****

**Texas A&M University**

**Computer Science & Engineering Department**

**Database System Course Project1**

**Movie Website**

**Name: Li-Hen Chen**

**UIN: 928003907**

**Part A. E/R Diagram**

This is a movie website using Python/ Django as backend framework and SQLite as database. This movie website is a website allows users to search the movies or actors, saving movies they have seen, displaying the movie and its actors and also displaying the actors and link to the movie they act. This website includes following features:

* Search Engine including inverted index and LRU cache
* User Management
* Login System

**A.1 Data description**

5000+ movie\_ids from Kaggle movie dataset are the origin data source (<https://www.kaggle.com/oxanozaep/imdb-eda/data>). With these movie\_ids, I utilized a python to collect movie images url, actors, movie title and movie description of data from IMDB movie website. Currently, there are about 3000 movies in the database. In order to stimulate the real world, I randomly generate 1000 users whose usernames are composed of 6-12 characters only and whose passwords are composed of 8-16 combination of characters and digits. For each user, I random add 0-3 movies to the “movie\_seen” column for them and the result can be viewed in the website.

**A.2 Backend description**

As we all know, python is one of the most popular programming languages in the world now. Besides, I am also familiar with python the most. As a result, I chose python as the programming language for the project. The topic is movie website and there are lots of framework for building website in python. One of the well-known is Django. Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so we can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Another reason I chose Django as backend because it’s a MVC architecture. Model–View–Controller (MVC) is a software design pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the ways information is presented to and accepted from the user. Following the MVC architectural pattern decouples these major components allowing for code reuse and parallel development. It really fits in our project because it can interact with database easily even without any SQL syntax.



**A.3 Database description**

The database I chose is SQLite in this project because I consider this project is not a large-scale application and SQLite can help us understand what relational database does. SQLite is based on the relational database management system model. RDBMS are comprised of columns and rows to store data within a structured format. This subset of DBMS are deemed relational since the values within the table are related to one another. Furthermore, tables themselves can also be related.

SQLite is a serverless solution that’s self-contained, highly reliable, and full of features. As a self-contained SQL database engine, it can handle all sorts of data in a relatively simple manner. Due to it is “Lite”, it has some pros and cons.

Pros

* Serverless which means it is simple to set up and zero configuration is required
* File-based system makes it very portable
* Great for development and testing

Cons

* Doesn’t provide network access
* Not built for large-scale applications
* No user management

**A.4 E/R Diagram**

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education and research.

In this project, we have five entities which are User, Movie, Actor, Admin log, Auth Permission.

User entity has 10 attributes including ID, is\_active, username, password, date joined, last login, is\_superuser, email, first name, last name. In this entity “ID” is unique value while others are not.

Movie entity has 9 attributes including movie including MovieID, title, year, length, genres, rate, poster, plot, trailer. In this entity “MovieID” is unique value while others are not. Actually, the poster and trailer attributes are the string urls which link to the corresponding website.

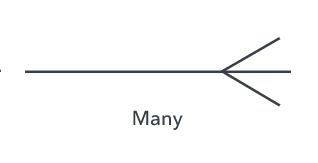
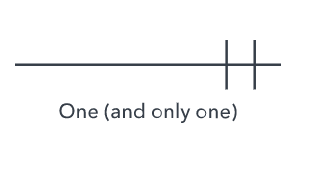
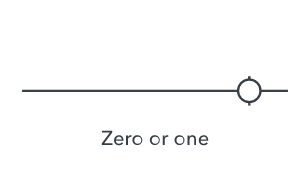
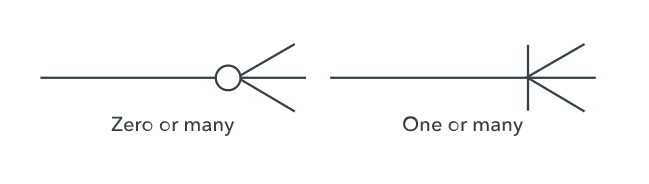
Actor entity has relatively less attributes including ActorID, name and photo. In this entity “Actor” is unique value while others are not. The photo attribute is the string url which link to the webpage of actor photo.

Admin log entity has 7 attributes including LogID, Action time, ObjectID, Change\_message, content\_type\_ID, UserID and Action flag. In this entity, “LogID” is unique value while others are not.

Auth Permission entity has 4 attributes including AuthID, contentID, code name, description. In this entity, “AuthID’ is unique value while others are not.

