Business Data Management

Rotten Tomato:

Tingyu Chen, Tong Wang, Naixuan Ma

Yang Zhang, Yubo Qin, Lufan Zhang

Dec 8th, 2014

Table of Contents

1 Introduction

Basically, our team develops a web application that includes a search page allowing user search movies stored in the database. All matched movies will be listed in the webpage and corresponding movie information including title, year, runtime, director, actors, plot, poster, IMDB rating, Tomato rating, fresh tomato and rotten tomato is displayed.

This web application is composed of two parts as database design and webpage design. For database design, we follow the order of write problem description, create ER diagram, produce relational schema, normalize schema and write SQL code to create schema. The final database has 6 tables including MOVIE, ACTOR, ACT, DIRECT and GENRE. Then, we download more than 200 movies from IMDB API. Since downloaded data is in JSON format, we clean the data and store them into each table respectively.

For webpage design, the main challenge is how to query the database by using the input keywords from user. Our webpage accepts at most three keywords from user input. For each keyword, we search in MOVIE.title, MOVIE.year, ACTOR.name and GENRE.genre. All matched records are joined together and duplicates are eliminated. Finally, we display the matched records below the searching box.

Following parts introduce the detailed implementation of our web application design.

2 Business Problem

Movies are becoming more and more popular these days. As a movie comes out, people tend to search it online before go to cinema. After watching the movie, people tend to comment and score on that movie. Anyhow, database design plays important roles in all of above.

In such trend, why not build a web application that allows user search movies and check the score on that movie. In term of movie score, IMDB rating is rated by critics and Rotten Tomato rating is rated by internet users. If users are interested in a movie or a set of movies, they can search and check the scores.

Therefore, we decide to design a web application including movie database that allows user search movies and check movie IMDB rating and Rotten Tomato rating, and corresponding movie title, year, runtime, director, actors, plot, poster information.

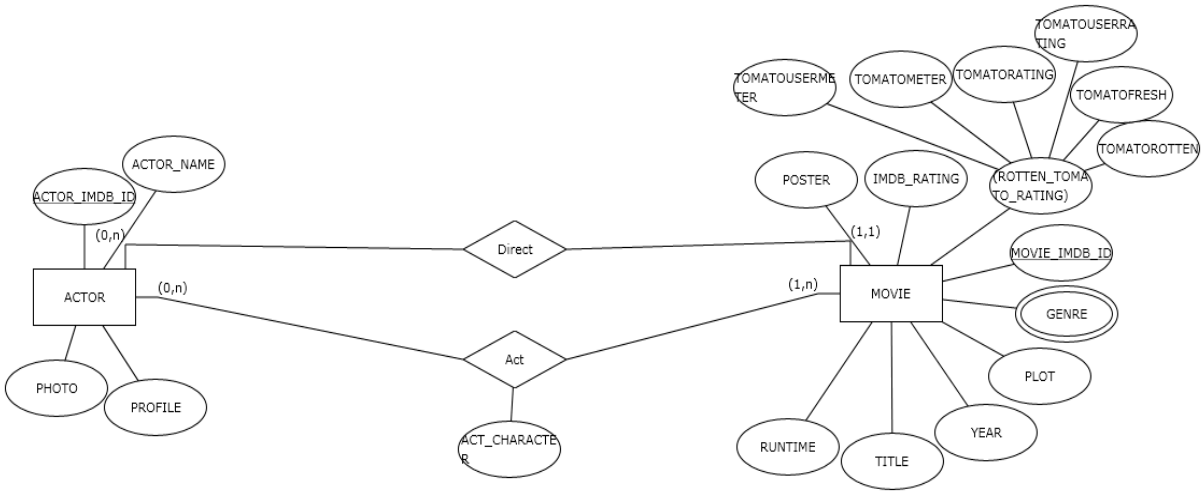
3 Database Design

3.1 Business Requirements

This is a movie database. Each movie has title, year, genres, runtime, director, actors, plot, poster, IMDB rating and Rotten Tomatoes ratings. One movie can have multiple genres and actors, but only one director. Director can also be actor. Actor can take part in different movies. Rotten Tomatoes ratings include tomato meter, tomato rating, tomato fresh, tomato rotten, tomato user meter, tomato user rating.

