IMDB-Slim Framework application

by Xiongfeng(Kevin) Hu

1. Introduction

This IMDB App is developed using Slim Framework and uses MySQL for database.

The tools for development is as follows:

1) XAMPP
Provide PHP development environment - Apache Server.
2) Composer + Slim Framework
Dependency manager for PHP. We need this to install Slim Framework
3) RestEasy
Chrome Extension - help us to make http request.

2. Filesystem Layout

Here's the directory structure I use for my own Slim Framework applications:

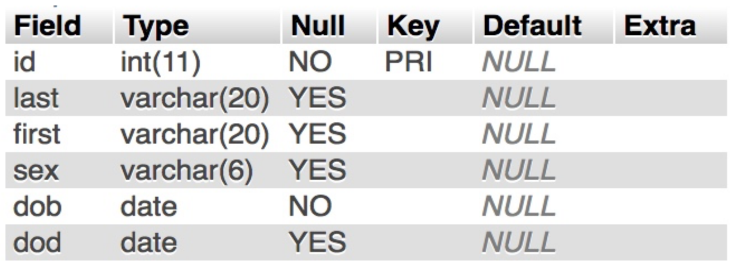
```
imdbapp/
        public/
                .htaccess
                index.php
        source/
                config/
                        db.php
                data/
                        actor1.csv
                        actor2.csv
                        actor3.csv
                        director.csv
                        movie.csv
                        movieactor1.csv
                        movieactor2.csv
                        moviedirector.csv
                        moviegenre.csv
                routes/
                        actors.php
                        directors.php
                        movies.php
                        search.php
                sql/
                        create.sql
                        drop.sql
                        load.sql
                        queries.sql
        vendor/
        readme.txt
```

3. DB Design - Tables in database imdbapp

1) The Actor Table

This table describes information regarding actors and actresses of movies. It specifies an identification number unique to all people (which is shared between actors and directors), the last name of the person, the first name of the person, the sex of the person, the date of birth of the person, and the date of death of the person if applicable. The schema of the Actor table is given as follow:

Actor(id, last, first, sex, dob, dod)

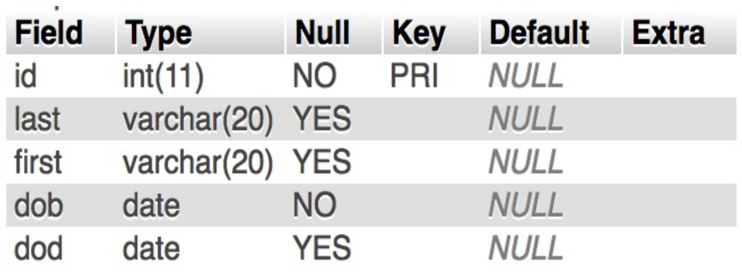


Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	*NULL*	
last	varchar(20)	YES		*NULL*	
first	varchar(20)	YES		*NULL*	
sex	varchar(6)	YES		*NULL*	
dob	date	NO		*NULL*	
dod	date	YES		*NULL*	

2) The Director Table

It describes information regarding directors of movies. It specifies an identification number of the director, the last name of the director, the first name of the director, the date of birth of the director, and the date of death to the director if applicable. The schema of the Director table is given as follow:

Director(id, last, first, dob, dod)

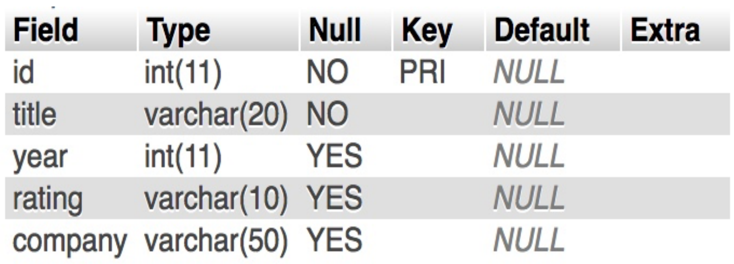


Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	*NULL*	
last	varchar(20)	YES		*NULL*	
first	varchar(20)	YES		*NULL*	
dob	date	NO		*NULL*	
dod	date	YES		*NULL*	

