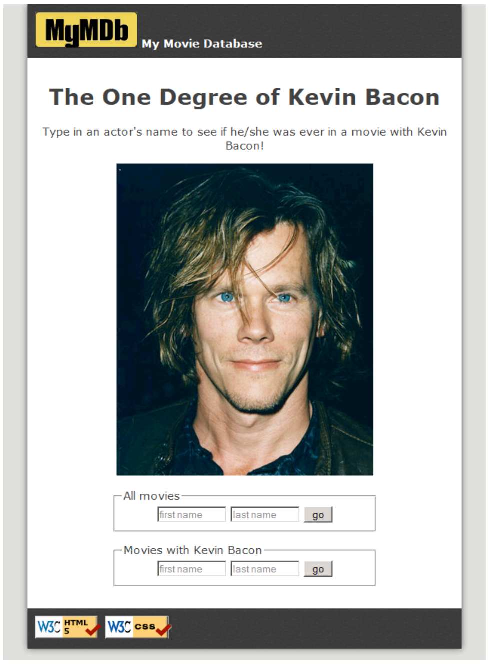
CS 325 - Introduction to Web Programming
JanPlan 2018

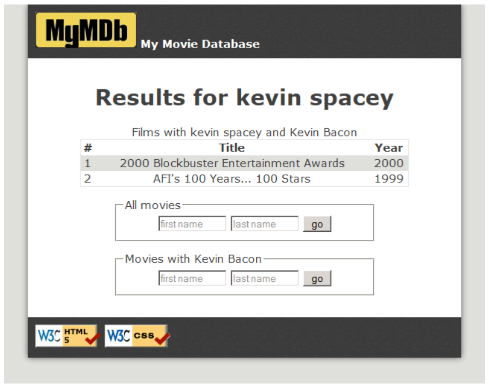
Project 4: Database Queries
Due: Tuesday, Jan 30, at 11:55pm

This assignment focuses on querying relational databases using SQL in PHP along with HTML/CSS.



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!



Results for kevin spacey

Films with kevin spacey and Kevin Bacon

#	Title	Year
1	2000 Blockbuster Entertainment Awards	2000
2	AFI's 100 Years... 100 Stars	1999

Background Information:

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/CSS 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.

(If you prefer, you may use another actor 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.)

The following files are provided to get you started:

- **top.html**, some common HTML that should appear at the beginning of every page
- **bottom.html**, some common HTML that should appear at the end of every page
- **index.php**, the initial front page that welcomes the user to the site
- the several image files and sample output files

You may need to change the path to images, etc. depending on where you store them locally.

The front search page **index.php** has two forms where the user can type an actor's name. The user can search for every film the actor has appeared in (which submits to **search-all.php**), or every film where both the actor and Kevin Bacon have appeared (which submits to **search-kevin.php**).

Here are files you must write:

- **search-all.php**, the page showing search results for all films by a given actor
- **search-kevin.php**, the page showing search results for all films with the given actor and Kevin Bacon
- **common.php**, any common code that is shared between pages
- **bacon.css**, the CSS styles shared by all pages

Front Page, index.php**:**

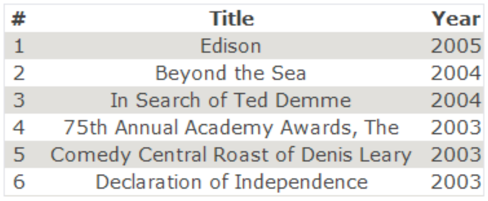
The initial page, **index.php**, allows the user to search for actors. This file is already provided to you; you may modify it in any way you like, or you may leave it as-is. If you modify it, indicate the changes you made in a README.txt file that you include in the project4 folder on the cs325 server. The forms on the page contain two text boxes that allow the user to search for an actor by first/last name.

firstname	for the actor's first name
lastname	for the actor's last name

Movie Search Pages, search-all.php and search-kevin.php:

The two search pages perform queries on the imdb database to show a given actor's movies. Query the database using PHP's PDO library as taught in class.

The data in both tables should be sorted by year descending, breaking ties by movie title ascending. The tables have three columns: A number starting at 1; the title; and the year. The columns must have styled headings, such as bold. The rows must have alternating background colors, called "zebra striping."



#	Title	Year
1	Edison	2005
2	Beyond the Sea	2004
3	In Search of Ted Demme	2004
4	75th Annual Academy Awards, The	2003
5	Comedy Central Roast of Denis Leary	2003
6	Declaration of Independence	2003

Database and Queries:

The database you are to use is the full imdb.sql. *Be sure to use the new version of the imdb database that I put on the Moodle page on Jan. 24.* It has the following relevant tables. (The roles table connects actors to movies.)

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

Your search pages perform the following queries. For some queries, you must use a **join** on several database tables.

1. search-all.php - List of all the actor's movies: A query to find a complete list of movies in which the actor has performed, showing them in an HTML table. If the actor doesn't exist in the database, don't show a table, and instead show a message such as, "Actor Borat Sagdiyev was not found." If the actor is found in the database, you may assume that any actor in the actors table has been in at least one movie.

