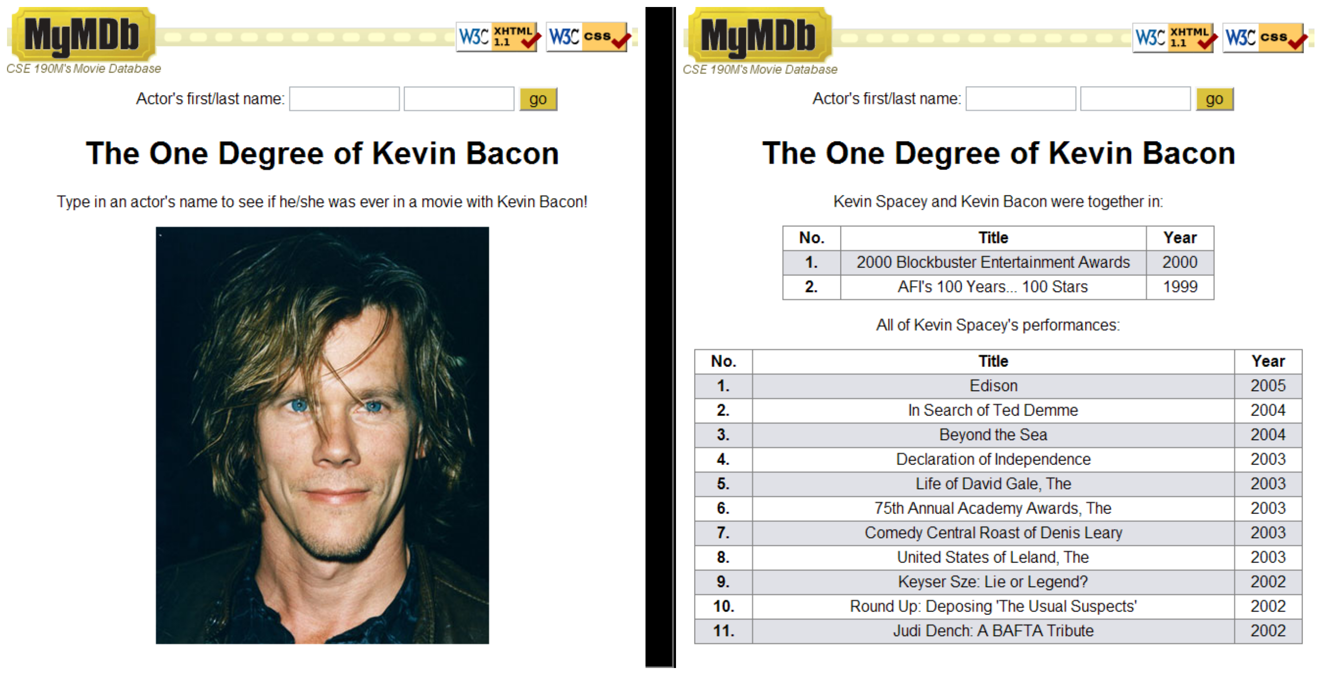
Internet Programming: Lab 7

This assignment asks you to interact with relational databases in PHP using SQL, as well as tying together the concepts taught throughout this course. You will write the following pages:



The One Degree of Kevin Bacon

Type in an actor's name to see if he/she was ever in a movie with Kevin Bacon!

The One Degree of Kevin Bacon

Kevin Spacey and Kevin Bacon were together in:

No.	Title	Year
1.	2000 Blockbuster Entertainment Awards	2000
2.	AFI's 100 Years... 100 Stars	1999

All of Kevin Spacey's performances:

No.	Title	Year
1.	Edison	2005
2.	In Search of Ted Demme	2004
3.	Beyond the Sea	2004
4.	Declaration of Independence	2003
5.	Life of David Gale, The	2003
6.	75th Annual Academy Awards, The	2003
7.	Comedy Central Roast of Denis Leary	2003
8.	United States of Leland, The	2003
9.	Keyser Sze: Lie or Legend?	2002
10.	Round Up: Deposing 'The Usual Suspects'	2002
11.	Judi Dench: A BAFTA Tribute	2002

The Six Degrees of Kevin Bacon is a game based upon the theory that every actor can be connected to actor Kevin Bacon by a chain of movies no more than 6 in length. (Most, but not all, can reach him in 6 steps. 12% of all actors cannot reach him at all.)

Your task for this assignment is to write the HTML and PHP code for a web site called *MyMDb* that mimics part of the popular IMDb movie database site. Your site will show the movies in which another actor has appeared with Kevin Bacon. The site will also show a list of all movies in which the other actor has appeared.

The front search page `mymdb.php` has a form where the user can type an actor's name. When the form is submitted, the results are shown on `search.php`. You will turn in the following files:

- `mymdb.php`, the front signup page (partial skeleton provided)
- `bacon.css`, the CSS styles for both pages
- `bacon.js` the JavaScript code for both pages
- `search.php`, the search results page
- `common.php`, any common code that is shared between pages *("optional")*

Since this is the last assignment of the quarter, one of its themes is to tie together much of the material you have learned. You will write some HTML/CSS and JavaScript and a large amount of PHP and SQL code.