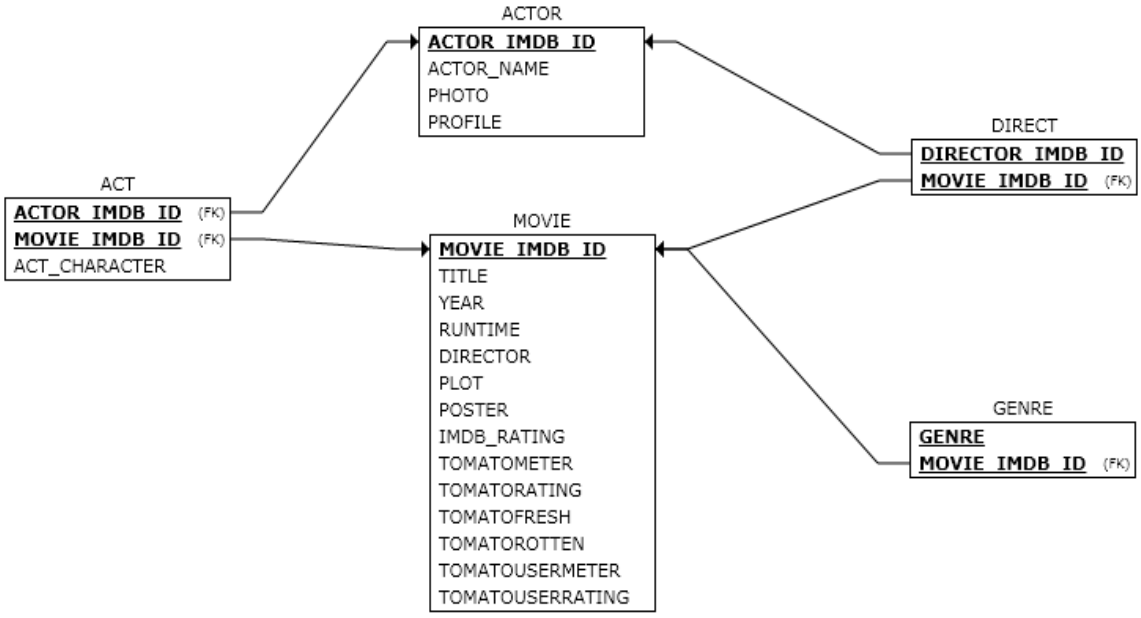
3.2 Entity Relationship Diagram

After analyzing the business requirements, we come up with two entities: ACTOR, MOVIE, and two relationships: Direct, meaning that director directs one movie; Act, meaning that actors act movies. So the Entity Relationship diagram as followed.



3.3 Relational Schema

After mapping the above ER Diagram into relational schema, our relational schema as followed.



3.4 Normalization

1. Because all the relations have no multivalued attributes or nested relations, so the above schemas are in the 1NF.

2. For ACTOR and MOVIE relations, their primary key only contains single attribute, so they are in 2NF. For DIRECT AND GENRE relations, beside primary key, they don't contains any non-key attributes. so ACT and GENRE are also in 2NF. And for the last relation ACT, although it contains non-key attribute - ACT_CHARACTER, this non-key attribute is functionally dependent on the primary key, not a part of primary key, so ACT relation is also in 2NF.

3. For all the relations, there is no non-key attribute functionally determined by another non-key attribute, so they are all in 3NF.

After normalization, the relational schema is still as same as previous design.

3.5 SQL Code for database creation

Detailed SQL code is in ../BDM Project/Database Design/Database_SQL.

