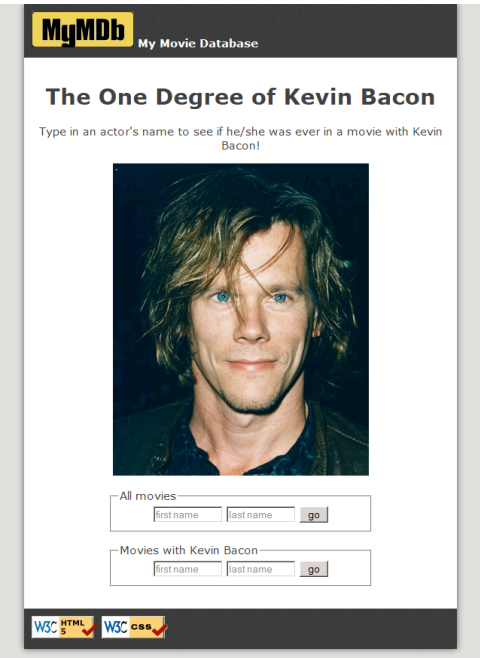
CSCI 4300: Web Programming

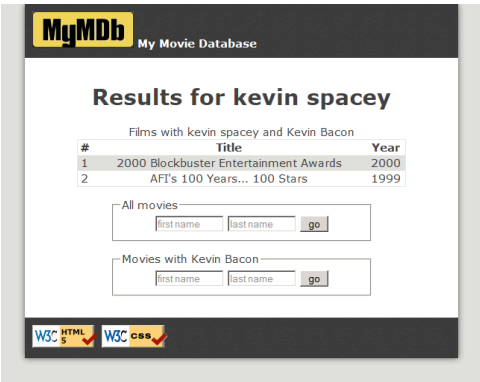
Spring 2016

Project 6: Database Management

Due: April 12 (11:59 pm)

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





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 your database and has a large number of connections to other actors.)

The following **provided file** is at ELC to get started:

- index.php, the initial front page that welcomes the user to the site

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 and turn in (besides index.php)**:

- 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 at ELC; you may modify it in any way you like, or you may leave it as-is. If you modify it, turn in your modified version. 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 API.

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 needs to have 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
roles	actor_id, movie_id, role

Your search pages need to 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 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.

Hint: To find the proper query, you will need to join all three of actors, movies, and roles. Retain only the rows where the relevant IDs from the tables match each other, and also retain only the rows that pertain to your particular actor. Our solution joins 3 tables in the FROM clause and has one test in its WHERE clause.

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. If the actor doesn't exist in the database, don't show a table, and instead show a message such as, "Actor Borat Sagdiyev 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."

This query is bigger and tougher, because you must locate a pair of performances, one by the submitted actor and one by Kevin Bacon, that both occurred in the same film.

Hint: You must join a pair of actors (yours and Bacon), a pair of roles that match those actors, and a movie that matches those two roles.

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 returns 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, you use the actor who has appeared in the most movies, breaking ties by choosing the actor with the lower-numbered ID.

You could figure out how many movies an actor has appeared in using a series of joins between tables, but this can be hard to get right and can be slow. To help, we have created a column in the actors table named film_count that contains the total number of roles played in all films by the actor. You can use this column to help write your query.

Hint: You don't need any JOINs here because all information comes from the actors table.

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.

- W3C validator button links (in the snapshots) are NOT required.
- 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 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:

We will provide some IMDB data in ELC.

Use the **MySQL console** to develop your queries before writing PHP SQL code. 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:

Avoid 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 = 123;   # Kevin Bacon's actor id  (BAD, don't do this)
```

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 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 (query #1 or #2 described previously, preceded by a call to query #3 to get the actor's ID first for use in query #1 or #2).

Properly use whitespace and indentation. Use good variable and method names. Avoid lines of code more than 100 characters wide.

How to Submit:

Submit your ".zip" file using ELC. Only team leaders need to make a submission. **Every student needs to submit a peer-evaluation form within 24 hours of the project submission deadline**.

Do not place your solution on a public web site. Submit your own work and follow the course misconduct policy.