Dr Richard Stibbard

The Ultimate Web Development Course

Learn ALL the major web development technologies in one working project

HTML, CSS, MySQL, PHP, jQuery and AJAX

www.webinaction.co.uk

The Ultimate Web Development Course

Chapter 1: Course Introduction

Welcome to this companion to my video course *The Ultimate Web Development Course*, available from http://www.webinaction.co.uk. Whilst it is possible to use this book as a stand-alone course, it is primarily intended to be used in conjunction with the video course as a reference and as an easy way to search the course for particular keywords rather than to be read on its own.

You will be familiar with how many websites are becoming increasingly like desktop applications, reacting instantly to your choices without making you wait for the page to refresh or go to a new page to see the results. An example of an impressively smooth web interface of this sort is www.trivago.com, where new results for hotels appear as you select different destination cities, and the search results are progressively filtered according to the price range you select, distance from a location, facilities and any facilities you require, all without leaving the main page.

I will base my project loosely on Youtube's 'Add to favourites' feature where you click on 'Add to' and then Favourites and the Movie title is added instantly to your favourites list on the left. (Here and throughout the book, refer to the online demonstration at www.webinaction.co.uk/favouritemovies/).

On the right of the Youtube page is a list of movies in the database, waiting to attract the user's attention. On the left are those movies which the user has already added to their favourites list. To save any disappointment later, I'll tell you now that these are not real movies – just images – and the database contains just a number of spoof movie titles and user names – but this dummy data is enough to set up a functioning demonstration website. Real data can be substituted later.

Moving the mouse over a movie reveals a heart icon which you can click on, and as on Youtube, the movie name will appear instantly in the list on the left. The style is altered to draw the user's attention to part of the screen where the change has taken place – otherwise it may not always be obvious that anything has changed. If you refresh the page, the movie will remain in the favourites list because we've stored it in the database the background, using AJAX.

The project demonstrates several ways of firing database updates, so instead of clicking, to delete movies from the favourites list, I've implemented a drag-and-drop interface – drag a title from favourites to the trash can and you'll see that the movie disappears from the list on the left and reappears on the right, again with visual feedback, in the form of a border.

Clicking on a movie thumbnail or title brings up the single movie page. If the movie's already a favourite, you can click to remove it; if not, click to add it. I'll teach you will learn how to dynamically change the text in this way according to the state of the data. If we continue deleting favourites until there are none left, the favourites list title changes, the trash can disappears, and we get an appropriate welcome message. Do the reverse and add all the titles to the favourites list and a different welcome message appears.

Without going into too much depth here, we'll also build an admin facility to add, delete or alter the names of movies and movie-goers in the database. Again, this mimics a desktop application, with the records instantly updated as you enter or alter them.

So, are you fascinated to know how this was done? Would you like to know how to tie together the various different technologies involved to make a website like this?

My experience is that there are many tutorials on the web which teach various aspects of what we are going to do here, but to get them to work may need a lot of digging around, learning some bits from one place and some from another, and you may or may not waste a lot of time in the process and not end up with anything that actually functions.

I have drawn everything you need into a coherent and structured course.

I will teach you to code this project from scratch if necessary, but the course is organised in a very flexible way to suit learners of all levels. It's suitable for absolute beginners, taking you right from basic HTML page structure through all the stages to the final product.

But if you're like me you won't have much patience sitting through long explanations of things you already know, and I realise that your previous experiences may be highly varied. For this reason, I have organised the course so that if you have existing knowledge you can join at any point and get straight to the parts you need, skipping the parts you already know, and will have no difficulty in finishing the project.

Completed working files are provided for every stage, matching the lessons exactly, so that if you don't need to be taken through a particular part, you can download the relevant files and concentrate on the exact topics you need, wasting no time.

My aim in my coding style is to be accommodating to you as a learner, so that you understand every line you write, because there's no point in typing in clever code if you have no idea of how it works or what you are doing.

I will explain all these stages step by step so that you truly understand what is going on and will be able to build on and adapt the foundations laid here to build your own web applications.

Dynamic, instantly refreshing websites powered by jQuery and AJAX are a key aspect of modern web design which every developer and designer needs to understand.

I do hope you will enjoy the course and find it very useful for your professional development or personal interest.

Richard Stibbard

Email: learn@webinaction.co.uk

(NB A few of the lessons are missing from this book as they are self explanatory)

Chapter 2: Komodo Text Editor

Lesson 4: Introducing Komodo Edit

In order to get started with our project, we have to begin by downloading and configuring a code editor. There are quite a number of code editors available and you may already have a favourite one, perhaps Notepad++ or TextPad, in which case it's fine to continue to use that, but I'm going to use Komodo Edit, which is a very popular choice, and I'll take you through the process of downloading and installing it.

The most important thing about all code editors is that they save your scripts in plain text only. Writing code is not like creating office documents – they're not meant to be printed out, but to communicate commands to the computer, so we must use an editor which saves files as plain text. A word processor is not suitable because the files it produces are full of formatting commands.

A dedicated code editor like Komodo Edit has a host of useful features for the programmer:

- it can highlight the syntax of your code, making it much easier to read and helping you to spot errors.
- the colours and formatting used for the syntax highlighting can be customized to suit your preferences.
- it can automatically complete programming code and close tags according to the language you are coding in, minimizing the amount of typing that you have to do and also minimizing the possibility of errors due to typing mistakes, forgetting to close tags, brackets, and the like.
- it can collapse and expand sections of code, making it easier to find relevant sections of your programs as they get longer.

And best of all, it is completely free.

We're going to begin by downloading the installation program from its website, activestate.com/komodo-edit. We'll install the software, leaving the all the options unchanged as we do.

Finally if you want to, you can change customize the syntax highlighting to a scheme of my own which I like because I think it is clearer than the default schemes.

Lesson 5: Download Komodo Edit

First of all, start up Firefox and navigate to the Komodo Edit homepage at activestate.com.

Under Developer Tools, choose Komodo and then Komodo Edit and then click on the Download button.

Choose the appropriate download for your operating system. I'm going to choose the Windows version.

According to the browser you're using, you may get the options to either run this file from the server or save it to your hard disk. In Firefox, which I'm using, you're only given the option to save. You can save it wherever you like, on your desktop or in your downloads folder; it doesn't matter as long as you know where it is.

The installation program will download, and in the downloads box in Firefox I can right-click on it and select Open Containing Folder to go straight to the folder to which the file is being downloaded. This is a large download, more than 50 MB, so I'll pause the video until it completes.

When the installation is complete, click on the installation file and choose 'Run'.

Click 'Next', agree to the license conditions, and click 'Next' again.

Here we have a list of optional components to be installed. Leave them as they are and click 'Next' and then 'Install'.

The installation process also takes a fairly long time so I will again pause the video until it completes. At the end leave the two checkboxes checked, to launch the program and load up the User Guide.

That's completed the installation of Komodo Edit. You can keep or delete the installation file as you wish.

In the next video we'll explore some of the features of the program and I'll show you how you can customize the appearance of its code highlighting function.

Lesson 6: Configure Komodo Edit

Komodo Edit will start up with two tabs loaded, the User Guide and the start page. We're not very interested in the start page so let's close it straight away. We can prevent the start page opening up every time we start Komodo Edit by going to Edit > Preferences and then choosing Appearance at the top on the left and uncheck the box where it says 'Show Start Page on Startup'.

Now we can use the User Guide to demonstrate Komodo Edit's preview pane. With the User Guide still loaded up, choose View > Browser Preview or click on the preview icon on the toolbar and choose to preview the page in a Komodo tab, and you'll see the formatted page just as it would look in a browser. This is a feature we can make use of if we want when we come to writing and styling our HTML, instead of switching to a browser. According to the page layout, we may want to have this pane situated at the bottom of the window as it is now or tiled to the right with the window split. We can change the layout by going to View > Rotate Split View.

Now you can optionally customize the display by using my custom syntax highlighting scheme.

Close down Komodo Edit.

Go to the Working Files, Chapter 1 and copy the file 'Custom Scheme.ksf' to your Application Data folder for Komodo Edit. To do this you will need to be logged into Windows as an administrator and you will need to be able to see hidden folders.

In Windows Explorer, go to Tools > Folder Options Click on the 'View' tab and check 'Show hidden files, folders and drives'.

Then in Windows Vista and Windows 7, go to to C:\Users\{your user name}\AppData\Local\ ActiveState\KomodoEdit\7.1\schemes\ and paste the custom scheme file in there. In Windows XP the Application Data folder is C:\Documents and Settings\{your user name}\Application Data and then find the 'activestate' folder and the schemes folder inside that.

Restart Komodo Edit and select Edit > Preferences > Fonts and Colours and under Scheme you will now find Custom Scheme. Select this and click OK. Type something in and you will find that the text is much larger – this is for the purposes of clarity in these videos – and as we go on you will find that the colours of the various parts of the file are more vividly distinguished. You can alter any aspects of this custom scheme by going back to the Fonts and Colours dialog. For instance if you don't want the text so big, you can reduce it here. All the colours are customizable. Or you can revert to the default scheme or choose any of the others in the list, as you prefer.

Komodo Edit is now installed and set up as we want it.

Chapter 3: Local Webserver Installation

Lesson 7: Introducing XAMPP

So far we've worked only with static HTML. This can only display the text written in it – it cannot react at all to the user's input, for instance by displaying the results of a particular search or settings specific to a user's account.

To do this we need to set up your computer to act as a web server. Whilst we could use a remote web host to host our databases and do all our work online, what we are doing is purely experimental and of no interest to anyone else. We wouldn't want it to be publicly accessible so it is much better to do all the work for this project privately on your own computer.

As it stands, your computer cannot do this. You have to download and configure a bundle of software called XAMPP.

XAMPP is an acronym. The X stands for Cross, as in cross-platform, meaning that it runs on a variety of operating systems: Windows, Linux, Mac OS and Solaris.

The A stands for Apache, which is the webserver underpinning the installation. Apache is the world's dominant webserver, powering over 100 million websites globally, more than any other web server.

The M stands for MySQL. MySQL refers to both the database or databases which we are going to be setting up and using to store the data needed to power our website, and to the language we use to communicate with the database, to create databases, to modify their structure, and to add, modify and delete data.

The first of the two Ps stands for PHP. In MySQL alone, it is not possible to run commands automatically. Every time we want to run a command, we have to type it in anew. PHP automates the process of running MySQL commands, allows us to control our database through a web browser, and integrates the database with our website.

The second P stands for PERL, which we are not going to be using at all in this tutorial. Perl is a general-purpose scripting language which is less relevant to our needs than PHP, which is tailored specifically for building web applications and dynamic database-driven websites.

We're going to download XAMPP from its website, apachefriends.org; we are going to set up your computer to act as a server, and finally we will access and test this server installation in your browser.

Navigate to apachefriends.org in your browser, click on the XAMPP icon at the top. Scroll down. Here are the four distributions. You need to choose the one appropriate to the operating system you are working on. I am going to choose the Windows distribution.

Again, we need to scroll down. Choose the installer version.

We are redirected to the sourceforge website. The download should start in a few seconds or if not there is a direct link you can click on. Choose 'Save file', click 'OK', and wait for the large download to complete. Then click on the XAMPP installer file you have downloaded and follow the prompts to install XAMPP.

Lesson 9: Change server time-zone

The time-zone for the Apache server is set by default to Berlin time; in this lesson we'll edit two configuration files to set it to your local time.

Start up the XAMPP control panel and next to Apache, click 'Stop'.

Wait for a moment until Apache has shut down and 'Stop' changes to 'Start'.

Then click on 'Config', and select 'php.ini' from the drop-down list.

Search for 'timezone', all one word, and you will find the setting:

```
date.timezone = Europe/Berlin
```

Change this to your local time-zone using the list of supported time-zones at php.net/manual/en/timezones.php.

Save and close 'php.ini'.

Then in the XAMPP Control Panel, next to MySLQ, click on 'Stop' and wait for MySQL to shut down.

Choose 'Config' and select 'my.ini'.

At the end of my.ini, add the lines:

```
[timezone]
default-time-zone = "{your timezone}"
```

(changing `{your timezone}` to your actual time-zone, of course!). Note that here 'time-zone', has a hyphen in the middle.

Save and close 'my.ini'.

Start up Apache and MySQL again by clicking 'Start' for each. Now your time-zone is set correctly.

This stage is not essential for the database project - you can skip it if you want, but a small exercise later uses time and if you do not go through this step, remember that the time zone may be wrong for that exercise.

Lesson 10: XAMPP tools and phpMyAdmin

Now we'll have a brief look at the important components of the XAMPP front page.

Click first of all on ‘Status’ in the menu on the left-hand side. This will give us a list of components which are activated and deactivated. For our work, the top two, MySQL database and PHP, must both be activated, so check that this is the case.

Under PHP, click on ‘phpinfo’. This gives us a lot of information about our installation – it goes on for miles. All that's important for the moment is just to notice in passing the version of PHP you are running – it's a good thing to know that. Mine is version 5.4.4.

The only other thing that matters to us is further down the menu on the left under Tools; click on phpMyAdmin. This is by far the most important component of these XAMPP tools.

PhpMyAdmin is the control panel that we use to work directly with our MySQL databases. It allows us to perform all possible commands on our databases, making new ones, adding tables, and adding, modifying and deleting data.

PhpMyAdmin allows you to become familiar with MySQL commands and to test them out directly, without the added complication of running them from within PHP scripts, and in the next chapter we will learn most, if not all, the MySQL commands you'll need to make a database-driven website, only some of which we will be applying to our project.

Chapter 4: Bare Bones HTML5

Lesson 11: Introducing HTML5

First we need to begin by coding the basic document structure for our new website, using HTML, hypertext mark-up language.

HTML is used to mark up the basic structure of webpages, to tell the browser which part of the page is which, and consists for the most part of a series of opening and closing matching tags or elements inside angle brackets, enclosing portions of the document.

Every HTML document must begin with a DOCTYPE declaration, to tell the browser what version of HTML is being used. The doctype declaration for HTML5 is very simple:

```
<!DOCTYPE html>
```

Following the doctype declaration, every HTML page is enclosed in opening and closing `<html></html>` tags, and these enclose a series of other tags, nested inside them. So we have opening and closing `<head></head>` tags, and inside these, `<title></title>` tags, these enclosing the title of the web page, like this:

```
<!DOCTYPE html>

<html>
    <head>
       <title>Instant Update Database Project by WebinAction.co.uk</title>
    </head>
</html>
```

Each pair of tags or elements tell the browser what type of information is contained in each section. This makes it easier for us to give functional descriptions to the various parts of our web pages, enabling browsers to render the information meaningfully, and is particularly important for accessibility as it can be used to enable screen-reading software to find the various sections, so that they don't, for instance, read out long menus every time a page loads but go straight to the important part of the page.

So HTML5 elements have names like `<header>`, where we can put the logo and banner for the website, `<nav>` for the navigation area, `<section>`, where we can put the main content, and `<footer>`, for the copyright statement and things like that.

I will draw your attention now to the importance of care in your HTML mark-up. This largely means making sure that your tags are nested properly so that pairs of outer elements fully enclose the elements inside them. This is an example of valid mark-up, because we have a pair of `<section>` tags fully enclosing a pair of paragraph tags, which in turn enclose the text of the paragraph:

```
<section>
    <p>Paragraph text</p>
</section>
```

The following example is not valid, because we have the closing paragraph tag overlaps outside the `<section>` tag, which the browser will not be able to make sense of:

```
<section>
    <p>Paragraph text
</section></p>
```

Take great care to avoid overlapping tags like this.

One final point is that if you have used previous versions of HTML you may be used to closing those relatively few tags which do NOT occur in pairs, such as the `<br>` tag for a line break, with a space and a forward slash or just a forward slash: `<br />` or `<br/>`. In HTML5 these tags are self-closing and do not need a forward slash, so use just `<br>`. Another example is the `<img>` tag used to link to an image – just use the tag without a forward slash at the end. Whilst putting in the forward slash will do no harm, it is unnecessary and can be dispensed with.

You can find many guides to HTML5 on the web – there is a discussion of semantic mark-up at http://dev.opera.com/articles/view/new-structural-elements-in-html5/ and a list of elements and descriptions of their functions at https://developer.mozilla.org/en-US/docs/HTML/HTML5/HTML5_element_list.

Lesson 12: Bare bones HTML – movies list page

We need to make just three simple HTML files – the first we'll deal with will be the movie list page. This will form the structure which we'll use to display the list of available movies, each with a thumbnail image, title and description. Clicking on a title or thumbnail will take us to the single movie page, which has a very similar structure, but displays only one movie, with a large movie image.

We also need a generic admin page. This will do for the structure for both our users-admin page and the movie admin page. The heart of this is an HTML table, with three columns in each case, one each for the firstname and lastname of the users, and for the title and description of the movies, and a column to add and delete entries.

If you already know HTML5 and don't want to type this HTML code in, then you'll find the completed files in the Working Files in Chapter 04.

Once we get to using PHP we will be working in the web root directory for the Apache server we have just installed, so we might as well start out in the right place from the beginning. This is 'c:\xampp\htdocs', so open up Windows Explorer and navigate there, and then make a new directory in it called 'favouritemovies'.

Start up Komodo Edit and begin by typing in what every HTML document has to start with - a doctype declaration to tell the browser which version of HTML is being used. We will be using HTML 5 for the project, and the doctype for this is simply `<!DOCTYPE html>`.

Save the document in the 'favouritemovies' directory you just made as 'index.html'.

Then every html document needs to be fully enclosed in `<html></html>` tags. In HTML5 it's recommended that all tags should be lower case; they will work whatever case they are, but this is best for compatibility, so I'll always use lower case. Komodo Edit autocompletes the tags and indents the document nicely for us.

Within the `<html></html>` tags we will be adding a series of other tags, to identify the various elements of the webpage to the browser. Most of these are in matching opening and closing pairs inside angle brackets, like the `<html></html>` tags.

Inside the <html> tags, every web page needs to have a <head> section and below that a <body> section:

```
<!DOCTYPE html>
<html>
    <head>
    </head>

    <body>
    </body>
</html>
```

The `<head></head>` tags enclose the parts of the document which do NOT appear in the document window.

We need to tell the browser what character encoding to use, with a META tag. We are using UTF-8, which can display a huge range of characters for international writing systems.

Type in `<meta charset = "UTF-8">` between the `<head></head>` tags:

```
<!DOCTYPE html>
<html>
    <head>
       <meta charset = "UTF-8">
    </head>

    <body>
    </body>
</html>
```

Also inside the `<head>` section we put the `<title>` tags, enclosing the title of the website, “Instant update database project by WebinAction.co.uk”:

```
...
    <head>
       <meta charset = "UTF-8">
       <title>Instant update database project by WebinAction.co.uk</title>
    </head>
...
```

This will appear in the title bar at the very top of the browser, not inside the browser window itself.

Then we get to the part of the file which corresponds to what we see in the browser window – this all comes in between the opening and closing `<body></body>` tags.

In HTML5, `<header></header>` tags enclose the banner part of the page:

```
<body>
   <header>
   </header>
</body>
```

These are not to be confused with the <head> tags we just used – they are quite different. The `<head>` tags are for things which do not appear in the browser window, while the `<header>` is for the main title at the top of the page so we’ll use these as the first part of the document.

Then inside the <header> tags, put a pair of <h1> tags, for heading level 1. HTML headings are available in six levels, with the tags <h1> to <h6>. This is our main title so we’ll use the top level heading, <h1>, enclosing the title “Instant Update Database Project”:

```
...
    <header>
       <h1>Instant Update Database Project</h1>
    </header>
...
```

After that we want a list. There are two types of list in HTML, numbered lists, for which the tag is <ol>, and lists without numbers, for which the tag is <ul>. We don’t want numbers, so we’ll type in a pair of `<ul></ul>` tags.