If you prefer, you may use another actor of your choice rather than Kevin Bacon as the center point, so long as that actor is in our database and has a large number of connections to other actors.

Appearance Constraints (both pages):

Your `mymdb.php` and `search.php` must **both** match certain appearance criteria listed below. Beyond these, any other aspects of the page are up to you, so long as it does not conflict with what is required.

- The same title, and links to the same CSS and JavaScript resources.
- A "favorites icon". If you like, you may use the provided `mymdb_icon.gif` from the course web site.
- A prominently displayed MyMDB logo; if you, like you may use the provided one `mymdb.png`.
- The standard W3C validator and JSLint buttons as links to the corresponding sites.
- A common stylistic theme, and an overall non-trivial number of styles, such as fonts, colors, borders, and layout. As much as possible, set up your styles to look correct on a variety of systems.
- A form to type an actor's first/last name and search for matching actors in the database.
- A central area of results/info. In `mymdb.php` this area contains content of your choice, including at least one image of Kevin Bacon (or your central actor) and some text about the site. Be creative! In `search.php` this area contains the actor's movies and all movies in which the actor starred with Kevin Bacon.
- The `search.php` page should contain a link back to `mymdb.php` if the user wants to start over.

NOTE: With so much in common between the two pages, it is important to find ways to avoid redundancy. You should use the PHP `include` function with a shared common file included by both pages.

Front Page, `mymdb.php`:

The initial page, `mymdb.php`, allows the user to search for actors to match against Kevin Bacon. A skeleton is on the course web site; you may modify it in any way you like, subject to the constraints in this document.

The form on the page must contain two text boxes that allow the user to search for an actor by first/last name. The end goal is to submit to `search.php`, but it is possible that the name the user types (such as "Will Smith") will match more than one actor. Some names match no one in the database.

To address this problem, you can instead contact an intermediate instructor-provided web service to find out which actor matches a given name. You don't need to use our web service if you'd rather write that functionality yourself. But either way, your page needs to map a name to a single actor.

Actor ID Web Service, `actorid.php`:

Parameter name	Value
`first_name`	actor's first name (or partial first name) as a string, such as "William"
`last_name`	actor's last name (or partial first name) as a string, such as "Shatner"

The provided `actorid.php` maps names to IDs. For example, to search for partial names `"ad"` and `"pitt"`:

```
https://webster.cs.washington.edu/actorid.php?first_name=ad&last_name=pitt
```

This query would return the following XML output. Notice that it contains the actor id:

```
<actor id="376249" first_name="Brad" last_name="Pitt" appearances="59" />
```

The service issues a 404 error if no actor is found. By default, the service queries `imdb_small`. To make it query the full imdb, pass an additional query parameter named `imdb` set to any value. If you want to return all the actors that matched a given first/last name, pass a query parameter named `all` set to any value.

```
https://webster.cs.washington.edu/actorid.php?first_name=ad&last_name=pitt&imdb=true
```

Movie Search Page, `search.php`:

The `search.php` page performs two queries on the `imdb` database to show a given actor's movies. Query the database using PHP's `mysql_` functions. Connect to the database using your UW NetID as your user name, and the MySQL password that was emailed to you. (If you lost your password, email us.)

The database has the following relevant tables. (The `roles` table connects actors to movies.)

table	columns
`actors`	`id`, `first_name`, `last_name`, `gender`
`movies`	`id`, `name`, `year`
`roles`	`actor_id`, `movie_id`, `role`

Your page should perform the following two queries. For each query, you will need to use a **join** between several tables of the database.

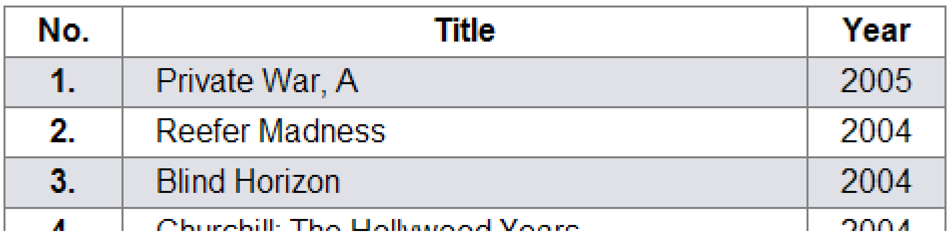
1. A query is to find a complete list of movies in which the actor has performed, displaying them in an HTML table. You may assume that any actor in the `actors` table has been in at least one movie.

