Northeastern University

Project Final Report: Searchable Movie Database

Matt Morgan & Erica Yee

CS3200 Database Design

Professor Kathleen Durant

April 19, 2017

README

First, load up MySQL Workbench and and run the existing Initialize.sql script to create the database. Then, run the import scripts in the import folder in the following order: movies, people, studios, directs, actsin, produces, makes. This will ensure that foreign key constraints are not broke as the data in imported. Once the database is filled with the preliminary data from the Stanford InfoLab, you're ready to connect the front end.

First you need to make sure you have php installed. Instructions on installation can be found here: http://php.net/manual/en/install.php. Next, you might need to go through the php files themselves and change some of the server information to match the criteria on your machine. Right now the username (DB_USER) and password (DB_PASSWORD) are set to "root", and the host server (DB_HOST) is set to "127.0.0.1", these are the default values for mysql, but they are different on your machine please change in the following files in the webdocs directory:

Mysql_connect.php
Fetchmovie.php
Fetchperson.php
Fetchstudio.php

You're now ready to get the php server running and start using the front end! From your console, cd into the webdocs directory. Then run the following command to start a local php server: php -S localhost:8080 . Then in your browser navigate to localhost:8080, and the front end should be up and running!

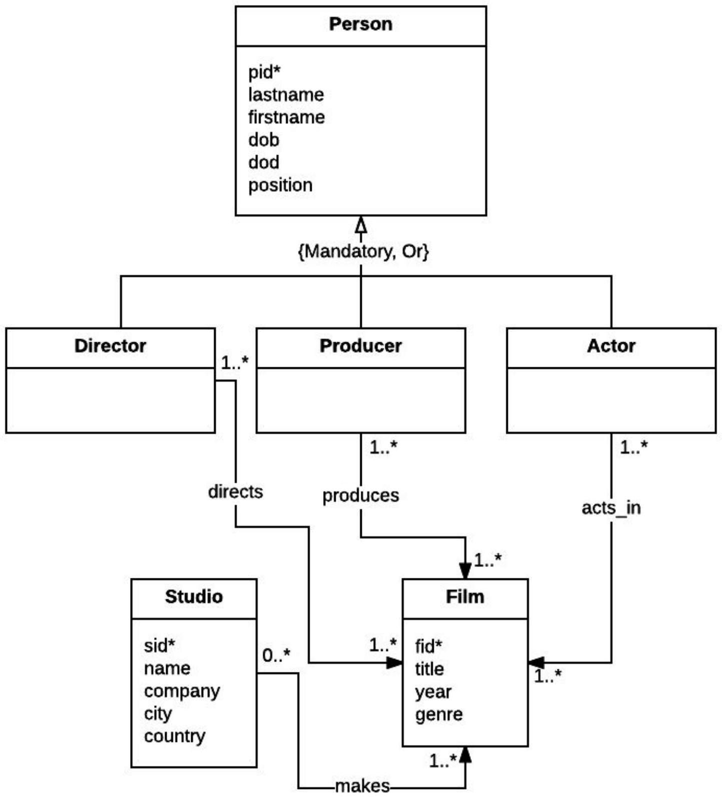
Introduction

Our project is a searchable movie database. Movie data are conducive to use in a relational database because each movie is also connected to other entities with their own fields, such as people and studios. After exploring the available data, we decided to narrow our scope for this project to just movies made in the 1980s, which still gave us plenty of data to work with (around 800 unique movies, 3000 people).

