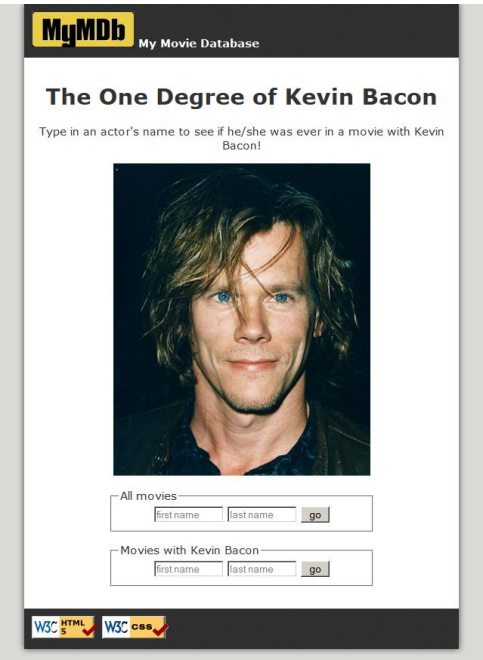
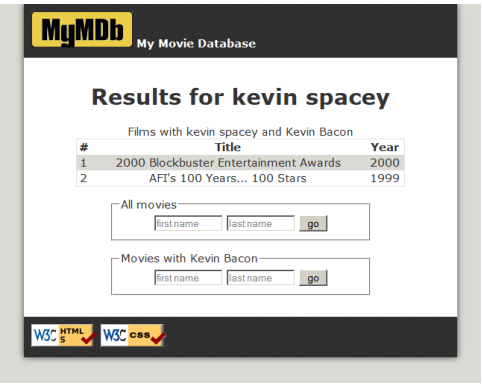
University of Washington, CSE 154
Homework Assignment 8: Kevin Bacon

This assignment focuses on querying a relational database using SQL in PHP.





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 **provided file** is on the course web site to get started:

- **mymdb.php**: the initial front page that welcomes the user to the site

The front search page **mymdb.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**:

- **mymdb.php**: the initial front page that welcomes the user to the site *(modified by you)*
- **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

Additional optional extra features are found in a separate spec on the class web site. You do not need to complete any such features but may do so for extra late days.

Front Page, mymdb.php:

The initial page, **mymdb.php**, allows the user to search for actors. This file is already provided to you on the course web site; 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 on Webster to show a given actor's movies. Query the database using PHP's PDO library as taught in class. Connect to the database using your UW NetID as your user name, and the MySQL password that was emailed to you. *(If you don't have your password or lost it, email the head TA.)*

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." You can do this using if/else statements or with the CSS `nth-child` pseudo-selector (see textbook Chapter 3).

#	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 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`
`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. Note that the behavior of the page is unspecified if the actor being searched for is Kevin Bacon himself.

1. 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" and "Smith" and expect the program to do the right thing. But if your code 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, 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.

For example, if you have written this query correctly and searched the `imdb_small` data set, for "Chris Miller" you would produce the actor ID 321300. On the larger `imdb` database, for "Will Smith" you would produce 1718223. For "David Cohen" you would produce 348849. For "Elizabeth Taylor" you would produce 3031853.

Hint: You don't need any `JOIN`s here because all information comes from the `actors` table. If you don't want to write this query right away, you could temporarily hard-code an actor's ID or just write a query to return the first actor with a given first/last name, which is what the correct query would do anyway when there are no conflicts.

2. 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 both of `movies`, and `roles`. Retain only the rows where the relevant IDs from the tables match each other, and also retain only rows that pertain to your actor.

3. 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. This is the harder query and should be done last. 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. Our query joins 3 tables in the `FROM` clause and contains 2 conditions in its `WHERE` clause.

Note: For all three queries, if any user input is inserted into the query as a string, it *must* be surrounded by quotes in the SQL query string. You can do this yourself or call the PDO object's `quote` method to wrap it in quote marks.

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. Please link to any images on your page using absolute paths, not relative paths.

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 a common amount of top/bottom content, including a 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. The only exception is when an actor isn't found. *(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. There must be some spacing around all content. 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:

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 code to refer to `imdb`.

Use the **SQL Query Tester** (https://webster.cs.washington.edu/cse154/query/) to develop your queries before writing PHP SQL code. Example test actor IDs are `376249` (Brad Pitt) or `770247` (Julia Roberts).

Enable exceptions on your PDO object to spot mistakes, as shown in the textbook and slides. 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. You are not required to test for the case where query parameters are empty, invalid, or not passed.

Implementation and Grading:

Your HTML (including PHP output) should pass the W3C HTML **validator**. Your HTML should choose appropriate tags for the semantics of the content. Pay particular attention to using proper tags around HTML tables, such as choosing between `td` (normal cells) and `th` (headings), and using the `caption` tag as appropriate. Your CSS code should pass the W3C CSS validator and should avoid redundant or poorly written rules. For example, if the page uses the same color or font family for multiple elements, group those elements into a single CSS rule, so that it would be possible to change the page's color/font by modifying a single place in the CSS file.

Your code should follow **style guidelines** similar to those on our past homework specs and the **style guide**. Avoid global variables. Use descriptive names. Use parameters and returns properly. Show proper separation of content, presentation, and behavior in HTML, CSS, and PHP. With so much in common between the two search pages, it is important to find ways to avoid redundancy. 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 the following:

```
$bacon_id = 22591;     # 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 (well, technically, a call to query #1 described previously to get the actor's ID, followed by a call to query #2 or #3). If your query might return multiple rows but you need only one row, make sure to include `LIMIT 1` in your query.

In general you should limit yourself to the SQL query syntax discussed in class and the textbook. In particular, you should not use advanced material such as SQL sub-queries, unions, or SQL procedures on this assignment. If you are not sure whether a given SQL command or syntax is allowed, please ask your TA or instructor.

Place descriptive **comments** at the top of each file, function, and on complex code. Place a comment next to every SQL query you perform that explains what that query is searching for. Make sure to follow the proper comment syntax for each language; in particular, don't put HTML comments in PHP files that do not produce HTML output.

Format your code similarly to the examples from class. Properly use whitespace and indentation. Use good variable and method names. Avoid lines of PHP code more than 100 characters wide.

Do not place a solution to this assignment on a public web site. Upload your files to the **Webster** server at:

- **https://webster.cs.washington.edu/students/*your_uwnetid*/hw8/**