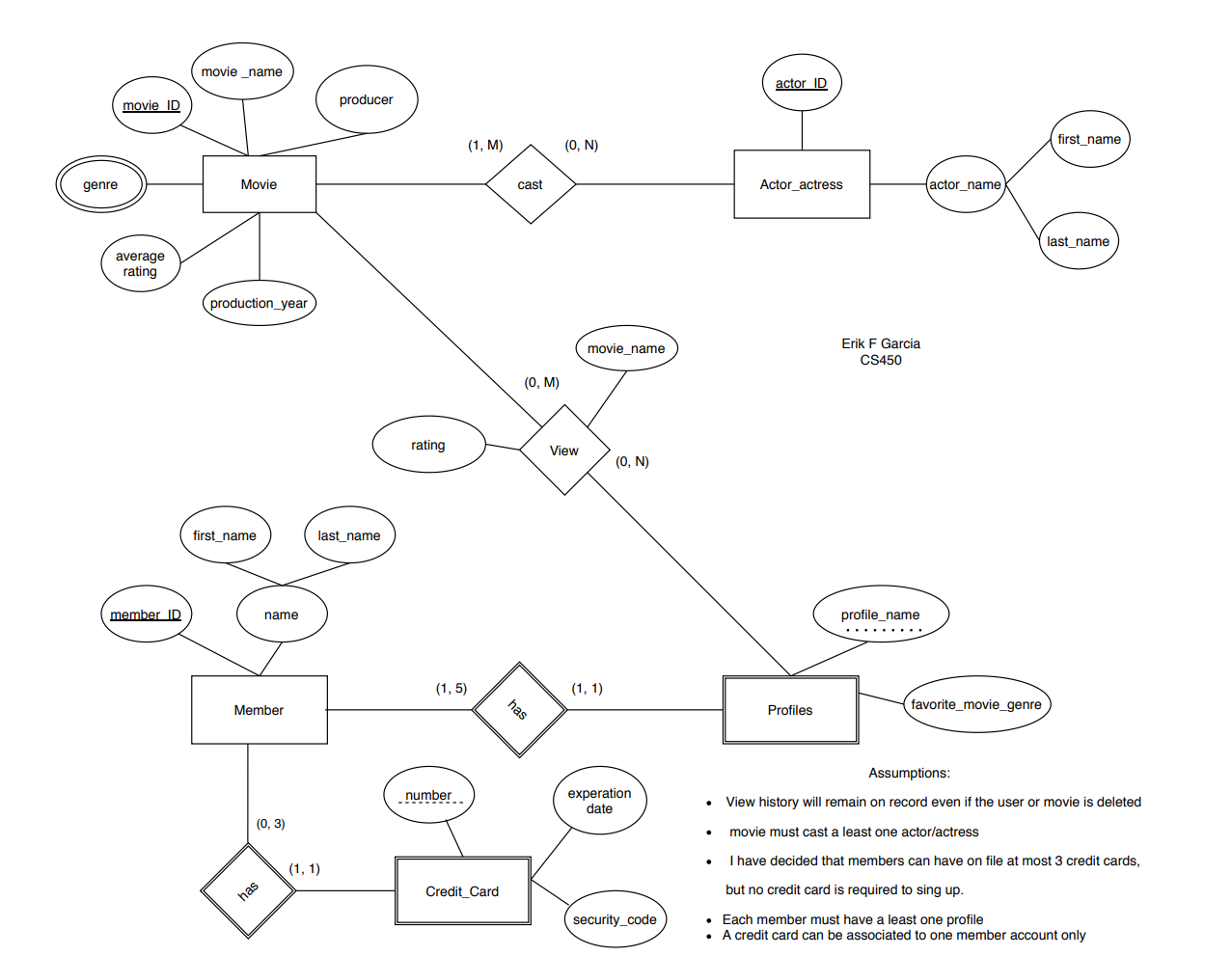
Video Demo: <https://www.youtube.com/watch?v=kuGnn0Z7nBA>