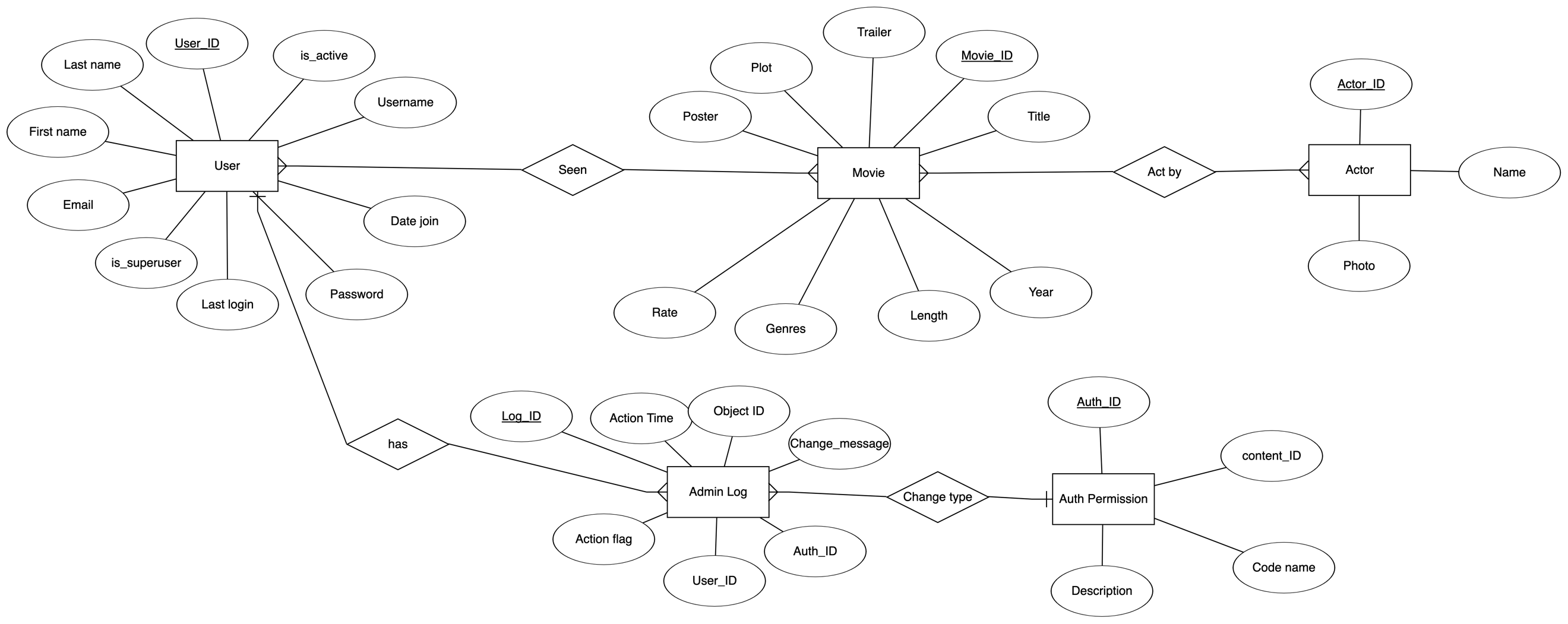
Entity

Relation

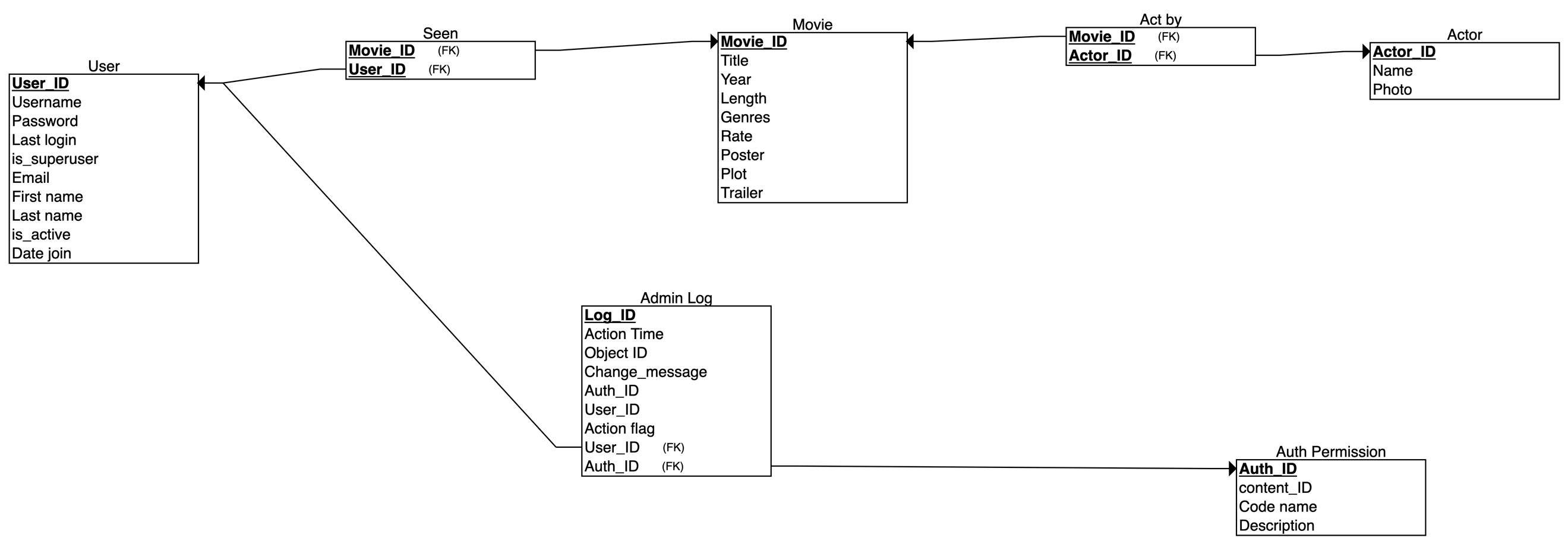
    

The three main cardinal relations in this ER diagram are one-to-one, one-to-many, and many-many. We can use the above fig. to represent our ER diagram and combine them into ER diagram in our case.

A weak entity is one that can only exist when owned by another one. In our case, we don’t have a weak entity so we don’t have to deal with it. The first relation is a User “Seen” many Movie which is many to many relations. A user may have seen many movies and a movie may have been seen by many users. That’s the reason that the relation we choose is many-to-many. The second relation is that a movie may acted by many actors and an actor may act on many movies. That’s the reason we choose many-to-many relation at this case. The third relation is the relation between User and Admin Log. The User may have many Admin Logs when the user is an admin. On the other hand, each Admin Log only has only one User. As a result, it’s many-to-one relation. The last relation which is called “change type” is between Admin log and Auth Permission. The Auth Permission may be changed by many Admin Logs. On the other hand, each Admin Log only has only one Auth Permission change. The following figure is how the ER diagram is presented in our movie website project.

****

**Part B. Relational Schema**

****

**Part C. Creating Your Database using SQL**

As mentioned above, the database I chose is SQLite because I consider this project is not a large-scale application and SQLite can help us understand what relational database does. If a person would like to look into what is going on in the database, she can use a software called DB browser for SQLite which is a software providing user interface to look up, manipulate the database we built. The database for the website is “movie.db” file.

There are 2 requirements and 2 important points to keep in mind

• At least two relations with thousands of tuples

• At least one relation with hundreds of tuples

• Make sure not to generate duplicate values for key attributes.

• Your database almost certainly includes relations that are expected to join with each other.

