

University of Washington, CSE 190 M

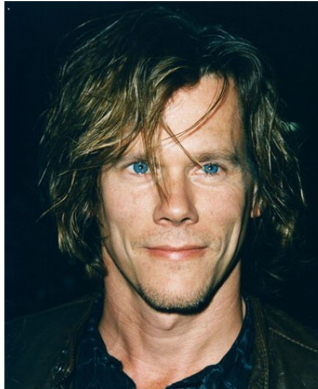
Homework Assignment 9: Kevin Bacon

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

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.) 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.

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, results are shown on [search.php](#). Turn in at least the following files:

- [mymdb.php](#), the front signup page
- [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")

You may have additional files beyond these if you like. But the above files are required at a minimum. If you have other files beyond these, submit those in a .ZIP archive [hw9.zip](#) on the same turnin page as the other files.

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.

To help with grading, in every HTML or PHP page that will be displayed to the user, you must include a script tag that links to the following instructor-provided JavaScript code. For now this file will be blank.

- <http://www.cs.washington.edu/education/courses/cse190m/11sp/homework/9/provided.js>

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

- 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** and info. In [mymdb.php](#) this area contains content of your choice, including at least one image of Kevin Bacon (or your central actor) and 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](#) to show movie results for that actor, but it is possible that the name the user types (such as "Will Smith") will match more than one actor. And some names match no one in the database.

To address this problem, you can instead contact an intermediate web service [actorid.php](#) provided by the instructor to find out which actor matches a given name. You can use an **Ajax request** to contact this web service, which accepts query parameters for the actor's first and/or last name. It returns output in XML format representing the actor(s) who match that name. The information flow is roughly the following:

1. [mymdb.php](#) User types a first/last name into the form and clicks the "go" button.
2. [bacon.js](#) JavaScript code creates an Ajax request to look up the actor ID for this actor.
3. [actorid.php](#) Looks up the actor(s) with that first/last name and outputs them as XML.
4. [bacon.js](#) Processes the XML that comes back, to discover the proper actor ID.
5. [bacon.js](#) Instructs [mymdb.php](#) to submit its form to [search.php](#) using the actor ID found.
6. [search.php](#) Shows the movie results for the given actor.

You don't need to use our web service if you'd rather write that functionality yourself. But either way, your page needs to find some way to map a name (or partial name) to a single unique actor.

Form submit issues: If your "go" button is a `submit` button inside a `form` tag, any click on it will by default submit the form. You may not want this to happen, if your form needs to do an intermediate Ajax lookup for an actor's ID before the form is to be submitted. If so, you should listen for the form's `submit` event and `stop` that event so that it doesn't happen when you don't want it to. Your JavaScript code can manually submit the form when you are ready by calling `.submit()`; on the DOM object representing the `form` tag.

The web service [actorid.php](#) is described in more detail on the next page.

Actor ID Web Service, [actorid.php](#) (provided by instructor):

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/cse190m/homework/hw9/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` database, pass an additional query parameter named `imdb` set to any value. If you want to return all the actors that matched a given name rather than just the first, pass a query parameter named `all` set to any value.

https://webster.cs.washington.edu/cse190m/homework/hw9/actorid.php?first_name=ad&last_name=pitt&imdb=true&all=true

```
<actors count="3">
  <actor id="376249" first_name="Brad" last_name="Pitt" appearances="59" />
  <actor id="755520" first_name="Adelaide" last_name="Pittaluga" appearances="1" />
  <actor id="782869" first_name="Nadia" last_name="Scarpitta" appearances="4" />
</actors>
```

Movie Search Page, [search.php](#):

The [search.php](#) page performs two queries on the `imdb` database on Webster 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.)

To help us with grading, please use the following **standard query parameter names** if you pass these kinds of values. Your [search.php](#) doesn't have to accept all of these parameters; but use these names if you do accept them.

- `actor_id` for an actor's ID
- `first_name` for an actor's first name
- `last_name` for an actor's last name
- `full_name` for an actor's full name

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 performs the following two queries. For each query, you must use a **join** between several database tables.

1. 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. You may assume that any actor in the `actors` table has been in at least one movie.

Hint: Join `actors`, `movies`, and `roles`; retain rows where IDs from the tables (actor ID, movie ID) match each other and the name/ID of your actor. Our solution joins 3 tables in the `FROM` clause and has one test in its `WHERE` clause.

2. 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 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. For example, don't write a line like:

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

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 5 tables in the `FROM` clause and contains 3 conditions in its `WHERE` clause.

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

No.	Title	Year
1.	Private War, A	2005
2.	Reefer Madness	2004
3.	Blind Horizon	2004
4.	Churchill: The Hollowood	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 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 two major operations described previously should be done with a single SQL query to the database.

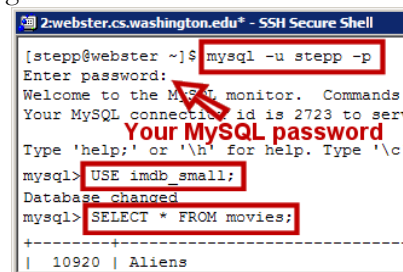
The behavior of the page is undefined if the actor being searched for is Kevin Bacon himself.

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 reserve the right to 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 test actor IDs 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 PHP `mysql_` functions to spot mistakes. The functions return `FALSE` if they fail. 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.



Implementation and Grading:

Submit your work from the course web site. For reference, our [search.php](#) contains roughly 95 lines (65 "substantive"); we have roughly 90 lines (60 substantive) of common shared code; our [mymdb.php](#) is very short, only around 15 lines; our [bacon.js](#) is roughly 50 lines (35 substantive); and [bacon.css](#) is 90 lines (57 substantive).

Your HTML (including PHP output) should pass the W3C XHTML validator. Your CSS code should pass the W3C CSS validator and should avoid redundant or poorly written rules. Your JavaScript code should pass the **JSLint** tool.

Your code should follow **style guidelines** similar to those on our past homework specs. Minimize redundant HTML/PHP code. Use functions and include files to break up code. Avoid global variables. 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.

For full credit, you must write your JS code using **unobtrusive JavaScript**, so that no JavaScript code, `onclick` handlers, etc. are embedded into the XHTML code. Properly use the XHTML DOM to manipulate page content.

Show proper separation of content, presentation, and behavior between HTML, CSS, and JS/PHP. As much as possible, your JS code should use styles and classes from CSS rather than manually setting style properties in the JS.

Format your code similarly to the examples from class. Properly use whitespace and indentation. Use good variable and method names. Avoid lines of 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/your_uwnetid/hw9/