Based on relational schema, we create 6 tables as MOVIE, ACTOR, ACT, DIRECT and GENRE.

```
CREATE DATABASE ROTTEN_TOMATO;

CREATE TABLE ACTOR
    (ACTOR_IMDB_ID VARCHAR(20) NOT NULL,
```

```
    ACTOR_NAME VARCHAR(30) NOT NULL,
    PHOTO VARCHAR(200),
    PROFILE VARCHAR(100),
    PRIMARY KEY (ACTOR_IMDB_ID));

CREATE TABLE MOVIE
   ( MOVIE_IMDB_ID VARCHAR(20) NOT NULL,
    TITLE VARCHAR(30) NOT NULL,
    YEAR INT,
    RUNTIME VARCHAR(20),
    DIRECTOR VARCHAR(30),     %redundancy for db, but make query fast
    ACTORS VARCHAR(100),      %comma separated string including main actors
    PLOT VARCHAR(500),
    POSTER VARCHAR(100),
    IMDB_RATING FLOAT,
    TOMATOMETER FLOAT,
    TOMATORATING FLOAT,
    TOMATOFRESH FLOAT,
    TOMATOROTTEN FLOAT,
    TOMATOUSERMETER FLOAT,
    TOMATOUSERRATING FLOAT,
    PRIMARY KEY (MOVIE_IMDB_ID));

CREATE TABLE ACT
   ( MOVIE_IMDB_ID VARCHAR(20) NOT NULL,
    ACTOR_IMDB_ID VARCHAR(20) NOT NULL,
    ACT_CHARACTER VARCHAR(30),
    PRIMARY KEY (MOVIE_IMDB_ID, ACTOR_IMDB_ID),
    FOREIGN KEY (MOVIE_IMDB_ID) REFERENCES MOVIE(MOVIE_IMDB_ID),
    FOREIGN KEY (ACTOR_IMDB_ID) REFERENCES ACTOR(ACTOR_IMDB_ID));

CREATE TABLE DIRECT
   ( MOVIE_IMDB_ID VARCHAR(20) NOT NULL,
    DIRECTOR_IMDB_ID VARCHAR(20) NOT NULL,
    PRIMARY KEY (MOVIE_IMDB_ID, DIRECTOR_IMDB_ID),
    FOREIGN KEY (MOVIE_IMDB_ID) REFERENCES MOVIE(MOVIE_IMDB_ID),
    FOREIGN KEY (DIRECTOR_IMDB_ID) REFERENCES ACTOR(ACTOR_IMDB_ID));

CREATE TABLE GENRE
   ( MOVIE_IMDB_ID VARCHAR(20) NOT NULL,
    GENRE VARCHAR(20) NOT NULL,
    PRIMARY KEY (Genre, MOVIE_IMDB_ID),
    FOREIGN KEY (MOVIE_IMDB_ID) REFERENCES MOVIE(MOVIE_IMDB_ID));
```

3.6 Database Storage

We download movie data from 2 API as API_1 http://www.myapifilms.com and API_2 http://www.omdbapi.com. API_1 is used to download data for ACTOR, ACT, DIRECT, GENRE, and API_2 is used to download data for MOVIE table. For each table, we write PHP code to download and insert into database. When downloading, we need to pay attention to the download order because of the foreign key constraint. Details about how to download data, please refer ../BDM Project/Database Design/properties.php. Finally, we store 219 movies in MOVIE table, 1255 actors in ACTOR table, 1510 records in ACT table, 109 direct info in DIRECT table (should be 219 but some director info is not exist from API) and 374 records in GENRE table.