Then we’re going to mark up our two navigation areas – on the top right we’ll be able to navigate from one movie-goer to another and also to the admin pages, and on the left we’ll be able to click on movie title to see the details for that movie.

HTML5 has special tags to identify navigation areas, `<nav>`, so we’ll use these to enclose both of these areas. After the closing `</header>` tag, type in opening and closing `<nav></nav>` tags, and inside them divide the area up into two parts, an area where our navigation header will go and the drop-down list in `<ul>` tags of users below it:

```
...
   <nav>
     <div></div>
     <ul></ul>
   </nav>
...
```

I have used `<div>` tags for the first section. I use `<div>` tags where there's no special meaning to a particular section and HTML5 semantic tags wherever there is some identifiable meaning, so as to avoid having masses of divs in the mark-up. Inside the `<div></div>` tags, put `<h2>`, 2nd level heading, tags, and inside them "Logged-in user's name".

```
...
   <nav>
     <div>
          <h2><a href="#">Logged-in user's name</a></h2>
     </div>
     <ul></ul>
   </nav>
...
```

This is going to link back to each user's main list page, so it needs an HTML link, an `<a>` tag, for "anchor", then `href =` to specify the page to be taken to. For the moment we'll put in a dummy link, using the hash sign # inside single quote marks: `<a href = '#'>`. This will mean we get the visual appearance of the link so we can sort out its style, without having to worry about exactly what the details of the link are.

Now we need to make a list, which will have in it all the users in the database except the currently selected one. For this we use the HTML tags `<ul>`, meaning 'unordered list'. Unordered lists have no numbering, 'ordered lists', for which the tag is `<ol>` have numbering.

Within the <ul> tags come the list items themselves, in `<li></li>` tags. For the moment just put "List of other users" there:

```
...
   <nav>
     <div>
          <h2><a href="#">Logged-in user's name</a></h2>
     </div>
     <ul>
          <li>List of other users</li>
     </ul>
     </nav>
...
```

Again, we need a link so we can style it when we come to that, so surround "List of other users" in `<a href = '#'>` and closing `</a>` tags:

```
...
     <ul>
          <li><a href="#">List of other users</a></li>
     </ul>
...
```

So that it looks like a list, copy and paste the list item in a couple of times so we have three of them.

Then we'll have another unordered list in `<ul>` tags, this time with just two list items, 'Manage users' and 'Manage movies', each again in a dummy link:

```
...
     <ul>
          <li><a href="#">List of other users</a></li>
          <li><a href="#">List of other users</a></li>
          <li><a href="#">List of other users</a></li>
     </ul>
     <ul>
          <li><a href="#">Manage users</a></li>
          <li><a href="#">Manage movies</a></li>
     </ul>
...
```

That's the end of that `<nav>` section.

Now we want a second `<nav>` section, the favourites list on the left, because this provides the navigation to go to a single movie and see its details.

So we need another set of `<nav></nav>` tags, inside which we'll put an `<h2>` heading, with the title 'Favourites' in it, and follow that with another unordered list, containing list items with the dummy title 'Movie Title', again surrounded by a dummy link:

```
...
  <nav>
    <h2>Favourites</h2>
    <ul>
       <li><a href="#">Movie Title</a></li>
       <li><a href="#">Movie Title</a></li>
       <li><a href="#">Movie Title</a></li>
    </ul>
  </nav>
...
```

Following that closing </nav> tag, we'll do the main section of the page, which we'll enclose in `<section>` tags:

```
...
    <section>
    </section>
...
```

Inside that, in h2 tags, we'll put in a greeting "Hi, (username will appear here)". Later PHP will put in the name of the current movie-goer from the database:

```
...
    <section>
       <h2>Hi, (username will appear here)</h2>
    </section>
...
```

Under that in paragraph tags, `<p></p>`, we'll put in generic instructions, "Here are some movies you might like. Click on the heart icon to add them to your favourites list." As you've already seen, this will later change according to whether or not the user has any favourites already or not:

```
...
    <section>
        <h2>Hi, (username will appear here)</h2>
        <p>Here are some movies you might like.
        Click on the heart icon to add them to your favourites list.</p>
    </section>
...
```

Now for the most complicated part, the list of movies, because each list item is going to contain a thumbnail image, the title, the description, and eventually the make favourite icon.

We'll start with the <ul> tags, for another unordered list:

```
<section>
    <h2>Hi, (username will appear here)</h2>
       <p>Here are some movies you might like.
       Click on the heart icon to add them to your favourites list.</p>
    <ul></ul>
</section>
```

And a list element in <li> tags again:

```
   <ul>
      <li>
      </li>
   </ul>
```

Then to enclose the image we'll use the HTML5 element `<figure>` and inside that for the moment we'll just put the text "Thumbnail image". We'll get the image later:

```
   <ul>
      <li>
         <figure>Thumbnail image</figure>
      </li>
   </ul>
```

As this is going to be a link, again enclose it in a dummy `a href = '#'` tag:

```
   <ul>
      <li>
         <figure><a href="#">Thumbnail image</a></figure>
      </li>
   </ul>
```

To keep the images and their associated text together on the screen and in the mark-up, we use `<figcaption></figcaption>` tags, inside the `<figure></figure>` tags:

```
   <ul>
      <li>
         <figure><a href="#">Thumbnail image</a>
            <figcaption>
            </figcaption>
         </figure>
      </li>
   </ul>
```

This is where we'll put the movie title, for which we'll use `<h3>` tags, and again a dummy link:

```
   <ul>
      <li>
         <figure><a href="#">Thumbnail image</a>
            <figcaption>
               <h3><a href="#">Movie Title</a></h3>
            </figcaption>
         </figure>
      </li>
   </ul>
```

After that we'll have the movie description, in a `<div>`, still inside the `<figcaption>` tags:

```
   <ul>
      <li>
         <figure><a href="#">Thumbnail image</a>
            <figcaption>
               <h3><a href="#">Movie Title</a></h3>
                  <div>Movie description</div>
            </figcaption>
         </figure>
      </li>
   </ul>
```

Then make sure it is all closed off properly and nested correctly, and that's a movie list item done.

Copy the whole list item and paste it in again twice so there are three of them, so it looks like a list and not a single item.

Finally, finish off by making the footer, inside `<footer></footer>` tags, enclosing paragraph tags, enclosing "Instant update database project by learn@webinaction.co.uk so that you have my email address in case anything should go wrong!

```
...
   <footer>
      <p>Instant update database project by learn@webinaction.co.uk</p>
   </footer>
...
```

To make an email link for this use `<a href = "mailto:learn@webinaction.co.uk"></a>`:

```
   <footer>
      <p><a href = "mailto:learn@webinaction.co.uk">Instant update database
         project by learn@webinaction.co.uk</a></p>
   </footer>
```

The closing </body> and </html> tags finish the document off:

```
...
      </footer>
   </body>
</html>
```

Check in your browser and you should have an ugly page, which nevertheless contains all the elements we need for the movies list page.

In the next lesson, we follow exactly the same procedure to make the single movie page, so I won't go through this step-by-step. The HTML, which is simpler than for the movie list page, is here for your perusal:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Instant Update Database Project by WebinAction.co.uk</title>
    </head>

    <body>
        <header>
            <h1>Instant Update Database Project</h1>
        </header>

        <nav>
            <div>
                <h2><a href="#">Logged-in user's name</a></h2>
            </div>
            <ul>
                <li><a href="#">List of other users</a></li>
                <li><a href="#">List of other users</a></li>
                <li><a href="#">List of other users</a></li>
            </ul>
            <ul>
                <li><a href="#">Manage users</a></li>
                <li><a href="#">Manage movies</a></li>
            </ul>
        </nav>

        <nav>
            <h2>Favourites</h2>
            <ul>
                <li><a href="#">Movie Title</a></li>
                <li><a href="#">Movie Title</a></li>
                <li><a href="#">Movie Title</a></li>
            </ul>
        </nav>

        <section>
            Large movie image
            <h3>Movie title</h3>
            <div>
                <p>Add to/delete from favourites</p>
            </div>
            <p>Description</p>
        </section>

        <footer>
            <p>Instant update database project by
            <a href="mailto:learn@webinaction.co.uk">
            learn@webinaction.co.uk</a>
            </p>
        </footer>
    </body>

</html>
```

Lesson 14: Bare bones HTML – Admin page

Again, we'll use an existing HTML file to make this new one. Open up movie-single.html and again cut out everything inside the `<section>` tags, leaving the section tags themselves in place:

```
...
   <section>
   </section>
...
```

Now we're going to make an HTML table. A load of nonsense has been written about how awful HTML tables are, to the extent that some people seem to think you should not use them at all. This is complete rubbish – we use tables for tabular data, which could be anything stored in a database – the thing about not using tables is that we no longer use them for layout. Once there was no choice, but now we have proper styling tools in CSS. But there is no need to avoid using tables for their intended purpose.

An HTML table begins and ends with `<table></table>` tags, and inside them each row of the table is made using <tr> tags, table row. We'll need three rows so put in three sets of `<tr></tr>` tags:

```
...
   <section>
      <table>
        <tr></tr>
        <tr></tr>
        <tr></tr>
      </table>
   </section>
...
```

Inside the first set of `<tr></tr>` tags, put `<th></th>` tags, for table header:

```
...
   <table>
      <tr>
         <th></th>
      </tr>
      <tr></tr>
      <tr></tr>
   </table>
...
```

We're going to have three columns, so put in three sets of `<th></th>` tags, and inside them the text for each column header, 'Firstname', 'Lastname', and 'Insert/Delete':

```
...
   <table>
      <tr>
         <th>Firstname</th>
         <th>Lastname</th>
         <th>Insert/Delete</th>
      </tr>
      <tr></tr>
      <tr></tr>
   </table>
...
```

Then put in three sets of `<td></td>` tags, table data, inside each of the remaining `<tr></tr>` tags:

```
...
   <table>
      <tr>
         <th>Firstname</th>
         <th>Lastname</th>
         <th>Insert/Delete</th>
      </tr>
    <tr>
       <td></td>
       <td></td>
       <td></td>
      </tr>
    <tr>
       <td></td>
       <td></td>
       <td></td>
    </tr>
   </table>
...
```

These are going to display data from the database, which we also want to be able to edit, so inside these `<td></td>` tags we'll have input boxes, which we can type into. So put `<input>` and then we need to specify the type of input – in this case it's a text input, so put `type = "text"`. Then give it a name, to identify the data it will contain, name='firstname'. So that we can see it, put in a dummy value, using "value=". You can put in any name you want. I'll put in "Albert":

```
...
   <table>
      <tr>
         <th>Firstname</th>
         <th>Lastname</th>
         <th>Insert/Delete</th>
      </tr>
    <tr>
       <td><input type="text" name="firstname" value="Albert"></td>
       <td></td>
       <td></td>
      </tr>
    <tr>
       <td></td>
       <td></td>
       <td></td>
    </tr>
   </table>
...
```

`<input>` is a self-closing tag, so we don't need a closing one.

Copy that and paste it in into the next <td> cell, changing 'firstname' to 'lastname' and changing the name to something else. Then make the last cell of the row, again <td> tags, with the text 'Insert/delete' inside:

```
...
   <tr>
      <td><input type="text" name="firstname" value="Albert"></td>
      <td><input type="text" name="lastname" value="Smith"></td>
      <td>Insert/delete</td>
   </tr>
...
```

This is where we will eventually be able to click to add and delete data.

Finally, copy the whole table row and its contents another couple of times so we get several rows of data, and put in some different names:

```
...
   <tr>
      <td><input type="text" name="firstname" value="John"></td>
      <td><input type="text" name="lastname" value="Tomkins"></td>
      <td>Insert/delete</td>
   </tr>
...
```

The closing `</table>` tag follows and ends the admin table.

The whole table should read:

```
<table>
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Insert/delete</th>
  </tr>
  <tr>
    <td><input type="text" name="firstname" value="Albert"></td>
    <td><input type="text" name="lastname" value="Smith"></td>
    <td>Insert/delete</td>
  </tr>
  <tr>
    <td><input type="text" name="firstname" value="John"></td>
    <td><input type="text" name="lastname" value="Tomkins"></td>
    <td>Insert/delete</td>
  </tr>
</table>
```

Save and preview that and we're ready to go on to the next chapter, on styling.

Chapter 5: Styling with CSS

Lesson 15: Introduction to styling with CSS

In this chapter we'll style our webpage. By the end of this chapter we'll have gone from bare bones HTML to this (see video). As ever, if you are familiar with CSS and don't need to go through this, you can download the finished files from Chapter 5 of the Working Files and go on to the next chapter.

My goal before working on the project is to introduce to you some of the basic principles of layout and styling. I'll take through the whole process of styling the movie list page, explaining what we're doing as we go along. As we go on, I'll gradually hand over to you, asking you to type progressively more of the style selectors yourself, so that your learning is active.

We'll start by stripping out all the default styling we currently have as this not useful and is potentially a nuisance. Then we'll create the basic layout, putting the various HTML elements in boxes positioned in different parts of the document window. Then we will add more detailed styling: fonts, colours and images. By the end you will have had a good overview of many of the style properties at our disposal. There are literally hundreds of CSS properties, so no tutorial can ever be exhaustive, unless it's exhausting as well, so I've given you some bookmarks so you can explore the world of CSS more on your own.

Our design will give us clearly delineated sections of the page, with a banner area, the various navigation panels, the main area for single movies or a list of movies and a footer.

It's possible to put styles inside our HTML document, in the head section and then inside opening and closing style tags, but we'll link to an external stylesheet. This has three advantages – it keeps our styling and content separate, which is something we always want to achieve. Lists of style declarations get very lengthy and if we have a great mass of style declarations in our main HTML file it makes it harder to read. Also, we can only use this if we are sure that the styles will only be used in that one file.

Finally, putting our styles in an external stylesheet will speed up page loading because after the first page has loaded the stylesheet will be cached on the user's hard disk and will not need to be loaded again.

If we want to style more than one page, as we do, we should always link to an external stylesheet. By doing this, any changes we make in the stylesheet will automatically take effect in all the pages which use it, no matter how many there are, so making changes to the styles is an efficient process.

In an ideal world, all browsers would display webpages identically. Unfortunately the world of HTML rendering is very far from ideal, and browsers do not all display webpages identically at all. For this reason, every time we write a web page and apply styling to it we must check it in a range of browsers as we go along, so that we identify any problems before they get out of hand.

I'll be checking this page in Firefox, Opera, Google Chrome, Safari and Internet Explorer versions 8, 9, and 10 and we will find that there is indeed a problem – Internet Explorer 8, which is the browser still used in Windows XP – cannot recognise HTML5 tags and completely garbles our layout. We will solve this problem by using a script called HTML-shiv.

Let's start by having a look at how CSS styles are written.

In our HTML, we have a series of tags or elements, like `<header>`, `<nav>`, `<section>`, and `<footer>`. To style these elements, we go to the CSS file and name a selector with exactly the same name as the HTML tag we are styling:

```
header
```

Any of the visible HTML elements we have used can be used as a style selector. The selector is followed by a pair of curly braces:

```
header {
}
```

Inside these curly braces we enter the properties for that selector, such as font-family, font size, or colour. Each property is followed by a colon, and then the value or values for that property, for instance, a font name for the font family, a certain font size, or a colour. Each selector may have a list of properties, and each one has to be terminated with a semi-colon. In this example, all text inside all paragraph tags would be given a style of Arial font, a size of 24pt, and the colour grey:

```
header {
    font-family; "Arial", sans-serif;
    font-size: 24pt;
    color: gray;
}
```

Armed with that knowledge, we can begin our work on styling.

Lesson 16: CSS reset

At the moment, although far from pretty, our putative webpages are far from style free – they have the default styles applied to the HTML tags by browsers – underlining and blue text for the links, various font sizes and bold text for the different heading levels, and bullets and indentation for the unordered lists.

This is a nuisance because none of these look anything like what we want, and they may even appear slightly different in different browsers, causing problems later, so we're going to start our styling by applying a CSS reset.

This will remove all these default styles and reduce everything to plain text, allowing us to start off with a clean slate.

You need to copy the css reset file from Chapter 5 of the Working Files to complete this lesson. We won't type all this in as it's not particularly interesting, but we'll open it up in Komodo Edit and have a look at it.

Go to 'Chapter 05 – Styling' and then to '05-02 CSS reset', where you will find the file 'reset.css'. Right-click on it and select 'copy' and then go to your 'favouritemovies' folder in htdocs, make a new folder called 'css', and copy this file in there.

Now open up 'index.html' in Komodo Edit and just below the `<meta>` tag enter a link to 'reset.css'. This begins with an opening angle bracket as usual, then `link`, then `href=`, which we have used before to link to another file:

```
...
<meta charset="UTF-8">
<link href=
...
```

As 'reset.css' is not in the same folder as 'index.html', but in the 'css' subfolder, we need to use a relative path to it, which is 'css/reset.css':

```
...
<meta charset="UTF-8">
<link href="css/reset.css"
...
```

We need to specify the relationship of the file to the present one, with `rel=` and it is a stylesheet, so we put `stylesheet` inside the quotes:

```
...
<meta charset="UTF-8">
<link href="css/reset.css" rel="stylesheet">
...
```

And close off the tag with `>`. As it is a self-closing tag, that is all that is required.

If you have worked in the field before, you may be used to specifying here the type of document using `type="text/css"`. This isn't necessary in HTML5, which cuts out some unnecessary verbiage, so we can save ourselves some typing and reduce clutter in the mark-up.

Save 'index.html' and refresh it in your browser and you will see that now it really is style-free, except for the underlining and blue for the links.

Copy the line `<link href="css/reset.css" rel="stylesheet">` and open up ‘movie-single.html’ and ‘admin.html’ and paste it in the same place, and they too will be rendered almost as plain text.

Open up ‘reset.css’ and you will see that it consists of a series of selectors corresponding to the HTML elements we have used, each one with properties like ‘0’, ‘normal’, and ‘none’. ‘Reset.css’ is overriding all default styles applied by browsers and replacing them with no styling at all.

We are now ready to move on to creating our own styles, with any potential problems removed at the start.

Lesson 17: Styling with Classes and IDs

A glance at our mark-up for 'index.html' will show that we are using the same HTML tags several times – the `<nav>` element is used twice, first for the user and admin navigation area and then for the favourites list, and within them we have `<ul>` and `<li>` tags, which for the moment have nothing to distinguish them.

We need to know how to distinguish the same tag name being used in different parts of the document. We do this in the HTML by assigning ID or class names to the elements using `id=` or `class=` in the HTML.

IDs are used to define an element as unique on the page, and the same ID name must not be used twice on the page. We could give a `section` element the ID name "main", and then we cannot use the ID name "main" again. To apply styling to an element identified with an ID, we prefix the selector name in the CSS with a hash symbol, `#`.

To apply the same styling to a set of elements which recur several times in the page we cannot use IDs as they can only occur once. Instead we use classes, with `class=` in the HTML. In the CSS to apply styling to one or more elements identified with a class, we prefix the selector name in the CSS with a dot. So in this example, we would in one go apply the same styling to all the images with the class name `.thumbnail`.

Here is a link to a useful article at CSS-tricks.com explaining the difference between IDs and classes: http://css-tricks.com/the-difference-between-id-and-class/.

IDs are very powerful when we apply styling to them, overriding styles applied to classes. This can cause unwanted effects, so basically I avoid using IDs for styling and use only classes. I reserve IDs for the job of identifying the function of particular individual elements, for instance by giving an ID number to refer to a particular user's ID number in the database. I use classes for visual styling.

We can also target our styles exactly by using selectors which combine the HTML element name and the class or ID name in one selector.

In our project we will have unordered lists distinguished by class name, one for selecting users and one to navigate to the admin pages. We target these in the CSS by concatenating the element name, ul, and the class selector with its dot, with no space between them. We can also style an item but only when it occurs inside another tag, using parent-child selectors.

In this case the outer, containing, element comes first, and the inner element, the child selector, follows, with a space between them.

These various types of selector allow for pinpoint accuracy in styling. You just have to take care and remember that a list of separate selectors being styled alike is separated by commas. A combination selector targeting an HTML element which has an ID or class name applied to it has the element name and class or id name concatenated with no space, and a parent child selector targeting one item inside another has a space between the parent selector and its child.

Now we can go on, adding classes to our HTML to distinguish the various elements, and at the same time styling the basic layout of the page.

Lesson 18: Applying classes to index.html

In this lesson, we're going to identify the various parts of our index.html page using classes and in the next lesson we'll write style selectors to make the basic box layout, placing the various parts of the page in the appropriate places in the browser window, taking our index page from its current state ... to this (see video).

Open up 'index.html' in Komodo Edit and copy the link to 'reset.css' and paste it in one line below, changing 'reset.css' to the stylesheet we are about to write, 'style.css':