• Screenshots on program

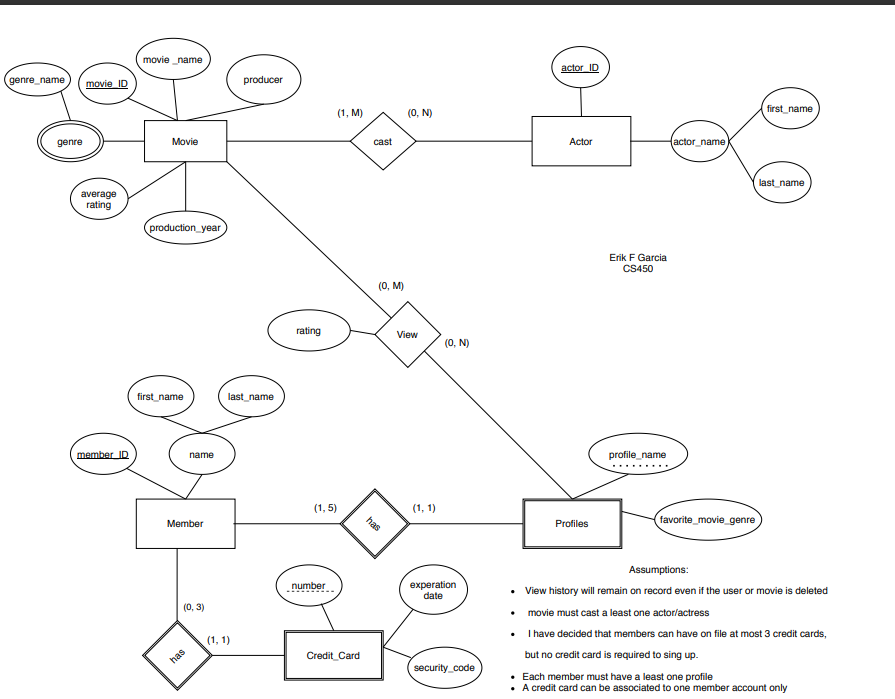


**Phase 1**

The goal of phase 1 was to designed an ED diagram that would satisfied the requirement. There were minor changes made by the end of phase 2.



Changes to original ER diagram by phase 2 marked by arrows



This ER diagram was used for phase 3

**Phase 2**

One of the goals of phase 2 was to translate the ER diagram into relation schema. The following is the relation schema presented for phase 2.

Strong Entities

• Movie(movie\_ID, movie\_name, producer, average\_rating, production\_year )

• Member\_(member\_ID, first\_name, last\_name)

• Actor(actor\_ID, first\_name, last\_name)

Weak Entities

• Credit\_Card(member\_ID , number, exp\_date, security\_code)

Foreign Key : member\_ID References Member

• Profiles(member\_ID , profile\_name, favorite\_movie\_genre)

Foreign Key : member\_ID References Member

• Genre(movie\_ID, genre\_name)

Foreign Key : movie\_ID References Movie

Relations

• View\_ (movie\_ID, member\_ID, profile\_name, rating)

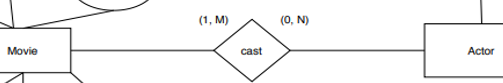
Foreign Key: movie\_ID references: Movie

Foreign Key: movie\_name references: Movie

Foreign Key: member\_ID references: Profiles

Foreign Key: profile\_name references: Profiles

This schema is **incomplete** because it did not include the relation “Cast”. Therefore, the phase 2 SQL script is also incomplete. This mistake was corrected during phase 3



**Phase 3**

During phase 3 “Casting” was aggregated to the relation schema of Phase 2. The SQL script was corrected. Triggers and checks were added to satisfy the constrains.

Strong Entities

* Movie(movie\_ID, movie\_name, producer, average\_rating, production\_year )
* Member\_(member\_ID, first\_name, last\_name)
* Actor(actor\_ID, first\_name, last\_name)

Weak Entities

* Credit\_Card(member\_ID , number, exp\_date, security\_code)

Foreign Key : member\_ID References Member

* Profiles(member\_ID , profile\_name, favorite\_movie\_genre)

Foreign Key : member\_ID References Member

* Genre(movie\_ID, genre\_name)

Foreign Key : movie\_ID References Movie

Relations

* View\_ (movie\_ID, member\_ID, profile\_name, rating)

Foreign Key: movie\_ID references: Movie

Foreign Key: movie\_name references: Movie

Foreign Key: member\_ID references: Profiles

Foreign Key: profile\_name references: Profiles

* Casting (movie\_ID, actor\_ID)