3) The Movie Table

This table describes information regarding movies in the database. It specifies an identification number unique to each movie, the title of the movie, the year the movie was released, the MPAA rating given to the movie, and the production company that produced the movie. The schema of the Movie table is given as follows:

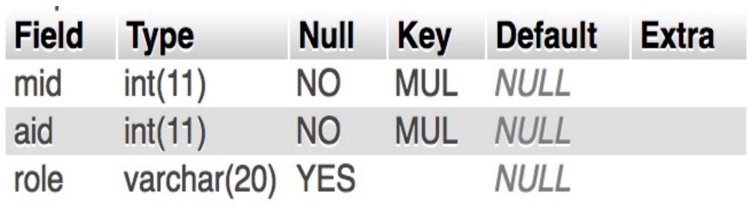
Movie(id, title, year, rating, company)



Field	Type	Null	Key	Default	Extra
id	int(11)	NO	PRI	*NULL*	
title	varchar(20)	NO		*NULL*	
year	int(11)	YES		*NULL*	
rating	varchar(10)	YES		*NULL*	
company	varchar(50)	YES		*NULL*	

4) The MovieActor Table
It describes information regarding the movie and the actor/actress of that movie. It specifies the identification number of a movie, and the identification number of the actor/actress of that movie. The schema of the MovieActor table is given as follow:

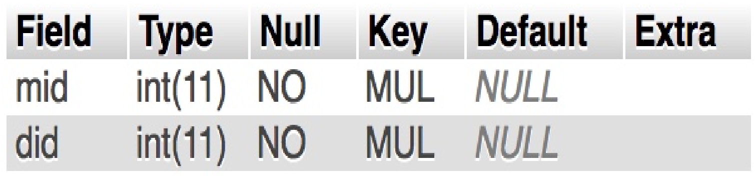
MovieActor(mid, aid, role)



Field	Type	Null	Key	Default	Extra
mid	int(11)	NO	MUL	*NULL*	
aid	int(11)	NO	MUL	*NULL*	
role	varchar(20)	YES		*NULL*	

5) The MovieDirector Table
It describes the information regarding the movie and the director of that movie. It specifies the identification number of a movie, and the identification number of the director of that movie. The schema of the MovieDirector table is given as follow:

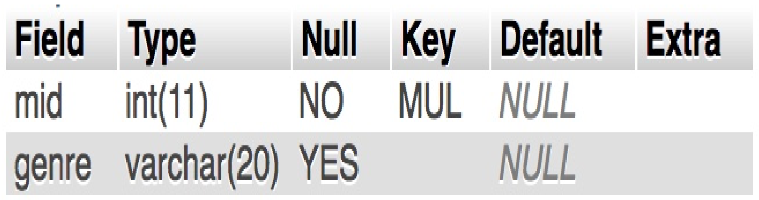
MovieDirector(mid, did)



Field	Type	Null	Key	Default	Extra
mid	int(11)	NO	MUL	*NULL*	
did	int(11)	NO	MUL	*NULL*	

6) The MovieGenre Table
It describes information regarding the genre of movies. It specifies the identification number of a movie, and the genre of that movie. The schema of the MovieGenre table is given as follow:

MovieGenre(mid, genre)

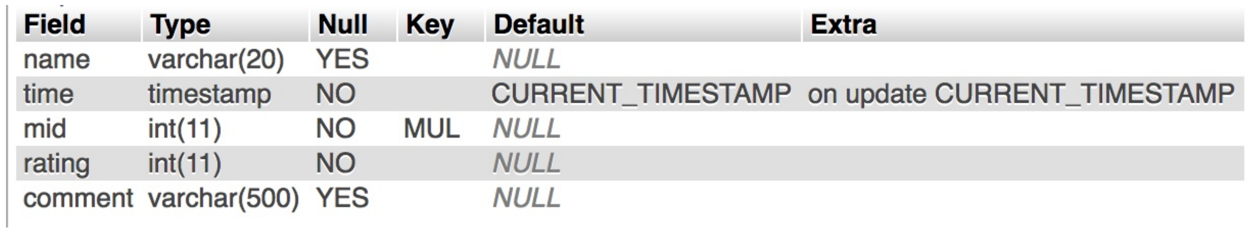


Field	Type	Null	Key	Default	Extra
mid	int(11)	NO	MUL	*NULL*	
genre	varchar(20)	YES		*NULL*	