```
<html>
    <head>
        <meta charset="UTF-8">
        <link href="css/reset.css" rel="stylesheet">
        <link href="css/style.css" rel="stylesheet">
        <title>Instant Update Database Project by WebinAction.co.uk</title>
...
```

It is very important that this link goes below the one to 'reset.css', as styles are applied in the order the browser reads the stylesheets, so that later stylesheet will override the earlier one. If we have 'reset.css' last, it will override all our styles.

Now, in the 'css' folder, make a new text file and call it 'style.css'. This is where we will write our styles.

Go also to the Working Files, 'Chapter 5, 05-03 Styling the Layout', where you will find two placeholder images, called 'movie.png' and 'thumbnail.png'. Select and copy these, go to your 'favouritemovies' folder and make a new folder called 'images-movies'. Make sure you spell it properly with a dash, not an underscore. Paste the two images in there.

Back in 'index.html', we need to give class names to some of our elements, especially where there is more than one occurrence of the same element name.

Find the first `<nav>` tag and give it the class name 'navigation' by typing in `<nav class="navigation">`. Then name the `<div>` inside it `class="select_users"`:

```
...
   <nav class="navigation">
      <div class="select_users">
         <h2><a href="#">Logged-in user's name</a></h2>
      </div>
...
```

The `<ul>` list containing the user names should be `<ul class="users_menu">`, and the one for the admin pages `<ul class="admin_menu ">`:

```
...
  <ul class="users_menu">
     <li><a href="#">List of other users</a></li>
  ...
  </ul>
  <ul class="admin_menu">
     <li><a href="#">Manage users</a></li>
     <li><a href="#">Manage movies</a></li>
  </ul>
...
```

Then the next `<nav>` element, which contains the list of favourite movies, is `<nav class = "favs_list">` and the `<ul>` inside that, where the list of favourites goes, is `<ul class="favs">`:

```
...
    <nav class="favs_list">
        <h2>Favourites</h2>
        <ul class="favs">
           <li><a href="#">Movie Title</a></li>
           ...
        </ul>
    </nav>
...
```

The section containing the list of movies is `<section class="movie_list">`, and the paragraph tag inside it where the welcome message goes is `<p class="welcome">`:

```
...
    <section class="movie_list">
       <h2>Hi, (username will appear here)</h2>
       <p class="welcome">Here are some movies you might like.
       Click on the heart icon to add them to your favourites list.</p>
...
```

Inside the `<figure>` tag, replace the words 'thumbnail image' with the relative path to the image:

```
...
<figure>
 <a href="#">
  <img class="thumbnail" alt="Thumbnail" src="images-movies/thumbnail.png">
 </a>
...
```

We know what 'class' means. 'Alt' is alternative text to be displayed if a user has images turned off in their browser or cannot see images and needs to have a description of them read out, and 'src' means 'source' and then we have the relative link to the two images we just copied to the 'images-movies' folder.

Give the `<div>` enclosing the movie description the class name 'description':

```
...
   <figcaption>
      <h3><a href="#">Movie Title</a></h3>
      <div class="description">Movie description</div>
   </figcaption>
...
```

That's our class names done and ready to be styled.

Lesson 19: Box layout

Now we have our various class names set up in the HTML, so we can distinguish the different navigation areas and other parts of the page, and we're ready to begin styling, first of all creating our overall layout, putting the various areas into boxes positioned where we want them in the browser window.

In Komodo Edit, open our empty file 'style.css', and we'll start typing style selectors, each one followed by curly braces. Have 'index.html' open in a browser at the same time so you can switch from one to the other as the styles take shape. We'll start at the top of the page and work to the bottom.

The header is an HTML element, so its selector has no prefix, it's just `header`:

```
header {
}
```

Give it a height of 50px and make it a visible area of its own by giving it a bottom border, with border-bottom: 1px solid grey. Don't forget the semi-colon at the end of each style declaration:

```
header {
  height: 50px;
  border-bottom: 1px solid grey;
}
```

Next the 'select_users' div, where our users and admin navigation will go. 'Select_users' is a class so the selector has to begin with a dot:

```
.select_users {
}
```

We'll force the position of this to the top right of the window using 'position absolute'. This takes its position out of the normal flow of the document and allows us to position it exactly where we want it. We'll position it 65px from the right of the window and 15px down from the top:

```
.select_users {
    position: absolute;
    right: 65px;
    top: 15px;
}
```

Using a comma-separated list as I explained earlier, we can style the common elements of the users and the admin menus together, using ul.users_menu, ul.admin_menu. I'll give these absolute positioning and make them butt up against the right-hand side of the window:

```
ul.users_menu, ul.admin_menu {
    position: absolute;
    right: 0;
}
```

We'll see how these styles go – it's all a bit experimental until we finally get it all together, so we may have to alter some of them as we go along.

The final website will have the users and admin lists positioned almost on top of each other, but with only one appearing at a time, using jQuery to control their appearance and disappearance. We can't do that yet so for the moment we'll just hide `ul.admin_menu` by giving it a `display` setting of `none`, which means it won't be visible:

```
ul.admin_menu {
    display: none;
}
```

Next, we want the favourites list, styled with `.favs_list`, to appear to the left of the main section of the page. We do this by applying `float: left` to it, meaning that it will float to the left and other elements will float to its right instead of appearing below it. In conjunction with this, we need to give it a width. I'll give it a width of 175px. And I'll also give it some space to the left with `margin-left: 45px`:

```
.favs_list {
    float: left;
    width: 175px;
    margin-left: 45px;
}
```

I'll temporarily put a border round this:

```
.favs_list {
    ...
    border: 1px solid blue;
}
```

This is a shortcut selector, short for:

```
border-width: 1px;
border-style: solid;
border-color blue;
```

This is a designer's trick so that the element is really visible and we can check its position more easily. We'll remove it later.

Now the favourite movies list itself, styled with `ul.favs`. I'll give this a width of 170px:

```
ul.favs {
    width: 170px;
}
```

Then we can style the main section, for the list of movies and the single movie, in one go, using a comma-separated list of selectors: `.movie_list, .movie_single`. These need to appear to the right of the favourites list. The favourites list is 170px wide plus 45px margin, making 215px. We'll give the 'movie_list' and 'movie_single' areas a left margin of 225px to push them 10px out beyond the favourites list. To copy the Youtube layout, this needs a width of 610px. And again temporarily I'll give it a red border, which we'll remove later:

```
.movie_list, .movie_single {
    margin-left: 225px;
    width: 610px;
    border: 1px solid red;
}
```

All the thumbnail images can be styled together. They just need their size specified, a width of 178px and a height of 100px, giving a 16:9 ratio:

```
.thumbnail {
    width: 178px;
    height: 100px;
}
```

Finally, the footer. If we don't clear the left float which we applied to the favourites area, this will continue forever, and all other elements will try to position themselves floated to the right of it. We don't want this, so we apply `clear:both` to the footer, which clears the floats applied earlier so it appears below the main movie area, not beside it. We'll give the footer a height of 60px, and another temporary and distinctive border, this time green, and a top margin of 30px to give a bit of space between the footer and the main page:

```
footer {
    clear: both;
    height: 60px;
    border: 1px solid green;
    margin-top: 30px;
}
```

Now the elements are positioned as we want them on the screen; next we'll style the text and colours.

Lesson 20: HTML-shiv for IE8

Next we have to use the patch I told you about to make HTML5 work in Internet Explorer 8. IE8 is out of date, but it's all that's available if your user is running Windows XP, and at the time of making these videos, about 10% of all users were still using IE, so we need to do this.

Just below the stylesheet links, type in:

```
<!--[if lt IE 9]>
```

An opening angle bracket followed by an exclamation mark and 2 hyphens, `<!--`, is the way we normally comment out lines in HTML, meaning that we can read them but they do nothing in the browser, but Internet Explorer from version 5 onwards recognizes conditional comments which begin in this way and runs any script in them. No other browsers recognize anything inside comment tags, and so completely ignore it. We can therefore use this to run scripts which are needed only in IE. So this is a condition, saying if the browser version is less than IE 9 then run the following script. Make sure it has no spaces in it.

We are going to run a script, 'html5shiv', from the Google code repository. To run it, type in:

```
<!--[if lt IE 9]>
    <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js">
    </script>
```

We can also run a similar script, 'html5shiv-printshiv.js', to do a similar job with the printed version. This is in your Working Files for this lesson, in a folder named 'scripts', so this needs to be loaded with a local path:

```
<!--[if lt IE 9]>
    <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js">
    </script>
    <script src="scripts/html5shiv-printshiv.js"></script>
```

Finally, close off the conditional comment with `<![endif]-->`, again with no spaces:

```
<!--[if lt IE 9]>
    <script src="http://html5shiv.googlecode.com/svn/trunk/html5.js">
    </script>
    <script src="scripts/html5shiv-printshiv.js"></script>
<![endif]-->
```

Now we have put this in, our styling will work in IE8.

Go back to Internet Explorer. and try switching between the various browser modes and you should find the box layout is OK right back to IE7.

Now we can go on to more detailed styling, of margins, fonts, and colours.

Lesson 21: Styling the header and top navigation

We've styled the box layout for our movies list page. Now we'll add more detailed styling, starting at the top of the page with the header and the top navigation, the list of users and the links to the admin pages.

I want to have sans-serif font for the whole website and there's a very easy way to do this. We can style every item by using the global style selector, `*`, an asterisk. Up at the very top of 'style.css', put:

```
* {
    font-family: sans-serif;
}
```

Save and refresh and we'll see that the font has changed throughout.

And we'll apply a global font size as well, of 16px, and we'll use this fixed size to apply relative sizes to everything else on the page:

```
* {
    font-family: sans-serif;
    font-size: 16px;
}
```

Then we'll give the header a grey background colour using a hexadecimal (hex) code. Hex codes are prefixed with a hash character: `background-color: #F1F1F1`:

```
header {
   height: 50px;
   border-bottom: 1px solid gray;
   background-color: #F1F1F1;
}
```

The top level heading inside the header is targeted using the parent-child selector `header h1`. We'll give it a font size of 1.5em. Ems are a relative size, relative to the global font size we just specified, so 1.5em is 1.5 times 16px, which is 24px. Notice it is `1.5em`, with no space between the value and 'em', and it is 'em', not 'ems':

```
header h1 {
   font-size: 1.5em;
}
```

An advantage of using relative sizes is that if we ever want to change all the sizes we can just change the global size and all the other text will change in proportion – we won't need to make multiple changes.

We'll give it a font-weight of bold, and space it out from the left with a left margin of 50px. The header already has a height of 50px and the way to get text to align vertically to the middle of a box is to apply the same `line-height` property as the height of the box, so we'll give it a line-height of 50px:

```
header h1 {
   font-size: 1.5em;
   font-weight: bold;
   margin-left: 50px;
   line-height: 50px;
}
```

Save and refresh and we'll see the header is styled as we want it.

Now we'll do the `h2` which displays the selected movie-goer's name as a link. This is inside the `select_users` class and is an `h2` with an `a` link inside it, so it is targeted using a complex selector: `.select_users h2 a`.

We want a bold font-weight, a relatively small font size of only 0.9em. I want to override the default link colour, which is blue, and give it a colour of black, and I also want to prevent the default underlining which a link normally has, so I'll assign `text-decoration: none`:

```
.select_users h2 a {
   font-weight: bold;
   font-size: 0.9em;
   color: black;
   text-decoration: none;
}
```

Now there's no way to know it's a link, so I'll give it a different appearance when the mouse hovers over it. This is done using the same complex parent-child selector, with the pseudo-selector `:hover` appended to the end: `.select_users h2 a:hover`.

I'll give it a hover colour with the hex code `#A32824` as this matches Youtube's colour scheme:

```
.select_users h2 a:hover {
   color: #A32824;
}
```

Save and refresh, and hover over that with the mouse to see the colour change.

We'll style the unordered lists in the users and admin menus the same, using `ul.users` and `ul.admin` in a comma-separated list.

As they butt up against the right of the page and against the border of the header at the top, they don't need a border all around, so I'll give them both a border on the left and bottom only, `1px solid grey` in both cases. I'll also assign the same background colour I gave to the header, `#F1F1F1`:

```
ul.users, ul.admin {
    position: absolute;
    right: 0;
    border-left: 1px solid gray;
    border-bottom: 1px solid gray;
    background-color: #F1F1F1;
}
```

So the two navigation menus will look alike but I'll give them different widths. We'll give `ul.users` a width of 200px and `ul.admin` a width of 150px:

```
ul.users {
    width: 200px;
    display: none;
}

ul.admin {
   width: 150px;
}
```

And now for the list items within them. These have `a` anchor tags within the `li` tags so we target them with `ul.users li a` and `ul.admin li a` in a comma separated list. I'll style them both alike, with a small font size of 0.75em, a bit of internal padding to make some space and I'll bulk out the list items a bit with 5px of padding:

```
ul.users li a, ul.admin li a  {
   font-size: 0.75em;
   padding: 5px;
}
```

Now we'll style the hover states for those list items, again both lists alike, using `ul.users li a:hover` and `ul.admin li a:hover`. We'll give them the background colour of #A32824, again matching Youtube's colour scheme, and the colour white for the text:

```
.users li a:hover, .admin li a:hover {
   background-color: #A32824;
   color: white;
}
```

Save and refresh and now if you move your mouse over those links you should see them change colour, but only if you put the mouse directly over the text. It would be better if the hover worked when we mouse over the whole list area.

We do this by assigning to `ul.users li a` and `ul.admin li a` the `display` setting `block`. This make it act as a box instead of just text and has the effect we want:

```
ul.users a, ul.admin a {
    ...
    display: block;
}
```

Save and refresh and finally give `ul.users a` and `ul.admin a` the styles `text-decoration: none` and `color: black`:

```
ul.users a, ul.admin a {
   display: block;
   text-decoration: none;
   color: black;
}
```

And that's how we want the two top navigation lists to appear.

Lesson 22: Styling the favourites and movie list panel

Now we'll style most of the rest of the page – the favourites list, main movie display area, and the footer.

To target the `h2` heading inside the `favs_list` class, we use the parent-child selector `.favs_list h2`. We'll give this a font-size of 0.8em, bold font-weight, and margins of 20px at the top, 10px at the bottom and 5px to the left:

```
.favs_list h2 {
   font-size: 0.8em;
   font-weight: bold;
   margin-top: 20px;
   margin-bottom: 10px;
   margin-left: 5px;
}
```

To style the anchor tag inside the list item in the favourites list, we'll use `ul.favs li a` and as before we'll give it the `display` setting `block`, so that the whole list item area acts as a link, not just the text. We'll give this a font-size of 0.75em, remove the default underlining again with `text-decoration: none`, and remove the blue link colour as before with `color: black`. I'll also give each list item a bit more vertical padding than we gave to the users and admin list items, using `padding: 7px 5px 7px 5px`. This specifies in order the padding at the top, 7px, then the right, 5px, then the bottom, 7px and the left, 5px. This will just make each list item a bit thicker and a bit more prominent:

```
ul.favs li a {
   display: block;
   font-size: 0.75em;
   text-decoration: none;
   color: black;
   padding: 7px 5px 7px 5px;
}
```

As before, we'll have a hover colour of white on the text using the selector: `.favs li a:hover` and the same dark red background colour we used before:

```
ul.favs li a:hover {
   color: white;
   background-color: #A32824;
}
```

Going on to the main move display area, we can style both the movie list area and the movie single area with a 1px dotted grey border:

```
.movie_list, .movie_single {
    margin-left: 225px;
    width: 610px;
    border: 1px dotted gray;
}
```

We'll give the `.movie_list` area 10px padding at the top:

```
.movie_list {
   padding-top: 10px;
}
```

Then we're going to have a welcome message, styled with the class `.welcome`. We'll give this a small font-size, padding all round of 10px except at the bottom where we'll have 20px. We'll give it a 1px dotted grey border on the bottom, and we'll justify the text:

```
.welcome {
    font-size: 0.8em;
    padding: 10px 10px 20px 10px;
    border-bottom: 1px dotted gray;
    text-align: justify;
}
```

Inside the movie list we have a second level heading, which we'll target with `.movie_list h2` to which we'll assign the font-size 1.3em and we'll give it padding of 10px except for the top, which will have 5px:

```
.movie_list h2 {
   font-size: 1.3em;
   padding: 5px 10px 10px 10px;
}
```

We can style the `.movie_single h2` title the same so add a comma and add `.movie_single h2`:

```
.movie_list h2, .movie_single h2 {
   font-size: 1.3em;
   padding: 5px 10px 10px 10px;
}
```

Then down to the unordered list, `.movie_list ul`, containing all the movie details. We'll make a bit of space at the top of the list with 20px top padding:

```
.movie_list ul {
   padding-top: 20px;
}
```

and at the bottom of each movie list item with 10px bottom padding:

```
.movie_list li {
   padding-bottom: 10px;
}
```

The level 3 heading in the movie_list, which has a anchor tag applied to it, is the title of each movie. This is targeted with `.movie_list h3 a`. We'll make the text in it a tiny bit bigger than its default size, 1.1em, and again we'll follow Youtube's colour scheme by giving this link the hex colour code `#438BC5`. I'll assign it bold font weight to make it prominent, and so that I can give it margins, a `display` setting of `block`:

```
.movie_list h3 a {
   font-size: 1.1em;
   color: #438BC5;
   font-weight: bold;
   display: block;
}
```

Now for the margins: 0.5em (half a line height) top and bottom. And I'll take away the default link underlining (text-decoration: none), but, as on Youtube, I'll have the underlining there on the hover state to make it clear that this is a link:

```
movie_list a:hover {
      text-decoration: underline;
}
```

We've done the thumbnail images, by giving them their sizes. Now for the 'figcaption', `.movie_list figcaption`. So that the title and movie description can appear inline with the thumbnail instead of below it, give 'figcaption' the display setting `inline-block`:

```
.movie_list figcaption {
    display: inline-block;
}
```

Align the caption, the title and description, to the top and make some space between the figcaption and the thumbnail with 20px of left padding:

```
.movie_list figcaption {
    display: inline-block;
    vertical-align: top;
    padding-left: 20px;
}
```

Now for the movie description in the movie list, `.movie_list .description`. A font size of 0.75em looks about right to me. It needs a width of 400px so as to fit in the movie list containing box alongside the thumbnail, and we'll give it padding on the right of 10px. Finally, justify the text:

```
.movie_list .description {
   font-size: 0.75em;
   width: 400px;
   padding-right: 10px;
   text-align: justify;
}
```

Lesson 23: Styling the footer

Finally, the footer which is relatively simple.

It's an element so there's no dot in the selector, it's simply `footer`. Remove the green border and give it a top border of 1px dotted grey, and the same on the bottom:

```
footer {
    clear: both;
    height: 60px;
    border-top: 1px dotted gray;
    border-bottom: 1px dotted gray;
    margin-top: 30px;
}
```

The paragraph tag inside the footer is selected using `footer p`. To centre it vertically give it the same line-height as the containing footer element, which is 60px. Match the header with a left margin of 50px, a small font-size of 0.7em and font-weight bold:

```
footer p {
   line-height: 60px;
   margin-left: 50px;
   font-size: 0.7em;
   font-weight: bold;
}
```

It has an anchor tag for the email link, `footer a`. Give this the colour black to remove the default blue link colour. But I'm going to leave the default underlining alone so I won't add `text-decoration none` to remove this. You'll see that the font size is too big. We can solve this using `font-size: inherit`:

```
footer a {
   font-size: inherit;
   color: black;
}
```

Lastly, give it the same dark red colour on hover which we used for the admin links, `#A32824`:

```
footer a:hover {
   color: #A32824;
}
```

Test it in all the browsers. Now we'll go on to add the images we haven't done yet – the profile picture, the little arrow to indicate the admin menu, the trash can, and the heart icon which later we will click on to add items to favourites. For this we'll use a CSS sprite.

Lesson 24: Styling the single movie page

Open up ‘index.html’ and ‘movie_single.html’ in ‘htdocs/favouritemovies’ in Komodo Edit and Firefox. You’ll remember that we have no link to ‘style.css’ in ‘movie_single.html’ and we haven’t yet added the class names, so at the moment all we have is the completely unstyled version.

We don’t want to do repetitive work, so I won’t ask you to type all those class names in again. Instead, close Komodo Edit and go to Chapter 05-10 of the Working Files and copy ‘movie_single.html’ and paste it into ‘htdocs/favouritemovies’, overwriting the current version.

Open it up in Komodo Edit and Firefox you will see that the new HTML file contains the class names and the link to ‘style.css’ that we need so we can go on and just write the relatively few style declarations needed to complete this page – the title, description and the ‘Add to/delete from favourites’ text.

You’ve done a lot of styling in the previous videos, so a lot of what we’ve got to do should be familiar to you. For this reason, I’m going to take a different approach in this lesson and ask you to write the style selectors for the parts of the HTML I identify. I’ll pause the video for a few seconds each time before I give you the answer.

First we want to target the large movie image. You can see in the HTML that it has a class of ‘movie_player’, so how do you target that in the css?

As the class name is unique and we don’t expect to use it anywhere else, we can just use the class name on its own, `.movie_player`. There is no need for a parent-child selector.

The main movie image needs a width of 608px and a height of 344px, giving it a 16:9 ratio:

```
.movie_player {
   width: 608px;
   height: 344px;
}
```

Next we want to target the movie title. Here it is, inside `<h3>` tags, which are inside the ‘movie_single’ class. How do you target that in the css?

The answer is with a parent-child selector, `.movie_single h3`.

Now give this a font-size of 1.3em and padding all round of 10px:

```
.movie_single h3 {
   font-size: 1.3em;
   padding: 10px;
}
```

Next, give all paragraph text in the ‘movie_single’ class a font-size of 0.8em and padding all round of 10px:

```
.movie_single p {
   font-size: 0.8em;
   padding: 10px;
}
```

Now what’s the selector for the movie description?

Hopefully you got it right – it should be `.movie_single .description`. We need to use a parent-child selector to distinguish this description from the one on the movie list page.

We'll give this a font-size of 0.75em, padding all round again of 10px and this time we'll assign a minimum height of 80px, using the min-height property. We'll also justify the text in it:

```
.movie_single .description {
   font-size: 0.75em;
   min-height: 80px;
   text-align: justify;
}
```

Check that that looks OK with more text in it before going on.

Now all we've got to do is the 'Add to/remove from' text. You'll notice in the HTML that we now have two nested divs, an outer div with the class name 'actions', which we will use to set one set of style properties, and an inner one called 'add_remove', to which we will apply another set of properties to get the look we want.

How do you target the actions class? As it is unique, we can do without a parent-child selector and use just `.actions`. Give it `position: relative`, a height of 37px, and a grey bottom border using `1px solid #CCC`:

```
.actions {
   position: relative;
   height: 37px;
   border-bottom: 1px solid #CCC;
}
```

Now inside that, target the 'add_remove' class. This time, we need to specify that this applies only when it is inside the actions class, so we need to use the selector `.actions add_remove`. Give this absolute positioning, with `right: 10px`:

```
.actions .add_remove {
   position: absolute;
   right: 10px;
}
```

This will position it inside and to the right of the 'actions' class. We needed relative position on the containing actions class in order to use position absolute inside it.

Now target the `<p>` tag inside the 'add_remove' class and give it a slightly smaller font size of 0.75em, bold font-weight and a grey colour using hex code #999:

```
.actions .add_remove p {
   font-size: 0.75em;
   font-weight: bold;
   color: #999;
}
```

Finally we'll give it a Youtubesque hover state with a black text colour and a bottom border 3px thick, solid, and with the colour `#993300`:

```
.actions .add_remove p:hover {
   color: black;
   border-bottom: 3px solid #993300;
}
```

The height of 37px on the outer div makes this bottom border line up over the grey border of the 'actions' div. That's the single movie page complete and now we can go on to the admin pages.

Lesson 25: Styling the admin pages

In this lesson, we'll follow the same procedure as in the previous one to style the admin page, 'admin.html'.

I'll ask you to type in the style selectors before I give them to you. This time, though, I'll also ask you to type in some of the style declarations in full as well.

As before, we'll save unnecessary typing by using a prepared version of the admin.html file with its classes already completed.

I've also found a slight error in the previous lessons – I was using the class 'admin' both for the menu in the navigation part of the page and for the main section of the admin pages.

I've changed this so that the navigation part has the class names 'users_menu' and 'admin_menu' so as to avoid any confusion, and I've made the relevant changes in 'style.css' and the other HTML files.

So this time, go to the Working Files, 'Chapter 05-10 Styling the single movie page', and copy everything in there, all the files and folders, and paste it all into 'htdocs/favouritemovies', overwriting the files already there. This will correct that error and give us what we need to go on.

Open the new version of 'admin.html in Komodo Edit and open 'style.css' in another tab.

We'll look at 'admin.html' – as before I have supplied the class names to save time.

The main part of the page consists of an HTML table, enclosed in a section with the class 'admin_users'. The table has the class name 'admin_table', the table header cells have the names 'data_col' for the cells containing the first names and last names of the movie goers, and 'admin_col' for those which we will use for the add and delete actions later.

Below that, the table rows which have existing data are named 'datarow' and the input boxes are named 'data'. The cell on the right of the existing data is where we will click to delete data so this is called 'deletecell'. Inside this cell we have a div, which we'll use to position the background image.

The bottom row of the table will be used to insert new data. This has the class name 'newdatarow', the input boxes are named 'newdata' and the cell on the right, where we will click to insert the data is named 'insertcell', again containing a div.

Now we'll write the styles to go from this unstyled version to this at the end of this lesson. In 'style.css', write the selector to target the section with the classname 'admin_users'.

You should have entered `.admin_users`, with a dot:

```
.admin_users {

}
```

Push the 'admin_users' area in from the left and down from the top now with margins on the left of 50px and top of 40px:

```
.admin_users {
   margin-top: 40px;
   margin-left: 50px;
}
```

Now select the `<h2>` tag, but only when it appears inside the 'admin_users' class, and give this a large font size of 1.25em and a bottom margin of 20px:

```
.admin_users h2 {
   font-size: 1.25em;
   margin-bottom: 20px;
}
```

Now target the 'admin_table' class itself.

You should have entered `.admin_table` as the selector.

Give this `border-spacing` of 2px and `border-collapse: separate`. This will put 2px spacing between the table cells, which I think looks better:

```
.admin_table {
   border-spacing: 2px;
   border-collapse: separate;
}
```

Then select the `<th>` (table header) elements inside the admin table.

The selector should be `.admin_table th`.

Give this bold font weight, make the text align to the left, and give it a background colour with the hex code #F1F1F1:

```
.admin_table th {
   font-weight: bold;
   text-align: left;
   background-color: #F1F1F1;
}
```

Target both the `<td>` and `<th>` elements in the admin table.

You should be using the selectors `.admin_table td` and `.admin_table th` separated by a comma. Give these a 1px solid black border:

```
.admin_table th, .admin_table td {
   border: 1px solid black;
}
```

Then give the `<td>`, `<th>`, and `<input>` elements in the 'admin_table' table a height of 35px.

You should have used `.admin_table td, .admin_table th` as the selector, and `height: 35px` as the style:

```
.admin_table th, .admin_table td {
   height: 35px;
}
```

This makes the table rows higher than normal.

We also want the input boxes in the table cells to fill the table cells vertically. You write the selector for that.

We add `.admin_table input` to the comma-separated list of selectors:

```
.admin_table th, .admin_table td, .admin_table input {
   height: 35px;
}
```

This makes the text input boxes fill the table cells vertically.

Give a fixed width to the data columns and input boxes of 200px:

```
.admin_table th, .admin_table td, .admin_table input {
   height: 35px;
   width: 200px;
}
```

Then target 'admin_col', the column on the right, and give this a width of only 80px.

Your style declaration should be:

```
.admin_table th.admin_col, td.deletecell, td.insertcell {
    width: 80px;
}
```

At the moment all the text in the table butts up against the border. Select the `<th>` element and the input box inside the admin table and give them padding of 5px on the left and right:

```
.admin_table th, .admin_table input {
    padding-left: 5px;
    padding-right: 5px;
}
```

The table should now have some internal padding.

The text in the input boxes is too big. Target it and give it a font size of 0.9em:

```
.admin_table input {
   font-size: 0.9em;
}
```

Using the same selector, remove the border round the input box with `border: none`:

```
.admin_table input {
    font-size: 0.9em;
    border: none;
}
```

The final version of the admin table has visual feedback in the form of this yellowish background colour that our cursor is in an input box, that is that it has focus.

Target all input boxes in the admin table and then, as we did before using the 'hover' pseudo-selector, append the `:focus` pseudo-selector to style the focus state, using #FEE5AC as the colour code.

You should have `.admin_table input:focus` as the selector:

```
.admin_table input:focus {
    background-color: #FEE5AC;
}
```

We are missing a border on the `<th>` and `<td>` elements – they need a 1px solid black border:

```
.admin_table th, .admin_table td {
   border: 1px solid black;
}
```

Now we can try the page in the various browsers. Firefox - OK; Opera – OK; Chrome – there's an ugly brown outline round the input boxes – and in Safari an equally ugly blue outline. In IE there's a peculiar nasty cross inside them.

We can fix the outline with `outline: none, border: none` on the `input` selector – this will fix any inputs we use on the whole site:

```
input {
   outline: none;
   border: none;
}
```

Lastly, we can fix the IE problem with the Microsoft-specific pseudo element: `input::-ms-clear` and set it to `display: none`:

```
input::-ms-clear {
   display: none;
}
```

That deals with that and now we'll move on to the last lesson on styling, how to add the various background images – the trash can and its hover state, the profile picture, the add/delete buttons and the favourite icon.

Finally, save and refresh and check that everything looks as it should in all three HTML files.

Lesson 26: Creating background images with a CSS sprite

We're going to use a single image for all the background images on the site – the trash can and its hover version, logged in and logged out versions of the profile picture, and the two states of the favourites icon, which changes to a success icon when we remove a movie from the list of favourites.

In the admin pages, we have a delete icon, again in two colours, an insert icon, and another success icon. Apart from the AJAX loader icon you've seen as the database updates, all these backgrounds are actually just one image, called a CSS sprite.

The AJAX loader can't be part of the sprite because it's an animated GIF and can't go into the PNG file which we'll be using for the other images.

CSS sprites are a way of using only one image and then displaying just the right bits of it in the right places by shifting its position so that only a little bit of the image is displayed in each place. The technique is extended so that a different part of the image appears when the mouse hovers over an element.

The advantage of this is only one larger image is loaded and is then reused, rather than loading many smaller images, so there's no delay waiting for the hover state to appear. This also reduces the number of HTTP requests being made and speeds up page loads.

You can read more about CSS sprites at CSS-tricks.com: http://css-tricks.com/css-sprites/.

I'll show you how to make a CSS sprite out of individual images using an online sprite generator, and how to make the CSS we need to shift the background position of the image around, but as the process is a bit fiddly I'll also give you the final code and a prepared sprite and it might be easier to use that.

Go to the Working Files, 05-12 'Creating background images with a CSS sprite', and, in the 'individual images' folder, you'll find the individual images which will appear in various parts of the finished website – the trash can, profile pics, *etc*...

Now in your browser, navigate to http://spritepad.wearekiss.com. This online sprite generator allows you to drag your images onto the spritepad and gives you the CSS for them automatically – each one will have a negative position associated with it to move the position of the sprite so that only the correct portion of it is seen in each place.

Drag the images onto the spritepad and position them with a 1px gap between them. On the right, the corresponding CSS is generated, for you to past into your stylesheet.

That's the principle. If you want to, you can go on and make your own sprite. When you have positioned all the images neatly on the spritepad, keeping the image dimensions as small as you can, click on 'Download' and you'll get a zip file which contains the sprite as a PNG image and all the CSS code you need.

Rather than making you go through the whole process, though, I'll give you the finished sprite and CSS and I think it might be easier to use that.

Go to the Working Files for this lesson and in the 'images' folder you'll find 'sprites.png'. Copy this to 'favouritemovies', make a new folder called 'images' and paste 'sprites.png' in there.

Now go back to the sprites folder in the Working Files and open up 'sprites.css' in the 'css' folder. Select everything in it, noticing that it follows the principle we have just learnt about, select all and

copy it, and open up your ‘style.css’ file in ‘favouritemovies/css’ and paste it all in just above where the styles for the footer start.

We also need to copy and overwrite ‘admin.html’ and ‘movie-single.html’ because I’ve added in a couple of divs to hold the profile picture.

Open up the three pages in your browser and the background images should all be working. Check their hover states – they should all change.

Check in all browsers – all should be OK – and check that the page validates successfully at http://validator.w3.org/#validate_by_input.

That’s our styling complete and now we’ll start the real programming – learning the essentials of PHP.

Chapter 6: Essential PHP

Lesson 29: Introduction to PHP

Before we begin working on using PHP to make our project interactive, able to take input from the user and act on it, we need to step aside and have a look at PHP – what it is and how it works. As always, if you feel you're familiar with the basics of PHP, you can skip any or all of this chapter – this while chapter is a series of small exercises and no changes are made to the project files – so you can skip it entirely or dip in and out as you want. Each lesson ends with a little piece of homework and it might be good in any case to do that, because each one covers an important point that will be needed when we get back to work on the project.

The name 'PHP' is a strange abbreviation. It's a recursive abbreviation, which means it stands for itself and means 'PHP – Hypertext Pre-processor'. It's the first true programming language we have looked at so far, meaning that it is the first language we have seen which can make decisions; it can do different things according to the data it receives.

PHP is specifically designed to work together with HTML and most PHP files are a mixture of PHP and HTML. The PHP supplies the parts of the document which changes – data of some sort, such as products or in the case of our project a particular movie-goer's list of favourites at a given moment. The HTML supplies the parts which do not change – the structure and layout of the page.

The official website for PHP is php.net – you can find the online manuals for PHP and all its built-in functions there.

In the next video I'll demonstrate in a simple example how PHP and HTML are combined in one file.

Lesson 30: Combining PHP and HTML to display strings, variables, and HTML tags

In this video I will show various ways in which to combine PHP code with ordinary HTML, which is what we do in nearly every PHP file.

In Windows Explorer, go to c:\xampp\htdocs and alongside the favoritemovies folder, make a new folder called php-demos. Inside php-demos, make a new text file and name it 'php-and-html.php'. If a file has any PHP code in at all, it must end in a .php extension, to TELL the server that it is a PHP file.

Using this file I'll demonstrate how to combine PHP which will produce dynamic content, with ordinary HTML which gives the static content, the framework of the webpage. As always, you can find the completed version in the Working Files.

Open up php-and-html.php in Komodo Edit and type in opening and closing PHP tags:

```
<?php
?>
```

All PHP commands have to go inside these tags, to separate them from the ordinary HTML.

Inside the PHP tags, type in:

```
<?php
echo date('l');
?>
```

exactly as it is, making sure the case is right and putting the semicolon on the end.

'Echo' is like a print command – it prints whatever follows to the browser screen.

This date command is followed by one parameter, lower case 'l', which tells the server to return the current day in its full form.

Open up the file in http://localhost/php-demos/php-and-html.php.

You must use localhost as the URL – opening it by right-clicking and selecting a browser will not work as you will not be opening it using the server. It should display the name of the day.

Then on the next line, enter `echo "<br>";` to print an HTML line break tag.

Now add below that line:

```
<?php
echo date('l');
echo "<br>";
echo date('H:i:s');
?>
```

This echoes, or prints, the hour in 24-hour clock format, that's the 'H', followed by a colon, followed by minutes, that's the 'i', another colon, and then seconds, that's the 's'.

Refresh the page in your browser and you should see the time – and if you keep refreshing it, you'll see the seconds changing – so this is our first step into displaying dynamic data drawn from the server. If the time is wrong, it is probably because you didn't do the lesson on changing your timezone earlier.

And after that echo another `<br>` tag in the same way as before and then `echo date('d M Y');` This will add the date, in day-month-year order, with a space between each.

Now we might want to label these values – day such-and-such time such-and such date such and such.

To do this, instead of echoing the various date functions straight out it would be neater to store each of them as a variable first and then echo the variables out, combined with the labels and HTML for formatting. I'll do this, demonstrating how bits of PHP code are interspersed in ordinary HTML.

Change `echo date('l');` to `$day = date('l');` Now the program no longer prints out the day to the screen – instead it saves it as a variable.

Variables are a way of storing a value by assigning a name to it. This allows a value to be identified for re-use throughout the program, and means it can be manipulated or changed later.

The contents of a variable can be numbers, text, dates and time, or a whole list of values, in which case the variable is called an array. We'll look at arrays a bit later.

In PHP, all variable names begin with a dollar sign and are case sensitive, so `$day` and `$Day` are different variables. In PHP it is best to assume that everything is case sensitive and make sure you type everything is exactly as it is presented to you.

Now we'll store the rest of the date and time values as variables. At the end of this the code should read:

```
$date = date('d');
$month  = date('M');
$year = date('Y');
$hours = date('H');
$minutes  = date('i');
$seconds = date('s');
```

Now all these values are stored as variables, but nothing will be displayed in the browser because we haven't done anything to display them.

Now I can demonstrate various ways of combining PHP and HTML as we display the data. We can use echo as before to print out on of the variables:

```
echo $day;
```

and then enclose it in double quotes and add a pair of html tags either side:

```
echo "<p>Today is $day</p>";
```

If you try removing the double quotes you will find it no longer works and you get a syntax error:

```
echo <p>$day</p>;
```

This is because this is nonsense to the server. If we want to combine HTML and PHP in an echo statement like this we must enclose it in double quotes.

Now try putting single quotes where we had double quotes;

```
echo '<p>$day</p>';
```

This is interesting – you no longer get a syntax error, but now you get the name of the variable, literally '$day', printed out to the screen instead of its value.

In PHP, as in many other programming languages, single quotes prevent their contents being interpreted. Instead we get the literal contents of the single quotes printed out verbatim.

Put back the double quotes so that it works as we want it to.

That's one way to combine PHP output with HTML tags – to wrap the output of an echo statement containing HTML tags and a variable in double quotes.

Another way is to reverse the order of the elements and enclose the PHP inside the HTML tags.

After the closing PHP tag, start typing ordinary HTML:

```
<p>Today is
```

and then inside that we'll embed PHP tags enclosing the echo statement:

```
<p>Today is <?php echo $day;
```

and close the PHP tags with `?>`:

```
<p>Today is <?php echo $day; ?>
```

and then we're back to HTML and we can close the `</p>` tag:

```
<p>Today is <?php echo $day; ?></p>
```

The result on the screen is exactly the same.

Take a look now at the source code in the browser by right clicking and choosing 'Source' or 'View Source' and you will find that there is no PHP code there at all – only ordinary HTML.

This is because PHP is a server-side language. All the work interpreting it is done by the server and the browser never sees the PHP code at all. This makes PHP very suitable for applications where security is needed because if it's written and configured properly there should be no way for a user to find out how the application works or discover secrets like passwords.

We have seen two ways of combining PHP and HTML.

A third way is to use concatenation, or joining strings together. I'll exemplify this with simple text strings first and after that with a text string and a variable.

In PHP, concatenation is performed with a dot operator. Quotes, single or double according to our needs, surround the parts which are ordinary HTML, and these are concatenated onto the dynamic PHP bits, using a dot.

So `echo "<p>Today is $day</p>";` can be rewritten as: `echo "<p>Today is " . $day . "</p>";` enclosing the HTML in double quotes, then a concatenation dot, and then the PHP variable outside the quote marks, then concatenation dots, and finally the remaining HTML in double quotes.

We must use the concatenation method if we want to perform any operations on variables.

Assign the value of 5 to the variable `$x` using `$x = 5`:

```
echo "<p>Five plus one is $x + 1</p>";
```

will not work.