   Hint: You will need to join the `actors`, `movies`, and `roles` tables, and only retain rows where the various IDs from the tables (actor ID, movie ID) match each other and the name or ID of your actor. Our solution's query joins 3 tables in the `FROM` clause and contains one condition in its `WHERE` clause.
2. A query to find all movies in which the actor performed with Kevin Bacon. These movies should be displayed as a second HTML table, with the same styling as the first. This is the hard query and should be done last. If the actor has not been in any movies with Kevin Bacon, don't show a table, and instead show a message such as, "Borat Sagdiyev wasn't in any films with Kevin Bacon."

   This query is bigger and tougher because you must link a pair of performances, one by the submitted actor and one by Kevin Bacon, that occurred in the same film. Do not directly write any actor's ID number anywhere in your PHP code, not even Kevin Bacon's.

   Hint: You will need to join a pair of actors (yours and Kevin Bacon), a corresponding pair of roles that match those actors, and a related movie that matches those two roles. Our query joins 5 tables in the `FROM` clause and contains 3 conditions in its `WHERE` clause.

The data in both tables should be sorted in descending order by year, breaking ties by movie title. The tables have three columns: A number for each movie, starting at 1; the title; and the year. The columns must have styled headings, such as bolded. The rows of the table must have alternating background colors, sometimes called "zebra striping." (Use PHP to apply styles to alternating rows.) For example:



No.	Title	Year
1.	Private War, A	2005
2.	Reefer Madness	2004
3.	Blind Horizon	2004
4.	Churchill: The Hollywood Years	2004

It is not acceptable to perform query filtering in PHP. Your SQL queries must filter the data down to only the relevant rows and columns. For example, a bad algorithm would be to query all of the actor's movies, then query all of Bacon's movies, then use PHP to loop over the two looking for matches. Each of the two queries above should be done with a single SQL query to the database.

Extra Feature:

Add at least one extra feature not listed among the requirements here. The following are suggestions:

1. **Select actor from list:** Once the user types the first/last name, if it matches more than one actor, let the user choose which one they meant. Use the `all` parameter to the web service previously described and present a list of possibilities, and let the user pick.
2. **Ajax search.php results:** Change `search.php` into a service that spits out a partial HTML page containing just the two tables described previously. Contact it using Ajax rather than by a normal form submission, and when it sends back its HTML response, inject this into the main `mymdb.php` page. If you do this feature, a few of the earlier Appearance bullets from ("same title/links", "link back to mymdb.php") no longer apply to `search.php` and should be ignored.

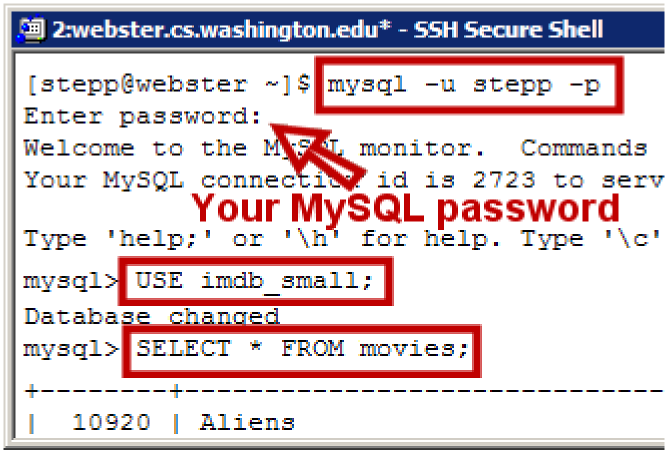
3. **Actor career stats:** Report information about the actor's career, such as their number of years they've been working in film (difference between first and most recent film), their average number of years between films, the percentage of their films in which they co-star with Kevin Bacon, and so on.
4. **Extended movie info:** In addition to the movie info listed previously, also show the genre and/or director name of each movie in which the actor appeared. You'll have to modify your SQL queries.
5. **Bling:** Add several Scriptaculous effects to your page to make it more awesome.

Developing Your Program:

Because the database is large and shared by all students, a bad query can hurt performance for everyone. We have a **smaller database `imdb_small`** with fewer records. While testing, please use that database and not the full `imdb`. When you think your code works, switch your PHP and/or JavaScript code to refer to `imdb`. We will post a "wall of shame" on the course web site. If you clog the server, you're going on the wall!

Use the **MySQL console** to develop your queries before writing PHP SQL code. Example tests are 376249 (Brad Pitt) or 770247 (Julia Roberts). If your query takes too long, press Ctrl-C to abort it.

Test the results returned by all PHP `mysql_` functions to spot query mistakes. Print out your SQL queries while debugging, so you can see the actual query your code is making. Many PHP SQL bugs come from improper SQL query syntax, such as missing quotes, improperly inserted variables, etc.



```
[stepp@webster ~]$ mysql -u stepp -p
Enter password:
Welcome to the MySQL monitor.  Commands
Your MySQL connection id is 2723 to serv

Type 'help;' or '\h' for help. Type '\c'
mysql> USE imdb_small;
Database changed
mysql> SELECT * FROM movies;
+--------+-----------------------------
|  10920 | Aliens
```