7) The Review Table

The Review table stores the reviews added in by the users in the following schema:

Review(name, time, mid, rating, comment)

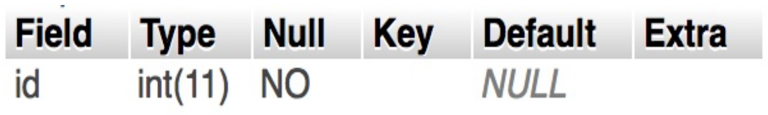


Field	Type	Null	Key	Default	Extra
name	varchar(20)	YES		*NULL*	
time	timestamp	NO		CURRENT_TIMESTAMP	on update CURRENT_TIMESTAMP
mid	int(11)	NO	MUL	*NULL*	
rating	int(11)	NO		*NULL*	
comment	varchar(500)	YES		*NULL*	

8) MaxPersonID and MaxMovieID

Once a user adds a new actor/director, the system should assign a new ID to the actor/director and insert a tuple to the Actor/Director table. Similarly, system should assign a new ID to a new movie. In order to assign a new ID to, say, an actor/director, the system has to remember what was the largest ID that it assigned to a person in the last insertion. The MaxPersonID table is used for this purpose, which has the following schema:

MaxPersonID(id)
MaxMovieID(id)



Field	Type	Null	Key	Default	Extra
id	int(11)	NO		*NULL*	

4. RESTful API Design

1) http://localhost/imdbapp/public/api/actors
   GET - Get all actors.
2) http://localhost/imdbapp/public/api/actor/{id}
   GET - Get a single actor by id.
3) http://localhost/imdbapp/public/api/actor/add
   POST - Add a new actor.
4) http://localhost/imdbapp/public/api/actor/{aid}/movie/add
   POST - Add movies to an existing actor.
5) http://localhost/imdbapp/public/api/actor/update/{id}
   PUT - Update an existing actor by id.

6) http://localhost/imdbapp/public/api/actor/delete/{id}
DELETE - Delete an existing actor by id.

7) http://localhost/imdbapp/public/api/directors
GET - Get all directors.

8) http://localhost/imdbapp/public/api/director/{id}
GET - Get a single director by id.

9) http://localhost/imdbapp/public/api/director/add
POST - Add a new director.

10) http://localhost/imdbapp/public/api/director/{did}/movie/add
POST - Add movies to an existing director.

11) http://localhost/imdbapp/public/api/director/update/{id}
PUT - Update an existing director by id.

12) http://localhost/imdbapp/public/api/director/delete/{id}
DELETE - Delete an existing director by id.

13) http://localhost/imdbapp/public/api/movies
GET - Get all movies.

14) http://localhost/imdbapp/public/api/movie/{id}
GET - Get a single movie by id.

15) http://localhost/imdbapp/public/api/movie/add
POST - Add a new movie.

16) http://localhost/imdbapp/public/api/movie/{id}/review/add
POST - Add user review to an existing movie.

17) http://localhost/imdbapp/public/api/movie/update/{id}
PUT - Update an existing movie by id.

18) http://localhost/imdbapp/public/api/movie/delete/{id}
DELETE - Delete an existing movie by id.

19) http://localhost/imdbapp/public/api/search/{name}
GET - Search for all movies a single person is involved in by his/her name.

5. SQL statement to pull all movies that Clint Eastwood took part in

```
SELECT M.title
     FROM  Movie M, MovieActor MA, Actor A
WHERE  M.id = MA.mid
     AND MA.aid = A.id
     AND A.last = 'Eastwood'
     AND A.first = 'Clint'
UNION
SELECT M.title
     FROM  Movie M, MovieDirector MD,Director D
```

```
WHERE  M.id = MD.mid
AND MD.did = D.id
AND D.last = 'Eastwood'
AND D.first = 'Clint';
```

The above sql statement is in **queries.sql.**

6. Sample API call to pull all movies that Clint Eastwood took part in

http://localhost/imdbapp/public/api/search/Clint Eastwood