3.7 Database Design Directory

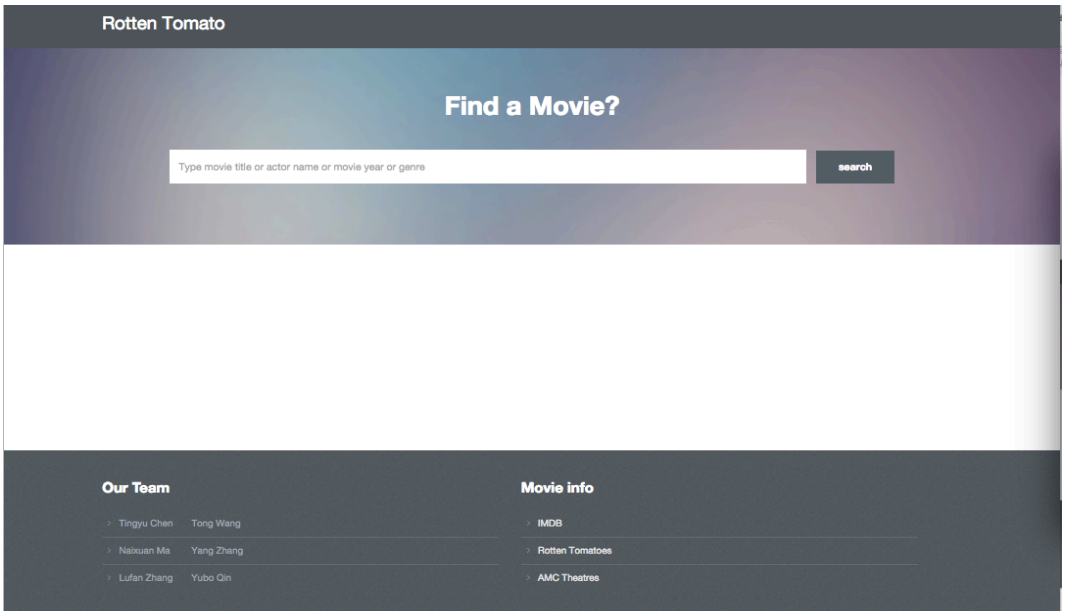
1. properties.php configure what movies to download.
2. database_connection.php configures local database connection.
3. API_1_act.php, API_1_actor.php, API_1_direct.php, API_1_genre.php, API_2_movie.php configure how to download from API and insert into corresponding table.
4. download_act.php, download_actor.php, download_direct.php, download_genre.php are used to start download for each table.
5. Database_SQL is database SQL creation code.
6. Rotten_Tomato.sql is dump database and can be imported.

4 Webpage Design

4.1 Webpage Interface

We design the following webpage. “Rotten Tomato” is our team name. There’s a search box on the page. User can type keywords in the search box and click “search” button to review the search results. Attention, user shouldn’t click “enter” on the keyboard to replace clicking “search” because we use “search” button to trigger the matched records displayed on the same page rather than open a new page. In short, click “search” button enables the usage of AJAX technique.

Besides, our application backend accept at most 3 keywords from user inputs. If user input more than 3 keywords, only the front 3 keywords will be used to find matched records. For each keyword, we search in MOVIE.title, MOVIE.year, ACTOR.name and GENRE.genre. All matched records are joined together and duplicates are eliminated. Finally, we display the matched records below the searching box.



4.2 Query Technique For Search Box

Search function and display function are in backend controller ../BDM Project/Web Design/controller.php. We first trim the user inputs and extract first 3 keywords. Then perform search in database. We extract 2 main SQL query statements to explain query technique.