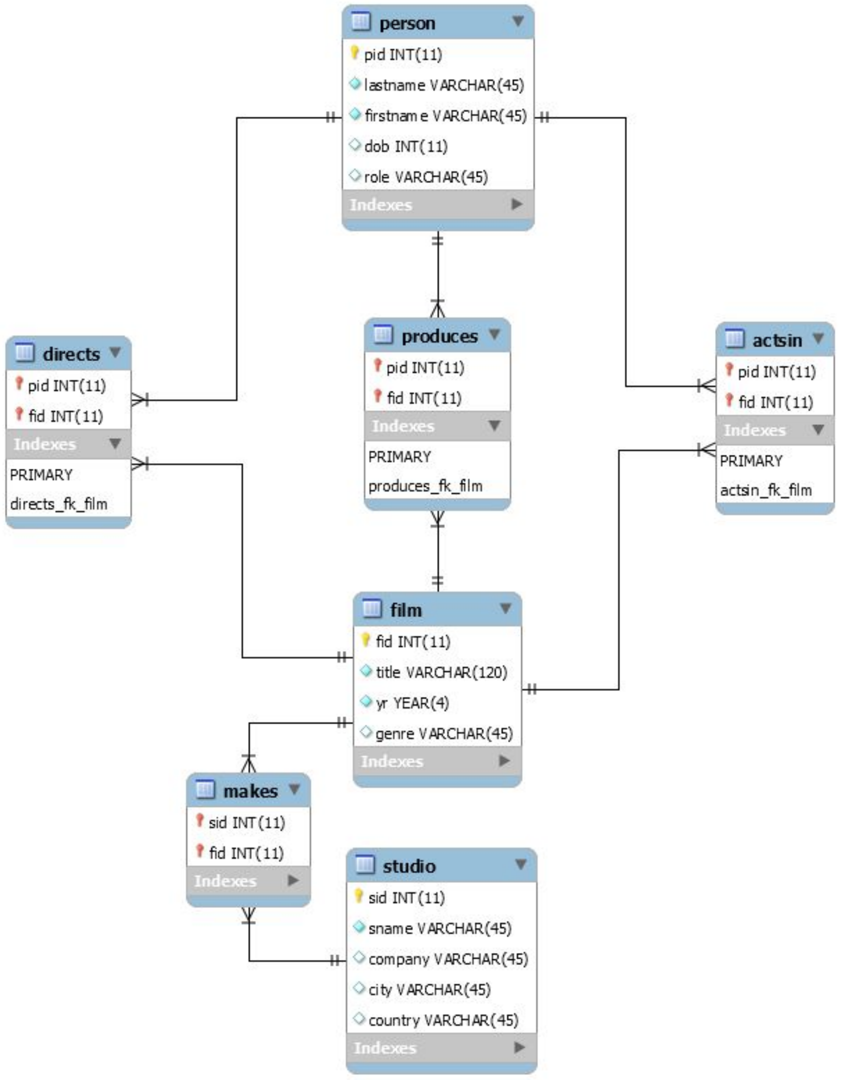
Technical Specifications

The data was scraped from the Stanford InfoLab website, documentation here. Some data is in XML format and some in HTML. We used online converters to get all the data in CSV files, then cleaned the data using Microsoft Excel. After standardizing the data, we used online converters again to format the data into SQL INSERT statements. Our database back-end is in SQL, built with MySQL workbench. The front end is web-based and uses html and javascript to provide a display to the user. We used the PHP language as the connector between the front and back end, since it allows us to connect to the sql server and perform queries directly from our webpage. For the read functionality, we used Jquery/Ajax to provide the user with a fast and responsive search for all the tables in the database.

UML Diagram


Person
pid*
lastname
firstname
dob
dod
position
{Mandatory, Or}
Director
Producer
Actor
1..*
1..*
1..*
directs
produces
acts_in
1..*
Studio
sid*
name
company
city
country
0..*
Film
fid*
title
year
genre
1..*
1..*
1..*
makes

EER Diagram



Final User Flow

From the homepage, the user can scroll down and see all the possible interactions via the front end. To add new rows to the database the user can click link to add a new film, studio, or person. They can also add new relationships between between movies, people and studios by adding a row to the directs, produces, makes, or acts-ins tables. All of these pages take user input from an HTML form and use php to send a Insert statement to the specified table. The page will specify if any fields are missing to complete the statement. The same actions are performed on the delete pages, but they send a Delete statement to the sql server. The Delete form only requires the id for the object, which can be found by searching the database.

From the homepage the user can also update the film, person, and studio tables. These pages also use a similar html form and require the id (PK) of the object being updated. The front end will notify the user if they are missing an id. Once the id is specified, the user can update one or all of the columns of a tuplet (besides the id).

Finally, the user can search the film, person, and studio tables. These links are also provided on the homepage. Once they have selected which table they want to search, the user can type text into the search bar and all columns are each tuplet will be searched. This search is especially useful if the user want to find the id of a certain object in order to update it. The search pages issues a SELECT * command on the specified table and uses ajax to immediately update the page with the selected tuplets.

Lessons Learned

We learned how to connect a SQL database to a website using PHP. We also learned a lot about data scraping and cleaning, since the data were originally not very standardized. It probably would have taken less time to import the original data straight into a NoSQL database. However, we believe the time we spent cleaning the data with =IF() and =VLOOKUP() formulas in Excel was worthwhile in order to produce a clean, easily searchable SQL database. If we ever make datasets we created available to the public, we will keep in mind how much we can help other users by making the data as clean as possible.

All code works.

Future Work

We will likely not use this specific database further in the future, but we will use the skills we gained from the project. This assignment was a good opportunity for us to learn how to create a complete application using a given set of data. The scope fit our skill levels, as well as provided challenges in data cleaning and connecting back-end/front-end. However, we do not see a pressing need for a movie database application with our functionality, since there are others (such as IMDB) already out there.