For the requirements, I use Kaggle IMDB dataset which contains more than 3000 movies and more than 5000 actors in the database. In the many-to-many relation “Movie\_Actor”, contains more than 20000 relations. Besides, I randomly generate 1000 users whose usernames are composed of 6-12 characters only and whose passwords are composed of 8-16 combination of characters and digits. For each user, I random add 0-3 movies to the “movie\_seen” column for them and the result can be viewed in the website. In the “movie\_seen” relationship, we have total 1500 relations by randomly generation.

For the important points to keep in mind, the actors and movie from IMBD database has its own ID in each case which would not have duplicate values for key attributes. For the users I generated, I put all usernames in a “set” data structure in python. A “set” is an unordered collection of unique elements so it would definitely contain duplicate keys. Then with all these usernames I generated corresponding password for them. Unlike the username, password is not a key so it could be duplicate in different accounts.

**Part D. Developing a user interface**

This is a movie website for user to remember what they have seen to search for the movies and search for the actors. There are some operations what a user can do including the following things:

* Search for the movie by movie name. It was built by an inverted index structure with wildcard to enable vague search.
* Search for the actor by actor name. It was also built by an inverted index structure with wildcard to enable vague search.
* Create their own account
* Login and logout the website where cookies will record which user it is or just a visitor
* Add a “Seen” option to the movie they have seen before
* View the movies a user have seen before in his personal account webpage
* View the movie and its corresponding information in the page
* View the actor and its corresponding information in the page

Finally, a website should be deployed in order to be accessed by everyone who wants to visit so I took the suggestion on the project description: “you may upload your webpages to cloud service HEROKU and provide a link to the website in your project report.” I deployed my movie website on the HEROKU server and I will provide the link in Part E. I have uploaded my python source code on my personal github, I will also provide my github link to the project in Part E.

**Part E. Project Submitting**