Foreign Key: movie\_ID references: Movie

Foreign Key: actor\_ID references: Actor

Table added to the schema

The following is a fraction of the Script used for this project it shows the creation of table and triggers. The complete script is provided with this report

|  |  |
| --- | --- |
| CREATE TABLE **Movie**  (movie\_id CHAR(6),  movie\_name VARCHAR(50),  Producer VARCHAR(50),  average\_rating FLOAT CHECK(average\_rating < 6  AND average\_rating >= 1),-- restrict 1-5  production\_year INTEGER,  PRIMARY KEY(movie\_id)  );  CREATE TABLE **Member\_**  (member\_id CHAR(6),  first\_name VARCHAR(30),  last\_name VARCHAR(30),  PRIMARY KEY(member\_id)  );  CREATE TABLE **Actor**  (actor\_id CHAR(6),  first\_name VARCHAR(50),  last\_name VARCHAR(50),  PRIMARY KEY(actor\_id)  );  CREATE TABLE **Credit\_Card**  (member\_id CHAR(6),  number\_ VARCHAR(16),  exp\_date DATE,  security\_code CHAR(4),  PRIMARY KEY(member\_id, number\_),  FOREIGN KEY(member\_id)REFERENCES Member\_ ON DELETE CASCADE  );  CREATE TRIGGER **creditCardConstrain**  BEFORE INSERT ON Credit\_Card  FOR EACH ROW  DECLARE  creditCardNum INTEGER;  Too\_many EXCEPTION;  BEGIN  SELECT COUNT(\*) INTO creditCardNum  FROM Credit\_Card  WHERE member\_id = :NEW.member\_id;  IF creditCardNum > 2 THEN  RAISE Too\_many;  END IF;  EXCEPTION  WHEN Too\_many THEN  RAISE\_APPLICATION\_ERROR(-20000, 'You can have up to 3 credit cards on file!');  END creditCardConstrain;  /  CREATE OR REPLACE TRIGGER **ratingUpdate**  FOR INSERT ON View\_ COMPOUND TRIGGER  newRating FLOAT;  m\_id CHAR(6);  BEFORE EACH ROW IS  BEGIN  m\_id := :NEW.movie\_id;  END BEFORE EACH ROW;  AFTER STATEMENT IS  BEGIN  SELECT AVG(rating)INTO newRating  FROM View\_  WHERE movie\_id = m\_id;  UPDATE Movie  SET average\_rating = newRating  WHERE movie\_id = m\_id;  END AFTER STATEMENT;  END ratingUpdate;  / | CREATE TABLE **Profiles**  (member\_id CHAR(6),  profile\_name VARCHAR(50),  favorite\_movie\_genre VARCHAR(50),  PRIMARY KEY(member\_id, profile\_name),  FOREIGN KEY(member\_id)REFERENCES member\_ ON DELETE CASCADE  );  CREATE TRIGGER **profileConstrain**  BEFORE INSERT ON Profiles  FOR EACH ROW  DECLARE  profileNum INTEGER;  Too\_many EXCEPTION;  BEGIN  SELECT COUNT(\*) INTO profileNum  FROM Profiles  WHERE member\_id = :NEW.member\_id;  IF profilenum > 4 THEN  RAISE Too\_many;  END IF;  EXCEPTION  WHEN Too\_many THEN  RAISE\_APPLICATION\_ERROR(-20000, 'You can have up to 5 profiles!');  END profileConstrain;  /  CREATE TABLE **Genre**  (movie\_id CHAR(6),  genre\_name VARCHAR(50),  PRIMARY KEY(movie\_id, genre\_name),  FOREIGN KEY(movie\_id)REFERENCES Movie ON DELETE CASCADE  );  CREATE TABLE **View\_**  (movie\_id CHAR(6),  member\_id CHAR(6),  profile\_name VARCHAR(50),  rating FLOAT CHECK(rating < 6 AND rating >= 1),  PRIMARY KEY(movie\_id, member\_id, profile\_name),  FOREIGN KEY(movie\_id) REFERENCES Movie,  FOREIGN KEY (member\_id, profile\_name) REFERENCES Profiles  );  CREATE TABLE **Casting**  (movie\_id CHAR(6),  actor\_id CHAR(6),  PRIMARY KEY(movie\_id, actor\_id),  FOREIGN KEY(movie\_id) REFERENCES Movie,  FOREIGN KEY (actor\_id) REFERENCES Actor  ); |

**JDBC Implementation**

This database management systema software is designed to define, manipulate, retrieve, and manage data on all relations based on the project requirement. The design resembles the suggested schema on the instruction, but it is not 100% similar.

All parts work as intended based on my understanding of the requirements. However, there is room for improvement. For example when listing table content the application does not list the columns title and more user input validation is necessary. Extra features such as search movie by genre would also complement this application.

Please read README.txt for description on how to run the program.

**Contributions:**

This project was 100% designed and implemented by Erik F Garcia