2. search-kevin.php - List of movies with this actor and Kevin Bacon: 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. If the actor doesn't exist in the database, don't show a table, and instead show a message such as, "Actor Borat Sagdiyev was not found." 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."

3. both pages - Find the ID for a given actor's name: One thing that makes this program more complicated is the fact that some actors share the same name. The imdb data resolves this by giving them slightly different first names, such as "Will (I) Smith" vs. "Will (II) Smith". The user presumably doesn't know or understand this, so they will just type "Will Smith" and expect the program to do the right thing. But if your code naively searches for "Will Smith" in the database, it will not find any match.

To resolve this, you need a third query that searches for the best match for the actor's name that was typed by the user. This query finds and return the ID of the actor whose last name exactly matches what was typed by the user, and whose first name starts with the text typed by the user. If more than one such actor exists, an error should appear listing all the actors that matched and telling the user to type in one of those names. You can make the error appear however you want. For example, you could put an error message on a new page or you could use an alert box.

The behavior of the page is undefined if the actor being searched for is Kevin Bacon himself.

Appearance Constraints (all pages):

Your three PHP pages must 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 intention is to give you some flexibility to be creative with the appearance of your page, while also expecting you to practice some non-trivial CSS styling.

We'll mention our own page's styles, but you don't have to exactly match those as long as you follow these rules.

- All pages must begin with the content from **top.html** and end with the content from **bottom.html**, including the "favicon", MyMDb logo, W3C validator button links, and the two forms to search for movies. It is especially important not to modify the provided form query parameters' names such as first_name.
- The **main section of content** must be a centered area that is narrower than the overall page body. This main section should have a different background color than the overall body behind it, to make it stand out. *(Our page uses a width of 90% and a white background, atop a body with a background of #dad9d4.)*
- Every page should have a descriptive level-1 heading explaining the contents of the page.

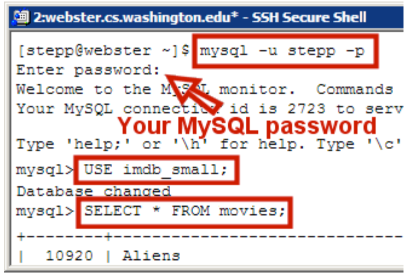
(Our pages have headings such as, "Results for Kevin Spacey" or "The One Degree of Kevin Bacon".)

- The top and bottom banner areas containing the MyMDb logo and W3C images should have a common color scheme and/or background image that make them stand out from other content on the page. *(Ours use a background image of **banner-background.png** and white text.)*
- The site should have a consistent **color and font scheme** used throughout. Your CSS should be structured such that it is easy to change the color/font scheme by modifying colors and fonts in a single place in the file. *(We use Verdana / sans-serif for text, black-on-white for the main area, and #dad9d4 for gray shaded backgrounds.)*
- All content should have reasonable sizing, padding, and margins such that content does not awkwardly bump into other content on the page. Content should also be aligned in reasonable ways for easy viewing. *(We center our various elements; our forms are 24em in width; many elements have 1em of padding or margin for separation.)*
- Any query results should be shown in **tables** with captions describing the tables, and headings describing each column. The rows of the table should alternate in background color, also called "zebra striping." Borders should be collapsed. *(Our page has every other row use a background of #dad9d4, starting with the 2nd row.);'/*

Development Strategy:

While testing, I suggest you use the imdb_small database and not the full imdb. When you think your code works, switch your PHP code to refer to imdb.

Use the **Terminal** to develop your queries on the command line before writing PHP SQL code.



```
[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
```

Enable exceptions on your PDO object to spot mistakes, as shown in the textbook and in class. Print your SQL queries while debugging to see the actual query being made. Many PHP SQL bugs come from improper SQL query syntax, such as missing quotes, improperly inserted variables, etc.

You do not need to secure your page against HTML/SQL injection attacks to get full credit, but you may if you like.

Implementation and Grading:

You do not need to submit anything to a Moodle dropbox. Instead, leave your files in a project4 folder in your web space on the cs325 server so that I can see your pages in action.

Your HTML (including PHP output) should pass the W3C HTML validator.

Your CSS code should pass the W3C CSS validator and should avoid redundant or poorly written rules.

Your code should follow **style guidelines**. Minimize global variables, and use descriptive names.

Place descriptive **comments** at the top of each file, each function, and on complex code. Also place a comment next to every single SQL query you perform that explains what that query is searching for.

Use parameters and return values properly. Show proper separation of content, presentation, and behavior between HTML, CSS, and PHP.

With so much in common between the two search pages, it is important to find ways to avoid redundancy.

You should use the PHP `include` function with shared common content included by various pages. Also use functions as appropriate to capture structure and repeated code and HTML content.

For full credit, do not directly write any actor's ID number anywhere in your PHP code, not even Kevin Bacon's. For example, don't write a line like:

```
  $bacon_id = 22591;   # Kevin Bacon's actor id  (BAD, don't do this)
```

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

In general you should limit yourself to the SQL query syntax discussed in class. In particular, you should not use advanced material such as SQL sub-queries in this project.

Format your code: Properly use whitespace and indentation. Use good variable and method names.