1. SQL1

SELECT **DISTINCT** MOVIE.MOVIE_IMDB_ID, MOVIE.TITLE, MOVIE.YEAR, MOVIE.RUNTIME, MOVIE.DIRECTOR, MOVIE.ACTORS, MOVIE.PLOT, MOVIE.POSTER, MOVIE.IMDB_RATING, MOVIE.TOMATOMETER, MOVIE.TOMATORATING, MOVIE.TOMATOFRESH, MOVIE.TOMATOROTTEN, MOVIE.TOMATOUSERMETER, MOVIE.TOMATOUSERRATING
FROM MOVIE **LEFT JOIN** GENRE **ON** MOVIE.MOVIE_IMDB_ID=GENRE.MOVIE_IMDB_ID
WHERE MOVIE.TITLE **REGEXP** 'input1' **OR** MOVIE.YEAR REGEXP 'input1' **OR** GENRE.GENRE **REGEXP** 'input1' **OR** MOVIE.TITLE REGEXP 'input2' **OR** MOVIE.YEAR **REGEXP** 'input2' **OR** GENRE.GENRE **REGEXP** 'input2' **OR** MOVIE.TITLE REGEXP 'input3' **OR** MOVIE.YEAR **REGEXP** 'input3' **OR** GENRE.GENRE **REGEXP** 'input3'

This SQL statement is used to search 3 user inputs respectively in MOVIE.title, MOVIE.year and GENRE.genre using regular expression. We use left join GENRE to avoid searching again so that raise time, but therefore produce duplicates. So we select distinct records.

2. SQL2

SELECT DISTINCT M.MOVIE_IMDB_ID, M.TITLE, M.YEAR, M.RUNTIME, M.DIRECTOR, M.ACTORS, M.PLOT, M.POSTER, M.IMDB_RATING, M.TOMATOMETER, M.TOMATORATING, M.TOMATOFRESH, M.TOMATOROTTEN, M.TOMATOUSERMETER, M.TOMATOUSERRATING
FROM MOVIE M, ACTOR A, ACT

WHERE A.ACTOR_NAME **REGEXP** 'input1' **AND** M.TITLE **NOT REGEXP** ' input1' **AND** M.YEAR **NOT REGEXP** ' input1' ACT.MOVIE_IMDB_ID=M.MOVIE_IMDB_ID **AND** ACT.ACTOR_IMDB_ID=A.ACTOR_IMDB_ID **AND** ((A.ACTOR_NAME **REGEXP** '%s' **AND** M.TITLE **NOT REGEXP** '%s' **AND** M.YEAR **NOT REGEXP** '%s') **OR** (A.ACTOR_NAME **REGEXP** '%s' **AND** M.TITLE **NOT REGEXP** '%s' **AND** M.YEAR **NOT REGEXP** '%s') **OR** (A.ACTOR_NAME **REGEXP** '%s' **AND** M.TITLE **NOT REGEXP** '%s' **AND** M.YEAR **NOT REGEXP** '%s'))

This statement is used to search 3 user input in ACT. ACTOR_NAME and eliminate duplicates which created from the previous statement.

3. Optimization
Considering the SQL2 is too complicated, we can optimize SQL statements by join ACTOR table and ACT table in SQL1, and search actor names in SQL so that avoid eliminating duplicates.

4.3 Web Search Demo

Please open ../BDM Project/Rotten Tomato Demo.mp4 and watch the search demo.

4.4 Web Design Directory

1. database_connection.php configure local database connection.
2. controller.php is the controller of web app. It receives user inputs and search in database, then display matched records.
3. page.html specifies interface design.

5 How To Use

Open ../BDM Project/ReadMe.txt, and it includes all detailed information about how to import database and how to use.

6 Individual Contribution

1. Database Design (Business Requirements, ER Diagram, Relational Schema, Normalization, Schema SQL):

Tingyu Chen, Tong Wang, Naixuan Ma, Yang Zhang, Yubo Qin, Lufan Zhang **ALL** together.

2. Database Storage:

MOVIE: Tingyu Chen, ACTOR: Tong Wang, ACT: Naixuan Ma, DIRECT: Yang Zhang, GENRE: Yubo Qin, Database Connection & Properties Setting: Lufan Zhang

3. Web Design:

Backend Query: Tingyu Chen, Naixuan Ma, Yubo Qin

Frontend interface: Tong Wang, Yang Zhang, Lufan Zhang