```
echo "<p>Five plus one is " . ($x + 1) . "</p>";;
```

will perform the operation and echo the result.

Here's a bit of homework for you to practice concatenating strings and variables in this way. Using whichever of these methods you prefer, or any combination of them, try to echo the following message:

"The date today is Monday the 3rd of June 2013".

Including the full-stop at the end of the sentence.

To get the suffix for the date, usually '-th' but '-st' if the date is 1, for first, '-nd' for second, the parameter for the PHP date function is upper case 'S'.

The answer is in the Working Files.

The date function is a very important one but it's not used in the favourite movies project at all so in case you want to know more about the various parameters it takes, I've included a file in 'php-demos' called 'date-and-time.php', and in the bookmarks there is a link to the relevant page at php.net.

So far we've used quote marks to output PHP and HTML – but what happens if quote marks themselves are part of what we want to display?

We'll look at this in the next lesson.

Lesson 31: Echoing quote marks – String delimiters

There are two possible solutions to the homework question in the file 'concatenation-homework.php' in the Working Files. The first uses double quotes to enclose the whole of the string, with the variables included with the text. Because we cannot work out the value of $year + 1 using this method, we have to work it out in advance and store it as a new variable, which I have called $next_year, and then echo this out.

The second method uses concatenation dots, which looks more cumbersome but allows us to perform the maths inside the echo statement.

You can use either, or a combination – it makes little difference – whatever is more convenient and easier to read.

We have seen that double quotes can be used in echo statements to combine variable values with html tags and we saw that if we use single quotes enclosing a variable, the literal name of the variable itself will be output, rather than its value.

What happens if we need to store or echo quote marks themselves as part of a string? What if I want to output "That's nice", he said. with the quotes as part of the string? Or tomorrow's date will be, with an apostrophe?

We use double quotes to enclose single and vice versa. So to output single quotes we could use:

```
echo "'Step outside', he said threateningly";
```

Or to output double quotes we enclose them with singles:

```
echo '"Step outside", he said threateningly';
```

The outer, enclosing quote marks delimit the beginning and end of the string so these are called the string delimiters.

If we need to output the same quote marks as those we using as the string delimiters, then we have to precede the output quote mark with a backslash:

```
echo '"That\'s nice", he said';
```

This backslash is the 'escape character' – it removes any special function from the next character and allows it to be processed as ordinary text. So in this case the function of delimiting the beginning and end of the string is removed from the following quote mark.

If we use double quotes as the string delimiters, we have to escape the double quotes in the output, not the single one:

```
echo "\"That's nice\", he said";
```

This will be put into use later on when we have to store HTML code containing quote marks as variables.

Try using a backslash in front of the dollar sign in a variable name and see what the effect is. As you probably expected, the dollar sign loses its function as a variable identifier and the literal string is returned.

For your assignment this time, use ‘time-and-date.php’ to adapt the concatenated string you made in the previous lesson so that it reads:

> “The time now is 6:54pm (or whatever the time is). In two hours’ time it will be 7:54pm”, said the timekeeper.

The answer is in the file ‘string-delimiters.php’ in the Working Files.

Lesson 32: PHP conditions

Now we'll look at possible the most important feature of every programming language – the ability to make decisions and carry out different actions according to the data presented to the program.

Open up 'php-and-html.php' and save it in 'php-demos' with a new name, 'conditions.php'.

Keep the variable assignments but delete the echo statements below them.

Now we will write a series of conditions which will make the program print out different messages according to the values of the date and time variables.

Below the last line of the variable assignments, type in

```
if($day == "Sunday") {
    echo "Have the day off!";
}
```

We have started a conditional clause, the first programming decision we have written. The 'if' starts the decision and evaluates whether the condition in the round parentheses is true. If and only if the value of the variable $day is "Sunday", then the code inside the curly braces will run and the message "Have the day off!" will be displayed in the browser window.

If today is not Sunday, you will have to wait until next Sunday to see this work.

Actually, I'm joking. There's an easier way to test if the program works and that's by temporarily overriding the value assigned to the $day variable – so test the Sunday condition by adding `$day = "Sunday";` after the variable assignments. Once we have tested the program with dummy data assigned manually, we can remove this and use the correct date and time data from the server.

When we test whether two sides of a condition are equal, we must use double equals, `==`, because this is not a statement – it is a test. Single equals is quite different – that it is an action – it assigns a value to a variable as we have done above. Putting single equals in conditions and forgetting the parentheses around conditions are two common programming errors which will lead to bugs in the program.

Notice the indentation, which Komodo Edit has applied automatically. We indent logical sections of the code to make the structure of the program easier to discern.

It would be nice to be able to echo a message which prints on either Saturday or Sunday. We do this using an OR test, which is written with a double pipe symbol. The pipe symbol is normally to the left of the space bar or the Z key on your keyboard.

To make an OR conditional test we write:

```
if($day == "Saturday" || $day == "Sunday" ) {
    echo "I hope you're having a good weekend!";
}
```

Let's put in a bit of HTML, as we learnt in the previous lesson, by surrounding this with <p> tags.

```
    echo "<p>I hope you're having a good weekend!</p>";
```

Let's make another conditional clause, above the previous one:

```
if($hour > 12) {
    echo "<p>Good afternoon</p>";
}
```

As you can see, this time the part inside the curly braces will run only if the value of the variable 'hour' is greater than 12.

To try to print out a message in the evening I could add

```
if($hour >= 18) {
    echo "<p>Good evening</p>";
}
```

But what will happen when the hour value is greater than 18, or 6pm? We will get both "Good afternoon" and "Good evening", which we don't want.

To overcome this, you need an AND condition, which we write using &&:

```
if($hour > 12 && $hour < 18) {
    echo "Good afternoon";
}
```

will solve that problem.

For the "evening" greeting we could use

```
if ($hour >= 18 && $hour < 23 ) {
    echo "Good evening";
}
```

Here's another little task for you. Use `if`/`&&` conditions to produce five different greetings according to the time of day:

"Good morning" from 6am to noon,
"Good afternoon" from noon to 6pm
"Good evening" from 6pm to 10pm
"Good night" from 10pm to 11pm
"Go to bed" from 11pm to midnight
"You should be in bed and fast asleep!" from midnight to 6am

The answers are in the Working Files.

Lesson 33: Switch ... case

As you can imagine, 'if' clauses can get quite complicated. There's an alternative which is sometimes more suitable, especially when we have a list of many different options to choose from, rather than an either/or choice. This is called 'switch'.

Make a new php file called 'switch.php'. Assign the value "Monday" to a variable named $day and then use

```
switch($day) {
}
```

to start the switch clause.

Switch takes an argument in parentheses and is followed by braces. The braces enclose a series of 'cases', with `case`, then the value we are looking for in quotes, then a colon.

So we can put inside the braces:

```
switch($day) {
    case "Monday":
        echo "Oh dear, back to work!";
}
```

Then we need `break` to break out of the switch clause and continue with the script after it.

```
switch($day) {
    case "Monday":
        echo "Oh dear, back to work!";
        break;
}
```

We can add the 'case' that it's a 'Saturday':

```
switch($day) {
    case "Monday":
        echo "Oh dear, back to work!";
        break;
    case "Saturday":
        echo "Horray! The weekend's here";
        break;
}
```

Try it in the browser and change the value of $day to "Monday" or "Saturday" to see it working.

We can also match several different values by making a list of cases separated not by commas but by semicolons, and we have to have the word `case` each time:

```
switch($day) {
    case "Tuesday"; case "Wednesday"; case "Thursday"; case "Friday":
        echo "Ho hum diddlee dum another day at work";
        break;
    case "Saturday"; case "Sunday":
        echo "Life's gooood";
        break;
}
```

We'll use 'switch' and 'case' extensively in the project.

Lesson 34: PHP loops

Another key feature of all programming languages is some way of running sections of code repeatedly, by looping through lines of code according to some rule governing how many times and under what conditions they should be run. Using loops, programs can perform repetitive work very quickly and loops form a very important part of many tasks. In this video we will look at two types of loop – 'while' loops and 'for' loops.

Make a new text file and save it as 'loops.php', again in the php-demos folder.

Inside PHP tags, type:

```
$count = 1;
while ($count <=10 ) {
}
```

and then inside the braces :

```
$count = 1;
while ($count <=10 ) {
    print $count . "<br>";
     $count = $count + 1;
}
```

What do you think this does?

It prints out the numbers from 1 to 10, with line breaks in between.

Going through it line by line, first we assign the value of 1 to the variable `$count`. Then we start the loop, to run while the variable `$count` is less than or equal to 10. So the code inside the braces will run while `$count` is less than or equal to ten, and will not run if it is not.

Then we add 1 to the value of `$count`, using `$count = $count + 1`.

So the first time through, the value of `$count` is 1 because it is set here, and "1" is printed out, followed by a line break, using the concatenating dot we have just learnt about. `$count` has 1 added to and becomes 2. The `while` condition is still true so it runs again, prints out '2' and adds 1 to the value of `$count` again.

This continues until the value of `$count` is 10 – the loop runs one last time, and adds 1 to make `$count` 11, at which point the condition is no longer true so the loop stops.

It is common to change the incrementing function, `$count = $count + 1`, to a shorthand version: `$count++`, which does exactly the same thing.

The `for` loop is a refinement of the `while` loop. It removes the need for the count number to be incremented manually, but the code in the condition is more involved.

Type in :

```
for ($count=1; $count<=10; $count++) {
      echo $count . "<br>";
}
```

This does exactly the same thing as the `while` loop but sets the initial value of `$count`, the maximum value of `$count`, and the amount to increment it by each time the loop runs all in the condition.

Notice that the three parts of the condition, the initial value, the final value, and the increment value, must occur in this order and must be separated by semi-colons.

We will use both `for` and `while` loops in our project to display data.

Your assignment this time is to use `for` or `while` loops to print out a grid of asterisks 20 rows wide and 10 columns high.

Lesson 35: PHP includes

In this video, we'll learn about PHP commands used to include the contents of one file inside another one. We'll do this now using extremely simple code and we'll apply the same method to the Favourite Movies project later.

In 'php-demos', make a new text file and call it 'include-files.php', open it up in Komodo Edit and type in opening and closing PHP tags.

Then inside the php tags, type the line

```
include 'file1.txt';
```

Then copy and paste this line two more times, changing 'file1.txt' to 'file2.txt' and 'file3.txt':

```
include 'file1.txt';
include 'file2.txt';
include 'file3.txt';
```

Now open a new code pane and make a file called 'file1.txt' and put the word 'apples' in it and save it. Then change that to 'bananas' and save it as 'file2.txt' and then 'grapes' and save that as 'file3.txt'.

Now run 'include-files.php' in your browser and you should see the output 'applesbananasgrapes', all as one word. The PHP script has been acted upon and has output the content of the three text files to the browser window. The contents of the three text files have been included in 'include-files.php' just as if they were part of it. There are no spaces or breaks between the words because we didn't put any in.

We can tidy up the output by echoing line breaks between the `include` commands:

```
echo "<br>";
```

At present we have saved our content files, the three .txt files, in the same folder as the PHP file which uses them. Normally, we wish to separate content from code as much as we can, and we would do this by putting the content files in a separate folder. Let's do this now.

Create a new folder in 'php-demos' called 'includes' and move the three included text files into it. Now edit 'include-files.php' so that the relative path points to this subfolder. Save and refresh and you should see no changes, but the site is now better organised as we have our text files with content in a folder of their own, separate from the PHP code.

We have used the PHP 'include' command. This will do exactly as it says and include those files every time it appears. So if we copy all the include lines, they will all be included a second time and the list of fruits will get longer.

Sometimes in real programming this causes problems and we may not want an include file to be included repeatedly.

The solution to this is to use `include_once` instead of `include`. If you use Edit > Replace > Replace All to replace all the includes with `include_once`, put a space after it to stop it replacing the folder name as well. Save the file and refresh it in your browser your will see that each file is now included once only.

There is a close relative of `include` and `include_once` – `require` and `require_once`. The difference between them is as the name suggests – with `require` and `require_once`, the included file is vital and must be found. If it is not found, the program will stop, whereas with `include` the program will give a warning and continue.

To see this in action, go into 'php-demos/php-includes' and rename 'file1.txt' to something else, so that the included file no longer exists. Refresh 'include-files.php' in your browser and you will see that there is an error message but the other includes still work, so the program continued to run.

Change 'include_once' to 'require_once' or 'require', save and refresh, and you will see that there is an error message and the program stops altogether.

So we can see that including files inside a PHP file has exactly the same effect as if we put all the code in one big file.

So far we have just included plain text files. Let's look at what happens to variables if we include PHP files in which values are assigned to variables and then use those variables in the including file.

Open up 'include-files.php' in Komodo Edit and change all the .txt extensions to .php. Delete the lines echoing 'br' tags.

Then make a new file, 'file1.php', in the includes folder, and in it, assign a value to the variable `$fruit1`.

Copy this and make two more .php files, 'file2.php' and 'file3.php' and assign values to the variables `$fruit2` and `$fruit3`.

Then, back in 'include-files.php', echo the values of the three variables out. You can revise using concatenation or one of the other ways of combining strings and variable values by echoing out "My three favourite fruits are ..." and then the values of the three variables.

You will find that the values we assigned to the variables `$fruit1`, `$fruit2` and `$fruit3` are available in the parent file, 'include-files.php' and we can echo their values with no problem.

This makes it very easy to split our code into small files, each with a single, well-defined purpose, secure in the knowledge that any variables we set in any of the included files will be available in the parent file.

We will be using PHP includes later on in our project, which will make the coding much more efficient and will reduce code duplication.

Lesson 36: Arrays

Earlier on we looked at assigning values to variables. In each of those examples, a variable had a single value – a string value like “Tom” or a numerical value like the current date or an increment number in a loop.

Let’s suppose we want to store a set of employee records. We could try to do this using a series of individual variables of the sort we used earlier, for instance:

```
$firstname = "Michael";
$lastname = "Thompson";
$payroll_number = "51687";
$salary = "26000";
```

but the problem with this is that it is only any good for one employee. Once we have more employees, we would have to set up different variable names for each one, like `$firstname2`, `$lastname2`, *etc.*, which would rapidly become unmanageable, and there would be no clear relationship between these variables.

We solve this using arrays.

An array is a type of variable which can contain sets of multiple values in a structured list. In this video we will look at how arrays are set up and how we can return the values stored in them.

In Komodo Edit, save a new PHP file as ‘arrays.php’ in ‘php-demos’. As always, the completed version of this is in the Working Files, should anything go wrong.

Start by printing out a top level heading for the page using:

```
echo "<h1>Methods of returning data from arrays</h1>";
```

Then we will use <h2> tags to head lists of all employees and <h3> tags for the titles of individual employees.

To set up an array to store the employee data in the example, we could use

```
$employee1 = array("Michael","Thompson","51687","26000");
```

Put in a title for our results:

```
echo "<h3>Details of Employee 1 (Manual method):</h3>";
```

Then we could return the values of the array using an echo statement of the sort we learnt about earlier:

```
echo "First name: " . $employee1[0]  . "<br>";
echo "Surname:  " . $employee1[1] . "<br>";
echo "Payroll no.:  " . $employee1[2] . "<br>";
echo "Salary: $" . $employee1[3] . "<br>";
```

This is called an INDEXED array because we access each value by referencing it using an index position, starting from zero. So `$employee1[0]` returns the FIRST element in the array, the first name. There are four elements altogether so the last one, salary, is referenced using `$employee[3]`.

This type of array, in which each element is identified by an index number starting from zero, is called a numerically indexed array.

Because we have that index number, which increments from 0 to 3 as we go through the array, this would be a perfect opportunity to use a loop to shorten the code and automate the process of getting the data printed out, just as we learnt earlier in the course.

We need to know how many elements there are in the array and we get that information by using the PHP function count, wrapped around the array name:

```
$count = count($employee1);
```

If we temporarily echo the result of that out using `echo $count`, we will see that it has given us the total number of elements, 4, not the highest index number, which is 3.

Comment that line out and put in a second title for this method of getting the results:

```
echo "<h3>Details of Employee 1 (Loop method):</h3>";
```

We can now use the value of $count to write a 'for' loop just as we did before, using

```
for ($x=0; $x<$count; $x++) {
}
```

`$x` is the name for our incrementing number, which starts at 0, the first index number in the array, and stops when it is LESS THAN `$count`. As the value of `$count` is 4, it will stop at 3, which is the highest index value in the array `employee1`, exactly as we need.

`$x++` increments the value of `$x` by 1 each time.

Inside the braces, put

```
for ($x=0; $x<$count; $x++) {
    echo $employee1[$x] . "<br>";
}
```

This prints out the value of `$employee[0]`, the first name, the first time through the loop, `$employee[1]`, the surname, the next time through until it gets to `$employee[3]`, which is the salary, and stops.

You can temporarily put in the line

```
for ($x=0; $x<$count; $x++) {
    echo '$x = ' . $x . " ";
    echo $employee1[$x] . "<br>";
}
```

above the line `echo $employee1[$x]` to see the value of `$x` incrementing each time through the loop, and then delete that line.

Now we can go one step further and use arrays to store not just one employee record but many. Below the line assigning values to `$employee1`, set up two more employee records, called `$employee2` and `$employee3`:

```
$employee2 = array("James","Rasmussen","19875","15000");
$employee3 = array("Henry","Foggett","19810","150000");
```

Then add the line:

```
$employees = array($employee1,$employee2,$employee3);
```

This adds the contents of all three employee arrays to a new array, `$employees`, so now we have an array of arrays, or multi-dimensional array.

To return the first name of `$employee1` manually, we would use `$employees[0][0]` – the first zero refers to the first employee record in the `employees` array, and the second zero refers to the first record inside that employee's record.

Question: what pair of index numbers would return the salary of Henry Foggett?

Answer: `$employees[2][3]`.

To return all the data in the $employees array manually we would use:

```
echo "First name: " . $employees[0][0] . "<br>";
echo "Surname:  " . $employees[0][1] . "<br>";
echo "Payroll no.:  " . $employees[0][2] . "<br>";
echo "Salary:  " . $employees[0][3] . "<br><br>";
echo "First name: " . $employees[1][0] . "<br>";
echo "Surname:  " . $employees[1][1] . "<br>";
echo "Payroll no.:  " . $employees[1][2] . "<br>";
echo "Salary:  " . $employees[1][3] . "<br><br>";
```

We are iterating through each of the employee arrays using the first index number and within that we are going through each one returning each item of data, firstname, lastname, *etc.*. in order.

As you can see, this is very repetitive code, and if we had an array with not three employees with 4 records each, but several hundred employees each with many records, it would be unusable.

What we need is a way of looping twice through these arrays, first through the outer one, the list of employees, and then within each employee through all the details about that person.

If you managed to do the homework on loops you did exactly this – by running one loop inside another.

Start out by putting in a title:

```
echo "<h2>Details of All Employees (Double loop method):</h2>";
```

First, set up a count for the employees array, using:

```
$employee_count = count($employees);
```

Then start an outer loop using $y as the incrementing number,

```
for ($y=0; $y<$employee_count; $y++) {
}
```

starting at zero, which will return the employee array with index 0, up to LESS THAN the value of `$employee_count`. The value of `$employee_count` is 4, so it will stop at 3, giving us our three employee arrays.

Then print out a title for each employee:

```
for ($y=0; $y<$employee_count; $y++) {
    echo "<h3>Details of Employee " . ($y+1) . "</h3>";
}
```

This will return `$y`, the count variable, but with 1 added to it so it matches our employee numbers.

Then we do the inner loop, which is the same as before:

```
for ($y=0; $y<$employee_count; $y++) {
    echo "<h3>Details of Employee " . ($y+1) . " </h3>";
        for ($x=0; $x < $count; $x++) {
    }
}
```

But this time we need to reference two index numbers, the first identifying first the zero[th], then the first, then the second employee record and within that looping through each data for each employee, in the same way as we did earlier manually:

```
for ($y=0; $y<$employee_count; $y++) {
    echo "<h3>Details of Employee " . ($y+1) . " </h3>";
    for ($x=0; $x < $count; $x++) {
        echo $employees[$y][$x] . '<br>';
    }
}
```

Close off both sets of braces, save and refresh, and we have a list of all the employees and their details with code which does not need to be changed however big the data record becomes.

What we have lost in using the loop to return the data is the ability to name each array element as we did the first time, where we manually printed out “First name”, “Surname” *etc.* using concatenated echo commands.

To do this using a loop we have to put those names inside the array itself. These are called keys, so the array will consist of a series of key/value pairs. This is then called an associative array,

We do this using the syntax 'key_name' => value', separated by commas in the array.

Save the file with a new name, ‘associative-array.php’, and change the three lines at the top of the file where we create the data for each of the arrays so that each line consists of a series of key/value pairs separated by commas using Firstname, Lastname, Payroll_ID and Salary as the names of the keys:

```
$employee1 = array('Firstname' => "Michael",'Lastname' =>
    "Thompson",'Payroll ID' => "51687",'Salary' => "26000");

$employee2 = array('Firstname' => "James",'Lastname' =>
    "Rasmussen",'Payroll ID' => "19875",'Salary' => "15000");

$employee3 = array('Firstname' => "Henry",'Lastname' =>
    "Foggett",'Payroll ID' => "19810",'Salary' => "150000");
```

Now we can use a different type of loop, the `foreach` loop, to return both the key names and their associated values. To see this in action in a single loop first, type in:

```
foreach ($employee1 as $key => $value){
}
```

Instead of looping through the numerical indexes this will loop through the keys and return both the keys and values.

Inside the braces, type in another echo command, concatenating `$key` and `$value` with a colon in between to make a list:

```
foreach ($employee1 as $key => $value){
    echo $key .": " . $value . "<br>";
}
```

This will print out all Employee1's details. The beauty of this over the manual method is that no matter how many key/value pairs the array may contain, the whole array will always be printed out.

To make this work as before for all the employees, we embed the `foreach` loop inside a `for` loop, exactly as before:

```
for ($x=0; $x<$employee_count; $x++) {
    foreach ($employees[$x] as $key => $value) {
        echo $key . ": " . $value . '<br />';
    }
}
```

Finally, we'll tidy up the output of both files and get some practice at the same time in combining HTML and PHP code in concatenated strings.

Let's indent the employee records so that they form visual sections under each of the headings, using a margin-left setting of 40px.

If you want, you can stop the video here and try to do this, assigning margin-left to the echo statements which print out the employee's details, using `<span style = 'margin-left: 40px;'>` and `</span>` at the end of the line.

Remember what you learnt about the dot concatenation operator – you will need that – and using single quotes to enclose double or double to enclose single.

The answers are

```
echo '<span style="margin-left: 40px;">' . $key . ": " . $value . '<br />';
```

in 'associative_arrays.php', and

```
echo '<span style="margin-left: 40px;">' . $employees<$y><$x> . '</span><br>';
```

in 'arrays.php'.

Lesson 37: Passing variables in the URL

We have learnt how to assign values to variables, concatenate strings, and deal with quote marks as part of variable values.

So far we have simply hard coded the values of variables into our scripts. We need a way of gathering input from the user and pass them to the PHP server through the browser for processing, and then do something with them; for example, perform a search or, in the case of the Favourite Movies project, see the data related to a particular movie-goer.

To do these things we have to gather data from one place, typically the end user, and pass it to another – usually another webpage - for processing.

There are two ways of passing variables from one page to another in PHP. In one way the variable name and value appear in the URL and in the other way they do not.

Navigate to a website like Amazon or Google – any website with a search facility – and type something into the search box and press return. If you look at the URL, the website address in the address bar, you will see that somewhere in it your keywords appear. The variable is being passed in the URL.

To pass a variable in the URL, we append to the end of the address a question mark followed by the variable name, equals, and its value. We do not use a dollar sign or quotation marks.

Then that variable has to be gathered by the page which receives it. This is done with the PHP superglobal $_GET using the syntax:

```
$_GET['name of variable'];
```

And then we can assign the value of that to any variable name, in the usual way.

Ordinary variables we can name ourselves, give them any name we want with some restrictions, and assign any value we want to them. Superglobals like this are predefined in the system. The sole job which `$_GET` does is to grab from the URL the value of the variable which matches this name, and the name of the variable to look for must be inside square brackets. Use single quote marks around the variable name.

Notice that the variable name has no $ sign in the GET statement, but in assigning this value to a variable of our choice we do need a $ sign because this is an ordinary variable assignment, just like we've been doing before. The variable name often matches the name of the variable sent in the URL, for convenience, but it doesn't have to. Any name can be assigned here.

Now we'll put this into practice.

In the ‘php-demos’ folder, create a new php file and name it ‘get.php’. The completed file is in the Working Files if you prefer to use that.

We will use the variable name $content, so inside PHP tags type in:

```
$user_id = $_GET[user_id'];
```

and an echo statement:

```
echo "<h1>The user ID you have chosen is $user_id</h1>";
```

Now open up ‘get.php’ in your browser and add `?user_id=12` to the URL. The script will gather this and print it out in the browser window. Try changing the variable to something else and this too will be echoed out.

Now we need to see how to gather this data from the user.

Create a new PHP file in the ‘php-demos’ folder and name it ‘send.php’. We will use this to send a link with a value for the variable $content to the file ‘get.php’.

Type in `<a href="get.php?user_id=1">User ID 1</a>` and refresh the page in your browser. The receiving page receives the variable value from the server and outputs it to the screen.

The GET method of sending variables is used when we want to generate a URL we can bookmark – such as search results or particular pages. It's not suitable for sending data which is sensitive such as usernames or passwords because the information is publicly available in the URL.

In the next video we'll learn an alternative way of sending variable values, in a way which does not involve them appearing in the URL.

Lesson 38: Passing variables without their appearing in the URL

In the previous video we gathered user input and passed this as a variable in the URL to a receiving page. Now we want to learn how to pass variables without them appearing in the URL.

This is done by using the PHP superglobal `$_POST` instead of `$_GET`. We start by gathering the data from the user, usually in an HTML form, and we specify in the opening `<form>` tag both the action, which is the page the browser will be taken to when the submit button is clicked, and the method, which is either "get", as before, or "post" if we do not want the data submitted to appear in the URL.

Then in the receiving page, we use `$_POST` instead of `$_GET` and assign the value of `$_POST` to a variable in the same way as before.

Let's put this into practice.

In the 'php-demos' folder, create a new text document and name it 'form.php'.

Type in a pair of opening and closing HTML `<form></form>` tags. Between the `<form>` tags, type in `<input type="text" name="content">` :

```
<form>
    <input type="text" name="content">
</form>
```

and load the page in your browser. You will see a familiar input box like we see on websites like Google which take and process user input.

`Input` tells the browser that this will accept input from the user. `Type="text"` tells it that it is a text input box for text to be typed into. The name is important as this must correspond to the variable name used on the accepting page. No `$` sign is used here.

Then we ask for more input from the user, but this time we're making a Submit button to click on, with the HTML5 button tag, `type=submit`:

```
<form>
    <input type="text" name="content">
    <button type="submit">Submit</button>
</form>
```

But nothing happens when we enter a value and click.

We need to set the action of the form with `action=` and the name of the page to be taken to when the button is clicked. Instead of sending the data to a new page, we can send the page back to itself with the data, so put 'form.php' as the action of the form:

```
<form action="form.php">
    <input type="text" name="content">
    <button type="submit">Submit</button>
</form>
```

Finally we need to specify the method to be used to send the data - we're using `post`:

```
<form action="form.php" method="post">
    <input type="text" name="content">
    <button type="submit">Submit</button>
</form>
```

We are sending the data back to this same page for processing this time, so we need the receiving part of the script in it as well. Above the form, between PHP tags, enter `$_POST['content']` as 'content' is the name of the input in the form, and assign its value to the variable `$content`:

```
$content = $_POST['content'];
```

Then below the form, end the PHP tags, put in a couple of line breaks and then type:

```
<p>The content you entered was <?php echo $content; ?></p>
```

This time we have done what we learnt about earlier and embedded PHP in HTML, rather than the other way round.

Load up 'form.php' in your browser and enter something in the input box and click 'Submit' and you should see the message confirming that the data has been successfully processed.

This method of passing variables is more suitable for sending variables which we wouldn't want to be publicly displayed in the URL. It is not suitable for pages which we want to have a URL we can bookmark.

Lesson 39: Guarding against missing variables

It's possible when using GET and POST to pass variables from one page to another, for a variable to be missing entirely. In the case of GET, where the variable is part of the URL, a user could change or delete that part of the URL. In the case of POST, which is used most for passing data from a form, an input box in the form could be left empty.

As our scripts are at the moment, if the variable is not set, in the URL for get and in the form for post, the script will fail with a system error. Let's demonstrate this with the GET version first.

Open up 'send.html' again in your browser and click on the link. As expected you will be taken to 'get.php', which will echo out the value of the variable set in the URL – in this case '10'.

If you change the value in the URL to anything else, the new value will be echoed out with no problem because we have done nothing to filter out non-numerical data. This even works if you delete the value and leave the URL ending with an equals sign – because the variable is set – although its value is empty.

But if you delete the `?user_id=` part of the URL or change the variable name, you will get a system error, undefined index.

In 'form.php' the first time the script runs we get the same undefined index error because the value of the variable content has not been set.

To avoid this, and to make the script able to fail gracefully or take some sort of appropriate action, we need to prevent the script from trying to access a non-existent variable.

We do this using `isset`, which checks whether or not the variable is set in the URL.

In 'get.php', wrap the variable assignment and the echo statement inside an IF statement –

```
if () {
}
```

and then inside the parentheses for the IF statement put isset and its parentheses:

```
if (isset()) {
}
```

and then inside isset's parentheses, S_GET, its square brackets and single quotes ['] :

```
if (isset($_GET[''])) {
}
```

and inside the quote marks the name of the variable as it appears in the URL, `user_id` without a $ sign:

```
if (isset($_GET['user_id'])) {
}
```

This tests whether or not $_GET has found `user_id` in the URL.

Then inside the braces put the code we had before:

```
if (isset($_GET['user_id'])) {
    $user_id = $_GET['user_id'];
}
```

to assign the value found with that name in the URL to the PHP variable `$user_id`.

We can now echo the message with the value of the variable as before – but only if the variable has been set.

Then we can catch the eventuality of a missing user_id in the URL by using an ELSE clause, which will execute if `$_GET['user_id']` is NOT set and echo out a message without trying to use `$user_id`:

```
if (isset($_GET['user_id'])) {
    $user_id = $_GET['user_id'];
} else {
    echo "User ID not set";
}
```

This prevents system errors when the variable we are looking for in the URL is not set, so whenever we use `$_GET` we will wrap the variable assignment inside an `if isset()` statement in this way.

Now you do the same in ‘form.php’, without copying from ‘get.php’, so that the POST version is similarly protected against a missing variable.

Lesson 40: Functions and variable scope

We saw earlier in the chapter how variables set in PHP include files are available in the including file. Sometimes this is very convenient, but at other times it is not what we want. If we're not careful it can result in chaos and confusion when you find that a variable has an unexpected value set somewhere and you have to track down where in a mass of code. Very often, we want to avoid having variables floating around freely – we want to constrain them so that they only hold their value for a specific section of code. To do this we use functions.

PHP has hundreds of built-in functions, but we can also write our own custom functions, which is where real flexibility and power can be achieved.

Make a new file in 'php-demos' and save it as 'functions.php'.

We create a function with the keyword `function`, then the name we are going to give the function, then a set of parentheses, and finally a set of braces enclosing the code which makes up the function:

```
function functionName() {
    // function code
}
```

Let's make a function called `showFruit`. Inside it, we'll assign a value to the variable `$fruit1` and echo out that value:

```
function showFruit() {
    $fruit = "apples";
}
```

Run the file in your browser and you get... nothing.

Functions don't run until they are called. We call the function by entering its name and parentheses:

```
showFruit();
```

Unlike in include files, where we have to declare a variable before we can use it, we can call a function before or after the function itself – it makes no difference.

Now refresh the page and "apples" should appear in the browser window.

Variables set inside functions are only useable inside the function in which they are set. Move the echo statement out of the function to the main body of the file, save and refresh, and you'll get an undefined variable error because the variable inside the function is unknown outside the function.

Move the echo statement back inside the function.

More often than not, this is how we want our variables to behave.

But it also applies the other way round – variables set outside a function are unknown inside the function. Let's try this.

Set a variable, `$fruit2`, before the start of the function:

```
$fruit2 = "peaches";
function showFruit() {
    $fruit = "apples";
}
```

and then try to echo the value of `$fruit2` from within the function:

```
$fruit2 = "peaches";

function showFruit() {
    $fruit = "apples";
    echo $fruit2;
}
```

Again, you'll get an undefined variable error.

Very often we do want variables set outside functions to be available inside them. We do this using the keyword `global` and then the variable name inside the function:

```
$fruit2 = "peaches";

function showFruit() {
    global $fruit2;
    $fruit = "apples";
    echo $fruit2;
}
```

Save and refresh and now the variable `$fruit2` is available inside the function.

If you search on the web for global variables in PHP you will find a lot about them being bad practice. I'll come back to this in more detail in Chapter 9 and again at the end of the course when we consider the way forward, into programming for larger projects, but the short answer at this stage is that for a small project of this sort, global variables are a perfectly acceptable technique to use – their main downside is that they make the code harder to manage. For our purposes, when you are just starting out on PHP in a small project, we can safely ignore this issue.

That's a short and simple coverage of variable scope in functions. In the next lesson we will look at using parameters with custom functions.

Lesson 41: Passing data to functions with parameters

Other than the global variable we set up in the previous lesson, our example function takes no information from the script outside it. Often we want to pass information of some sort from the main script into the function. We do this using parameters, which go inside the function's parentheses. In this lesson I'll exemplify the use of numerical and text parameters to change the information passed into a function and to format the output from the function.

Make a new file in 'php-demos' called 'functions2.php' and give it its PHP tags. We'll write a function which returns the area of a rectangle from the length of its two sides.

Start with the keyword `function` and give it the name `area`:

```
function area() {
}
```

And this time inside the parentheses we'll enter the two variables it needs to perform the calculation, which we'll call $x and $y, separated by a comma:

```
function area($x, $y) {
}
```

Then we'll perform the calculation, `$x` times `$y`, assign the result to the variable `$area`, and echo the value of this:

```
function area($x, $y) {
    $area = ($x * $y);
    echo $area;
}
```

Finally, call the function using 'area' and put inside the parentheses the values you want to perform the calculation on, e.g.:

```
area(4,5);
```

Save and refresh and you should get the correct result in the browser.

Let's make a couple of refinements to this.

Usually we don't want the function to echo results out immediately – we would normally prefer it to return the results to the main script and then use the script to perform the action when we want it done.

Change `echo $area` to `return($area)`:

```
function area($x, $y) {
    $area = ($x * $y);
    return $area;
}
```

`Return` is the way we normally get data out of a function. It returns the value of whatever the function has done to the main script and exits the function at that point. Save and refresh ... and nothing appears. We have returned the value of $area to the main script but have done nothing to print it out.

We can do this with:

```
echo area(3,3);
```

... that works, or in two stages:

```
$area = area(3,3);
echo $area;
```

Both give the same result.

You can verify that `return` in the function stops the function at that point by putting in an `echo` statement after it:

```
function area($x, $y) {
    $area = ($x * $y);
    return $area;
    echo $area;
}
```

The echo statement will not run.

As well as numerical data in the parameters, we can put other types of data. We could use a text parameter to format the output. Add a comma and then `$format` as a third parameter:

```
function area($x, $y, $format) {
    $area = ($x * $y);
    return $area;
}
```

We'll output a text/variable string with opening and closing `<h1>` tags.

To make use of this in the function we need two stages – to set up a string variable with the HTML and area variable and then return this new output variable.

So first set $format as the value of a new variable to output, which we will call `$output`, and for the moment remove the line `return $area`:

```
function area($x, $y, $format) {
    $area = ($x * $y);
    $output = "$format";
}
```

Remember we must use double quotes or variable names will be returned, not their values. Add the opening and closing angle brackets to make $format an HTML tag:

```
function area($x, $y, $format) {
    $area = ($x * $y);
    $output = "<$format>";
}
```

Add the variable `$area` to the output string:

```
function area($x, $y, $format) {
    $area = ($x * $y);
    $output = "<$format>$area";
}
```

And then add the closing angle bracket and forward slash for the closing format tag:

```
function area($x, $y, $format) {
    $area = ($x * $y);
    $output = "<$format>$area</$format>";
}
```

Then we return $output to the main script:

```
function area($x, $y, $format) {
    $area = ($x * $y);
      $output = "<$format>$area</$format>";
    return $output;
}
```

Finally, in the function call put in single quotes ‘h1’ as the value of `$format`, e.g.:

```
area(3,4,'h1');
```

Save and refresh, and you should see the value of `$area` formatted as a level 1 heading.

Change the value of `$format` in the function call and you’ll get different formatting in the HTML.

We’ll make use of this method of returning formatted data from functions later on in the project.

Chapter 7 - MySQL in phpMyAdmin

Lesson 42: What is a database?

This chapter explains what a database is and goes through the MySQL commands needed for this project.

If you're already familiar with the basics of using MySQL in phpMyAdmin and know what those terms mean, you can import the SQL file 'initialise_movies_database.sql', which you will find in the 'INITIALISE DATABASE' folder in the Working Files.

This sets up the favourite movies database as it is at the end of this chapter, allowing you to move straight on to Chapter 8.

If you haven't used phpMyAdmin or MySQL before then you should complete this chapter.

A database is an organised collection of data, records of items, stored in one or more tables.

Each record is a complete set of information, and each record is made up of individual items of information, called fields.

So to take a driving licence database as an example, each record would contain all the details relating to one person – that person's first name, last name, date of birth, driver license number, any motoring transgressions, etc.

Looking at our favourite movies project, each record in the movies table will consist of a single movie's title and description, the column names of the table being the names of these individual fields.

We will also want to guard against the possibility of two different movies the same title and description, perhaps because they are different releases of the same movie, and we do this by including a unique identifying number so that whatever else is the same, no two records can ever be confused. This identifying number, or index, will be set up in such a way that is never repeated in the table.

We'll also have a table called 'movie-goers', where the movie-goers' details will be stored. This is very similar, consisting again of an ID number which is unique to each individual, and the individual's first name and last name.

Then we need to have a way to link the movies table and the movie-goer's table, so that we can store certain movies as favourites for each user and can change these. A crude way to store favourites would be to store a list of favourite movies and the names of the people that like them in a table called favourites.

So for example if *The Twilight Sage* was a favourite of Steven Jones, Sweeney Todd, and The Queen of Sheba, the favourites table would have the movie listed three times, once for each person who likes it. The immediately obvious problem with this is that we now have three records for the same movie. This is a bad approach as each of these movies is just one entity and should be stored as just one record in the database, not repeated each time someone adds it as a favourite. If there were a lot of users and movies, this would rapidly lead to an enormous database, with a huge amount of duplicated data.

As before with coding, if we find we are duplicating identical information, we are doing something wrong and need to think again.

We do this by creating a linking table, consisting just of the ID numbers from each of the two tables to be linked. I have omitted the description column from the slides to save space here.

So in this example, we want to store movie goers who like particular movies.

We take the movie_ids and user_ids which link movies and movie goers and store these in a new table, called favourites.

So we have the movie_id 1, linked to user_ids 1, 3, and 5, the IDs of the users who have that movie as a favourite.

Then we link these user_ids to their details in the movie_goers table.

This avoids all duplication of data.

A database which consists of a number of tables linked in this way is called a ‘relational database’.

This tutorial will demonstrate how to set up a relational database for our favourite movies project and deploy it over the web.

In this chapter, we will learn how to work with databases using MySQL.

SQL stands for Structured Query Language; the "My" part of MySQL is apparently the name of the daughter of the founder of MySQL, Michael Widenius.

MySQL is the language used to control databases, to create and delete them, and to add, modify and delete data as well as to select data according to criteria we set.

The first, and simplest, way to work with databases is to use the phpMyAdmin interface which we looked at briefly in Chapter 3.

This graphical user interface or GUI, often pronounced ‘gooey’, allows us to do everything there is to do with a database simply by clicking on buttons and we will often use this to perform maintenance tasks on our database as it is the easiest way to do jobs of this sort.

The downside is that we have to click buttons every time we want to do anything; nothing can be automated, so its main use is for one-off jobs such as maintenance tasks for which it would be too much trouble to write a program.

Working with the GUI is easy and doesn't really need much instruction – by the end of this chapter you will be able to do that.

The second way is to use the MySQL command line. This involves typing commands into the SQL box in PHPMyAdmin. It performs the same tasks clicking on buttons, and is the easiest way for us to learn how to write MySQL queries.

The third way and final way is to embed MySQL in PHP scripts.

This allows full automation of all the functions of MySQL deployed over the web, so that multiple users can query the records of the database, as they perform searches.

In the next video, we will begin by using the MySQL command line in PHPMyAdmin to work directly with our MySQL databases and learn some key MySQL queries.

Lesson 43: Create a database, add tables

This video is an introduction to working directly with a database by using the SQL command line in phpMyAdmin, typing MySQL queries in. In your browser, navigate to http://localhost/phpmyadmin.

First, notice that in the left-hand panel, there are a number of databases already listed, named ‘information_schema’, ‘mysql’, ‘performance_schema’, ‘phpmyadmin’, and ‘webauth’. These are all system databases and should be left alone, so don’t click on any of these.

We can do everything there is to do with our databases in phpMyAdmin by clicking on buttons as I said earlier and we'll do some of the tasks this way and others using the command line, so that you learn to write MySQL queries.

So far we have no favourite movies database, so we'll begin by creating a new one.

Click on SQL at the top of the page to bring up the SQL query window.

We'll make a new database called ‘movies’. The command for this couldn't be easier: type in `CREATE DATABASE movies` and click 'Go'.

The new ‘movies’ database will appear along with the system databases in the left-hand panel.

Click on its name and you will get the message ‘No tables found in database’, because so far the database is empty. The first time we create a table we'll take the easy option and use the phpMyAdmin user interface.

In the ‘name’ box, type in the table name ‘movies’ and in ‘number of columns’, put ‘3’.

You'll be confronted with a grid into which you must enter the names of the fields for the table.

For the first field, enter the name ‘movie_id’. Every table will have an ID field, containing the unique number I told you of earlier, which serves, if everything else is identical, to distinguish one record from another.

Next we need to tell the database what TYPE of data this field will contain. Clicking on the question mark next to ‘Type’ will take you to the MySQL reference manual for data types where you can find more information on data types. For this field, which will be the primary index for this table, we will choose INT, for INTEGER.

In the ‘next’ box, give it a length of 11.

Scroll over to the right until you get to INDEX and choose PRIMARY. This will make the database use this as its main index and will speed up queries.

If you move your mouse over the abbreviation ‘A_I’ above the next box, you will see that it stands for ‘auto-increment’. This means that each time a new record is added, the ID number assigned to it will automatically increase by one. This has the effect that no index number can will appear more than once, ensuring that even if two titles are the same, no two records can ever be identical. Check this box.

Then go to the next field and assign it a name of ‘title’ and the type VARCHAR.

‘Varchar’ stands for ‘variable character’. This data type is for variable-length character strings, which can be a mixture of letters and numbers. This is the most common used data type for strings. Varchar has to be assigned a maximum string length. Let’s assume none of our titles will have a first name longer than 256 characters and assign it a length of 255 (numbering starts at zero).

Leave the rest of the options for 'title' empty and repeat the process for the third field, 'description'.

Click 'Save'. You will get a confirmation message that the table has been created and will be taken to the 'structure' view of the new table, showing the field names and types that we've just created.

Now follow the same procedure to create the two other tables we need, 'movie_goers' and 'favourites'.

'Movie_goers' consists of three fields, 'user_id', which is an auto-incrementing primary index as before,

and two VARCHAR fields, 'firstname' and 'lastname'.

Give them both a maximum length of 40 – that should be more than enough for any name.

'Favourites' consists of two fields, 'user_id' and 'movie_id', both of which are integers which cannot be empty. For this we use INT (integer) as the type and don't click on the NULL checkbox, because we do not want to allow any empty values. We need to allow duplicates for both, so don't click on auto-increment this time.

Now we have our three tables set up but they are all empty – there are no records in them.

In the next video we will learn how to add data to tables.

Lesson 44: Insert data

Now we'll enter data into the movies table, first using the GUI and then the command line.

Click on the 'movies' database and then on the 'movies' table to enter it.

Click on the 'Insert' button at the top of the screen. The input boxes allow you to enter data.

Leave the value input for 'movie_id' empty because as this is an Auto-Increment field it will automatically fill itself as records are added.

In the 'value' input for title, enter *The Twilight Sage* and as the description enter "A long, straggly tale about a man with a fake beard. Avoid", or whatever else you want.

Click 'Go' and you will get a confirmation message.

You will also see that the SQL query used to display these results is displayed, and we can use the SQL queries displayed by phpMyAdmin to learn command line syntax. We'll adapt this query to do another INSERT, this time using the SQL command line, so copy it while it is on the screen.

Now click on 'browse' and you will see the contents of the table.

You will see that ID has indeed been automatically filled in, with *The Twilight Sage* assigned the movie_id of 1 and *Great Cheeses of our Time* the movie_id number 2.

Go to the SQL command box and paste the command you just copied in there again, and we'll have a look at it.

The query reads:

```
INSERT INTO `movies`.`movies` (`movie_id` ,`title `, `description` )
    VALUES (NULL, 'Great Cheeses of our Time', 'A study of some memorable
cheeses.');
```

This is a relatively complicated MySQL query, as it has to tell the system not just WHAT to insert, but which value goes into which field.

So this is spelled out in two clauses, first specifying in order the fields where the data is to go and then, in the same order, the values to insert into each field.

It starts with `INSERT INTO` and then the database and table name, inside backticks and separated by a dot: `` `movies`.`movies` ``. This dot notation identifies first the database and then the name of a table inside that database. The backtick character, by the way, is at the top left of a UK or US keyboard, next to the number 1 key.

Inside round parentheses is a comma-separated list of field names, again each inside backticks, identifying in order which fields we are inserting data into:

```
(`movie_id` ,`title `, `description` )
```

Notice that the last field name is not followed by a comma. If it is, the command will not work and an error message will be returned.

This clause is ended with closing parentheses.

Then we have to specify WHAT to insert, the `VALUES`, again inside parentheses:

```
VALUES (NULL, ' Great Cheeses of our Time', 'A study of some memorable cheeses.');
```

This is a matching comma-separated list of values, this time inside ordinary single quotes, not backticks, of the values to insert.

The order of these must match the order of the fields in the INSERT INTO clause.

The first field, 'movie_id', is given a value of NULL, because thanks to its Auto-Increment setting it will automatically insert a value one greater than the ID of the previously inserted record, so there's no need to enter a value here.

The second item in the comma-separated list, `'Great Cheeses of our Time'`, corresponds to the second field in the INSERT INTO clause of the command – the title field.

And the third item in the list corresponds to the third field name, the description.

As before, the last of the values must not be followed by a comma or the query will fail.

Now try the INSERT INTO query yourself, by inserting a third title and description into the 'movies' table, which can be whatever you want. I will put *Life of Pie* as the title and "A silly pun" as the description.

Pause the video here and see if you can do this before I tell you.

Did you get it right? All you have to change is the title and description, leave movie_id as NULL, and click Go:

```
INSERT INTO `movies`.`movies` (`movie_id` ,`title `, `description` )
    VALUES (NULL, 'Life of Pie', 'A silly pun.');
```

Then click on 'Browse' and you will see that this third title appears in the table and the movie_id of 2 has been assigned to his record.

Let's do one more INSERT INTO query, this time inserting two records at once. We do this by having TWO lists of values after the VALUES clause, separated by a comma, with the values themselves in parentheses:

```
INSERT INTO `movies` (`movie_id` ,`title`, `description` )
    VALUES (NULL, 'Sightseers', 'A party of tourists see London from an open top bus.'),
    NULL, '13 Types of Cheese', 'Ten truly great cheeses. And three awful ones.');
```

Remember to leave the values for ID as NULL.

Click on 'Browse' and verify that the data has indeed been inserted.

We have created a database, added a table, and inserted data using the command line. That's enough for this video, but there are several general points to make at this stage.

The first concerns the backticks used to enclose database names, table names and fieldnames. In the example used here these backticks could be omitted – we can insert another data item without them to demonstrate that this is the case:

```
INSERT INTO movies ( movie_id ,title, description )
    VALUES (NULL, 'End of the Match', 'A very short film indeed about the
electricity supply in 1970s Britain');
```

... the purpose of the backticks is to ensure that the query will still work if we happen to use a MySQL command name as the name of a database, table, or field. For example, as long as I enclose it in backticks, I could name a table INSERT, although I certainly wouldn't recommend doing so. If we don't use backticks we have to take care not to use words with special functions in MySQL for database, table, or fieldnames.

Second, the carriage returns and white space in MySQL queries are purely for our own convenience, to make the commands easier to read. We can change this to whatever style we find easiest to read (or if we are working for someone else, whatever our boss specifies) as they have no effect on the workings of the command.

The mixture of uppercase and lowercase is also mainly for our convenience. It is normal programming style to put the MySQL commands, such as INSERT INTO, VALUES, etc in uppercase to make them stand out and make the code easier to read.

In fact they're not case sensitive and will still work if they are in lowercase or even a mixture, but it is considered bad style to have them in anything other than uppercase, so stick with uppercase.

Database, table names and field names should all be treated as case sensitive. Although in Windows they will work if the case is wrong, if we ever upload this project onto a webserver, it will be running on a Unix-style system. On Linux these are case sensitive, so the case must be correct or it will fail to work and will cause problems to sort out.

In the next video, we will get some more data into our table and learn how to perform searches on it.

Lesson 45: Import data

Before we do any searches on our database, let's populate it with some more data.

I'm sure you dislike unnecessary typing as much as I do, and working in phpMyAdmin is a slow process, so I have provided some more data which you can import from an SQL file.

Click on the 'Import' button at the top of the page. Click 'Browse' and select 'initialise_movies_database.sql' in the 'INITIALISE DATABASE folder' in the Working Files.

Leave the options unchanged, just checking that 'Format' is set to 'SQL', and click 'Go'.

Click on the database and then on the 'movies' tablename and you should see that some spoof movie titles and descriptions have been imported successfully. Click on the 'favourites' and 'movie_goers' tables and you will see that these too have been populated with data.

Now open 'initialise_movies_database.sql' in Komodo Edit and we'll have a look at its contents. It consists of a series of SQL queries, just as we have been using, each one terminated with a semi-colon.

We could use this in two ways – by importing the file as we just did or we could copy the queries and paste them into the SQL command line box.

Looking at the first line, 'drop' is the mysql command used to delete a whole database so `DROP DATABASE IF EXISTS movies` deletes the whole database if it exists, so we can start again from scratch.

Then if the database named 'movies' does not exist, it is created, using `CREATE DATABASE IF NOT EXISTS movies`. So now we've started again with a database named 'movies' with no tables in it.

`USE movies` tells gets us inside that database – the equivalent of clicking on the database name in phpMyAdmin so we don't have to specify the database name from then on.

Following that, we redo what we did before in the GUI, creating a table named 'movies', with `CREATE table movies`.

Here's the whole of that query, and as it is complicated I'll go through it in detail:

```
CREATE TABLE movies (
  movie_id INT(11)
  NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(movie_id),
  title VARCHAR(255),
  description VARCHAR(255)
 ) ENGINE=InnoDB
  COLLATE utf8_general_ci;
```

In parentheses we first specify a comma-separated list of fields to create, with a field named 'movie_id', which is of datatype INTEGER, and has a length of 11: `movie_id INT(11)`.

Empty or null values are not allowed: `NOT NULL`, and it is set to auto-increment: `AUTO_INCREMENT,`. Notice the comma. Then we set this as the primary key for the table: `PRIMARY KEY(movie_id),` then the next field, 'title', which is of type VARCHAR and has a length of 255 characters: `title VARCHAR(255),`.

Then the same for the ‘description’ field: `description VARCHAR(255)`

Close off the parentheses: `)`

Then we need to specify the database storage engine to use. We’ll use InnoDB: `ENGINE=InnoDB`.

And for the collation for the character encoding we’ll use Unicode, UTF-8: `COLLATE utf8_general_ci`.

After that we have an INSERT INTO command just like the one we wrote in the previous lesson, but inserting a larger number of records.

Just as before, we specify the fieldnames in order and then we have a comma-separated list of sets of values, each set corresponding to a record, each record enclosed in parentheses and separated from the previous one by a comma.

After that, the process is repeated to create the tables ‘movie_goers’ and ‘favourites’. In the ‘favourites’ table we have a list of IDs from the two other tables which will show which users have which movies as favourites.

Now close Komodo Edit and go back to phpMyAdmin and in the next lesson we’ll learn how to select particular records according to criteria we set.

Lesson 46: Select records

Now we'll see how to select records from a database. In phpMyAdmin, click on the 'movies' table name. The data, 15 movie titles, descriptions and IDs, is displayed on the screen, along with the query used to display it:

```
SELECT * FROM `movies` LIMIT 0 , 30
```

The asterisk has a special meaning in MySQL – 'all fields'. So, this query means "Select all fields FROM the table called 'movies', with a limit on how much data is returned, from the first record, number zero, to the record numbered 30. As we have less than 30 records this limit has no effect.

That's nice, but our records are displayed in 'movie_id' order, which is not of much relevance to us. It would be even nicer if we could choose to sort them into alphabetical order by title. Copy the query, leaving off the LIMIT clause and add ORDER BY title to the end of the query:

```
SELECT * FROM `movies` ORDER BY `title`
```

Press 'Go' ... and the results will be ordered alphabetically by last name.

Although we would probably usually want results ordered from A to Z rather than Z to A, there are times when we want to sort in reverse order – for instance if the data was numeric we might want the largest number first in a list. It's easy to do this. Just add DESC to the end of the ORDER BY clause:

```
SELECT * FROM `movies` ORDER BY title DESC
```

We may not always want to display all the fields in a table. For instance, the ID number is interesting to the database system but not of much use to us as humans. Change the SQL query to `SELECT `title` FROM `movies`` and only the title field will be displayed.

We can also include more than one field in our select query by separating them with a comma:

```
SELECT `firstname`, `lastname` FROM `movie_goers`
```

Remember: make sure there is no comma after the last fieldname in the list.

So far we have displayed all the records in a table, which isn't normally what we want to do, so now we'll modify the select query to search for given patterns.

Do this by entering

```
SELECT `title` FROM `movies` WHERE title = 'Life of Pie'
```

and you will see that only that one record is returned.

That matches the whole of the title field.

For our project we won't need to, but if we wanted to search for a partial match we do so using `LIKE` instead of `=` and using `%` marks as a wildcard to match any number of any character:

```
SELECT * FROM `movies` WHERE title LIKE 'Life%'
```

with the wildcard at the end of the search string, it will match any title beginning with 'Life'.

With wildcards at the beginning and end of the search string, we can return any records containing the pattern 'cheese' anywhere in the title:

```
SELECT * FROM `movies` WHERE title LIKE '%cheese%'
```

You can see straight away that MySQL is rather user friendly in its terminology – it's not a cryptic computing language at all, but rather like ordinary English, with these keywords we have been learning, CREATE, INSERT INTO, SELECT FROM, WHERE, and LIKE so it's quite an easy language to learn.

It is important, however, to enter the various clauses in the right order – they must go in the order shown in the examples.

That's the basics of the SELECT query. In the next lesson we'll learn how to update an already existing record in the database.

Lesson 47: Update existing database records

Now we'll see how to alter, or update, an existing record in the database. We'll do this first using the GUI and notice the SQL query generated.

In phpMyAdmin, click to get into the 'movies' database and then into the 'movies' table.

Click on one of the movie titles, change its value and when we leave the input box, the value will be instantly updated.

Incidentally, we'll be coding this exact same interface where you make an alteration directly in an input box and save the new value just by leaving the input box for the favourite movies project later.

Look at the SQL query generated:

```
UPDATE `movies`.`movies` SET `title` = 'The Twilight Sage'
    WHERE `movies`.`movie_id` = 1;
```

It consists of UPDATE – then in backticks the database name - dot - tablename – SET – then the fieldname EQUALS - the new value - WHERE – and the criteria to match, using the value of 'movie_id', the primary index.

You can copy that query ... and edit it to create another one of your own:

```
UPDATE movies SET title = 'The Right Trousers'
    WHERE movie_id = 15;
```

As you know, the database name and backticks are not needed because we are not using MySQL reserved words for the database, table, or fieldnames; I have omitted them here.

Check now that the table has updated as expected.

That, very simply, is how existing records are updated in MySQL. In the next lesson we'll learn how to use more advanced SELECT syntax to perform the join we need to link favourite movies and users by their ID numbers.

Lesson 48: Select favourites

Now we've reached a central feature of the project – how to display a list of favourite movies for each particular user.

There are various ways to do this. I am going to demonstrate one method, using the `IN` command in MySQL. `IN` is used to search for a list of matching records, rather than matching a search pattern as we did before.

Using `IN`, we can select a list of values which occur in a field.

We do this by specifying after the `IN` command a comma separated list of values to search for, inside parentheses:

```
SELECT firstname, lastname FROM movie_goers
    WHERE lastname IN ('Seagul', 'Toad', 'Douglas');
```

And you will see that records which match any of the items in the list are returned.

Or we can exclude those in a list, using `NOT IN`:

```
SELECT firstname, lastname FROM movie_goers
    WHERE firstname NOT IN ('Wallace', 'Gromit');
```

This returns every moviegoer in the table except Wallace and Gromit.

How can we use this to find which movie goers like a particular movie?

For each person, identified by their unique user_id, we can get a list of that person's favourite movies from the 'favourites' table, using `SELECT movie_id FROM favourites WHERE user_id =`

I'll use 9 as the example as I know that movie-goer has a lot of favourites:

```
SELECT movie_id FROM favourites WHERE user_id = 9;
```

So if I specify that I want to search for movie_goer 9's favourite movies, I will get the list of that person's favourite movies, by their movie ID numbers.

But how do we get the movie titles from that, as they're in another table?

We can do this by using the list of movie_ids returned from the query we've just written to search in the movies table for the titles which match those movie ids.

So we use `SELECT * FROM movies` and then have a `WHERE` clause in which we'll specify our search criteria.

For the `WHERE` clause we want to use any occurrences of the movie_id we are interested in, which will be a list not a single pattern to match, so we need to use `IN`.

Inside the `IN` clause we embed the whole of the `SELECT` statement we have just written, knowing that this will return the list of movie IDs representing the favourites for the user we have specified.

The final code is:

```
SELECT * FROM movies
    WHERE movie_id IN (
        SELECT movie_id FROM favourites
            WHERE user_id = 1
        )
ORDER BY title;
```

Copy that and press ‘Go’.

And this nested SQL query returns exactly what we want.

To reverse it so that it returns only non-favourites just change `IN` to `NOT IN`.

There are other ways of doing the same job, but this is an effective and simple method for our purposes.

Lesson 49: Delete records, empty and delete tables and database

To delete data we use the command DELETE FROM and specify the table we are working in:

```
DELETE FROM movies;
```

If we run this query it will delete all the records in the 'movies' table. To delete just the records matching a particular string, we specify the criteria to match using the WHERE command just as we did to select data:

```
DELETE FROM movies WHERE lastname = 'Gaits';
```

Will delete just the record with the surname 'Gaits'.

To delete records where more than one field is involved in the selection we use the same syntax as in PHP, using a double ampersand for the AND clause:

```
DELETE FROM favourites
    WHERE user_id = 3 && movie_id = 9
```

This is the syntax we'll use later to delete a particular movie from a given movie_goer's list of favourites.

Just for completeness, we'll do the commands to empty a table of its data whilst retaining its structure – the fieldnames and their datatypes.

This is `TRUNCATE`:

```
TRUNCATE TABLE favourites;
```

Now when you click on 'Browse' you will be taken to the structure view as there are no records in the table.

Finally, to delete a table or database we use DROP:

```
DROP table movie_goers;
```

and that table will no longer be there.

You can use `DROP` on a whole database:

```
DROP DATABASE movies;
```

and the whole database has gone.

Those are all the MySQL queries we need for the project to function – we can display the results of searches, and using `INSERT INTO` and `DELETE FROM` we will be able to add and delete movies from the movie-goers' favourites lists.

So far we have done this all by hand, entering SQL queries directly into the SQL command line in phpMyAdmin. This is no use if we want to develop an interface we can use through a browser or deploy over the web.

For this we need to go a step further and embed MySQL queries in PHP scripts so that they can form part of dynamic webpages, and in the next chapter I will show you how to do this.

Before we go on we need to do a bit of housekeeping and return our database to its original state. To do this, import ‘initialise_movies_database.sql’ again from the ‘INITIALISE DATABASE’ folder in the Working Files. This will delete the database we have been working with and rebuild it with the full data set.

Chapter 8: MySQL in PHP

Lesson 50: Initialise and connect to a database with MySQLi

We have learnt the SQL queries we need to get data from a database and to add and delete data, and we have learnt the more complex query we need later to select favourites for a given user. But so far we can only do all this working directly with the database in phpMyAdmin. In this chapter we will learn how to embed those SQL queries in PHP so that we can work with a database through a browser.

In phpMyAdmin there was no need to do anything to connect to the database; we simply clicked on the database and then on the table names to gain access to them. But to access the database through a browser, we have to connect to it first.

Let's begin by importing a test database to work with, using phpMyAdmin as we did before.

Open up phpMyAdmin, click on 'Import', select 'initilalise_test_database.sql' from the folder for this lesson and click 'Go'.

The database name 'test_db' should appear on the left of the phpMyAdmin screen.

Click on the database name and you should see a table containing the fields 'id', 'name', 'password' and 'secrets'.

Each one contains very simple data, user1, user1pass, *etc.*., that we can use to demonstrate working with a database and, later in this chapter, security vulnerabilities inherent in working with MySQL in PHP.

Now we need to write the code to connect to it. As this is outside the Project, we'll do this in the folder 'php-demos'.

In 'php-demos', make a new text file and call it 'connect.php'. Open it up in Komodo Edit and enter opening and closing PHP tags.

Within the PHP tags, we need to set up four variables, which are the connection parameters we need to gain access to the database – the name of the database, the name of the server it's on, the account name of the user we are using to connect to it, and that user's password.

I could use four separate variables for this, for instance `$dbName`, `$dbServer`, `$dbUser`, and `$dbPass`, but to keep things neat, I'll use one associative array for all four values.

We'll use `$dbConnect` as the array name. Notice that I use camel case for my PHP variables.

Then we set up `$dbConnect` as an array:

```
$dbConnect = array();
```

and inside the parentheses, assign the four values we need as a series of key-value pairs.

We're using our local machine, and the server name, as we know, is 'localhost', so

```
$dbConnect = array(
    'server' => 'localhost',
);
```

We'll use the default admin username, 'root' to gain access:

```
$dbConnect = array(
    'server' => 'localhost',
    'user' => 'root',
);
```

As we're working on our local system, no password is needed so we can leave this blank:

```
$dbConnect = array(
    'server' => 'localhost',
    'user' => 'root',
    'pass' => '',
);
```

Lastly the name of the database is 'test_db':

```
$dbConnect = array(
    'server' => 'localhost',
    'user' => 'root',
    'pass' => '',
    'name' => test_db'
);
```

We are using 'root', the system administrator's account, as the username to connect to the database, and we have set the password to empty by enclosing nothing within quotes, meaning there is no password. We would not do this if the project was online as there is no security at all; we are handing over full access to our database with no password. But for the moment we're working on our local system only so it doesn't matter.

Those are our four connection parameters.

Now we connect to the database using a MySQLi connection, which has superseded the older MySQL_connect command:

```
new mysqli();
```

We start a new instance of the mysqli class with the command new mysqli, followed by the four arguments we have just set up – the server name, user name, password, and database name, and these must be entered in that order:

```
new mysqli(
    $dbConnect['server'],
    $dbConnect['user'],
    $dbConnect['pass'],
    $dbConnect['name']
);
```

To use this class, we need to create an object, which we do in the same way as for assigning values to variables, with the object name beginning with a dollar sign and equals:

```
$db = new mysqli(
    $dbConnect['server'],
    $dbConnect['user'],
    $dbConnect['pass'],
    $dbConnect['name']
);
```

Now the object `$db` gives access to a range of methods which are available in the MySQLi class. Objects are rather like super-powerful associative arrays; they contain a whole range of information which we get by calling various methods.

For example, `$db` now stores information about the host and connection type. We can access some of this by using

```
echo $db->host_info;
```

Save this and refresh your browser and you should see information about the 'localhost' connection.

Every time you see this `->`, the arrow operator, a dash and a greater than symbol with no space between them, we're accessing information in the object to the left, in this case $db, using the method on the right, in this case `host_info`.

Information about connection errors is accessed using the method `connect error number`, which returns a numerical error code.

Enter `echo $db->connect_errno;` and refresh the page in your browser.

You should see the number 0 at the top of the window. This means that there was no error in the connection. Any number higher than 0 means that there was some error.

As it stands, that is not very convincing proof that we have connected to anything, so let's verify that we are actually talking to the database, by creating an error, first at the PHP end and then at the database end.

Change the value of `$password` to anything other than empty, which will make the connection fail, and refresh the page in your browser.

You should see an error message, telling you the line number where the error occurred.

Then return the value of `$password` to `''`, single quotes enclosing nothing, to set the password to empty, and refresh the page and the problem is solved: zero is displayed again.

Now go into phpMyAdmin in a new tab in your browser and change the name of the database by clicking on 'databases', then on 'test_db', then 'operations', and then in 'Rename database to', change the database name and press 'Go'.

Go back to the other tab and refresh the page again and you will get a different error message saying that the database 'test_db' is unknown.

So by creating these errors we have proved that we are indeed talking to the database.

In phpMyAdmin, rename the database back to 'test_db' and the error code will return to zero.

We can use this error code to stop the program if a connection error occurs, using an IF statement as we did earlier.

If there's any problem, then the value of `$db->connect_errno` will be greater than zero, so an IF clause which tests for that condition will trap errors.

Write the condition which tests for `$db-> connect_errno` being greater than zero and if it IS, echo the message "Database connection error" and exit the program.

The code is:

```
if($db->connect_errno > 0) {
    echo "Database connection error<br>";
    exit;
}
```

If we want, we can get information about what went wrong by concatenating onto the string "Database connection error" the method `connect_error`:

```
if($db->connect_errno > 0) {
    echo "Database connection error " . $db->connect_error .  "<br>";
    exit;
}
```

which will give us more information about the nature of the error.

This is for debugging purposes – we would want to remove this in production as we don't want to reveal information about our system to the world.

`Exit` causes the program to halt so no more processing is done.

We have now successfully connected to the database. In the next lesson we will embed an SQL query in PHP and use it to select the full set of data in the database and display it in the browser.

Lesson 51: Select all records from a database

In this lesson you'll learn the basic way to embed an SQL query in a PHP script and intersperse static HTML with PHP code so that dynamic data can be displayed in the browser.

To display data, we first need the SQL query to select all the data from the 'test_db' table, and we need to embed this in PHP.

Can you remember the SQL query, using `SELECT FROM`, which would return ALL RECORDS from the table named 'test_db', with an underscore in the name? Try now to write this SQL query just as we did in the chapter on MySQL. As we are already connected to the database, there is no need to specify the database name – the table name alone is enough.

The answer is: `SELECT * FROM `test_db``, with or without backticks round the table name.

Now add on a clause to ORDER the results BY the field name:

```
SELECT * FROM test_db ORDER BY name
```

That's the SQL query we need. But as it stands it's not useable in PHP. To make it useable we surround the whole SQL query in quotes, and assign it as a string to a variable, which I'll call `$sql`:

```
$sql = "SELECT * FROM `test_db` ORDER BY `name`";
```

To run this SQL query, we use it as the argument of another mysqli method, working on the object `$db`. This time we use the method 'query', operating on `$sql`:

```
$db->query($sql);
```

And we assign the result of this method to another object, which we'll call `$result`:

```
$result = $db->query($sql);
```

Then we can access different parts of this `$result` object with various different mysqli methods.

To return the results in an orderly way, we can loop through them using a `while` loop, using the `fetch_object` method working on `$result`:

```
while ($row = $result->fetch_object()) {}
```

`Fetch_object` returns an associative array, with key/value pairs as we saw earlier, the keys being the field names in the table.

Now we loop through the records, assigning the values of each row to variables.

We get the values of each key name using `$row ->` and then the field name.

So to get the value of the ID for the current row and assign it to a variable named `$id`, we use `$id = $row-> id;`

Then we repeat this for the other field names in the table:

```
$name = $row->name;
$password = $row->password;
$secrets = $row->secrets;
```

Type in `echo "$id  $name $password $secrets<br>";` and refresh the page in your browser and you should see the full set of data in the table.

But we're not quite done yet with displaying database records as HTML: we need to guard against characters in the database which could cause problems in the HTML.

We do this by passing the database result through a built-in PHP function called `HTMLentities()`, which converts any problematic characters to HTML code.

Around the `$row` object and the key it uses to look up the fieldname in the database, wrap `htmlentities()` :

```
$name = htmlentities($row->name);
```

and specify that we want to convert quote marks using `ENT_QUOTES`, and that we want to output UTF-8 encoding using "UTF-8", in double quotes:

```
$name = htmlentities($row->name, ENT_QUOTES, "UTF-8");
```

Now if we have a username with characters in it which would cause problems in HTML, they will be replaced by HTML entity codes.

We can demonstrate this graphically with the extreme example of a name with the angle brackets used in HTML tags. With htmlentities they are displayed OK – we can see in the source that they are replaced by HTML codes – but without htmlentities they are invisible because the browser thinks it has encountered an opening HTML tag.

Apply `htmlentities()` to the 'password' and 'secrets' fields as well in the same way. There's no need to do this for the ID as this is bound to be an integer so it can't cause problems.

This displays the full list of names from the database and arranges them in alphabetical order. But it's messy. Let's practice interspersing PHP with HTML to output the data as a table.

A basic HTML table is provided for you in this chapter in the Working Files. Open this up in Komodo Edit and see if you can intersperse the PHP and HTML to get the desired output.

A possible answer is provided in the Working Files.

Lesson 52: Select matching records using WHERE ... AND

In this lesson I am going to demonstrate, in a very simplified way, how a login system works, and then in the next lesson, how a badly coded login system can easily be broken into. In any system requiring a login, the purpose is that a user logs into their account using their username and password and can see their own data but no-one else's. This means we need to get a user's login credentials and check them against the database, and if we find matching records that person is logged in so they can see the records which match that account.

First, open phpMyAdmin and click to go into the 'test_db' database and then the table and remove those HTML tags from the first record which we used to test `htmlentities()`.

Go to the Working Files for this lesson and copy the files 'connect.php', 'login.html' and 'display.php' to 'php-demos', overwriting the existing version of 'connect.php'.

'Connect.php' is the same as before except that I have removed the lines which output data, so it just does the job of connecting to the database and no more.

'Login.html' is a simple HTML form which gets the username and password using input boxes and sends these to 'display.php', using the POST method.

In 'display.php', we include 'connect.php' to connect to the database and then use the `if isset()` statement we learnt about before to check that username and password have been found in the `$_POST` superglobal, and if so we assign each of them to variables of the same name.

At the moment the SQL query will select all data from the database regardless of what the user enters in the login form. To select only data which matches the user's credentials, we need to add a WHERE clause to the SQL query: `WHERE `name` = '$username'`. But that will not check that the password is correct so we need to add `&& `password` = '$password'`:

```
$sql = "SELECT * FROM `test_db`
    WHERE `name` = '$username' && `password` = '$password'
    ORDER BY `name`";
```

Now if you run 'login.html' and put in a correct username and password, for instance 'user1' and 'user1pass', 'user2' and 'user2pass', *etc.*, you should get that user's record displayed.

That's the basis of selecting a particular user's data following a successful login, but it is full of security holes.

Our project isn't meant to be high security in any way, but we still need to do better than this – in the next lesson I will demonstrate how SQL injection can be used by a fraudulent user to display all the records from the table without knowing anyone's username or password.

Lesson 53: SQL injection demonstrated

SQL injection is a technique used to gain access to the content of a database without providing the correct user credentials.

This is done by inputting bits of PHP code using the username or password field of a login form in such a way that the logic of the SQL query is altered.

PHP scripts which provide access to databases should be written in such a way that SQL injection is difficult if not impossible. At present our script, 'display.php', which returns records from the database 'test_db', is wide open to such an attack, and in this lesson I'll demonstrate how this is done.

Open up a new tab and use Google to search for 'SQL injection'. Go to the Wikipedia page on the topic at http://en.wikipedia.org/wiki/SQL_injection and scroll down until you find the bit of code:

```
' or '1'='1
```

Copy this and paste it into the password input box in 'login.html'. You can leave 'username' empty or put random text into it.

Press 'Login' and you will see the entire contents of the database – you have just cracked the security and got inside without supplying any username or password!

Let's look and see how this works.

In 'display.php', delete $password from the SQL statement, leaving the single quotes enclosing it in position, and paste in the SQL injection snippet:

```
$sql = "SELECT * FROM `test_db`
    WHERE `name` = '$username' && `password` = '' or '1'='1'
    ORDER BY `name`";
```

If we enter nothing for 'username' and this code snippet for 'password', we are effectively saying "Show everything if the username is `''` or password is empty **or `1 = 1`**". And, as the username is empty and 1 does equal 1, we gain access to all the data.

The reason for the vulnerability is that we are able to pass the single quote marks and the `or` statement straight into the SQL query, radically altering its meaning.

In the next lesson I will demonstrate two ways of overcoming this vulnerability.

Lesson 54: Combating SQL injection with mysqli real_escape_string

We have just seen how raw user input can be injected into an SQL query and used to alter the logic of the query. The culprit is the single quote. If this were not allowed to enter the SQL query, the problem would be solved. To achieve this, we need to escape these quote marks with backslashes, just as we did earlier on in the lesson on string delimiters. One way of doing this is to use the PHP function `mysql_real_escape_string()`.

So we can see the effect of this, let's first edit 'display.php' to echo out the username and password:

```
...
if (isset($_POST['password'])) {
    $password = $_POST['password'];
}

echo "Username: $username<br>";
echo "Password: $password<br>";
```

Just above that, in the variable assignment, instead of taking the POST value raw, wrap `mysql_real_escape_string()` around it:

```
if (isset($_POST['password'])) {
    $password = mysql_real_escape_string($_POST['password']);
}
```

Save and refresh. Now if you try to use the PHP snippet to get into the database, it won't work anymore, and we can see the single quotes have been escaped with backslashes so they can no longer interfere with the logic of the SQL query.

`Mysql_real_escape_string()` performs a standard battery of escapes regardless of the data it receives.

A slight improvement on it is to use mysqli's `real_escape_string()`, which acts on the `$db` object and takes into account the character encoding of the database:

```
$password = $db->real_escape_string($_POST['password']);
```

As this takes into account characters already made safe by the database encoding, it saves server effort by avoiding making unnecessary changes to the data.

Either of those methods are fine and effectively prevent SQL injection. A further method is to use prepared statements, where the user input and the SQL query are completely separate.

We'll look at these in the next lesson.

Lesson 55: Combating SQL injection with prepared statements

Another way of combating SQL injection is to use prepared statements. Prepared statements remove any danger of user input interfering with the logic of the SQL query because the SQL query logic and the user input are sent along separate channels to the database. They also escape special characters such as the single quote mark, so you don't need to use `real_escape_string()` with them.

First, edit 'login.html' so that the form 'action' tag points to a new file, 'display-prep-stmt.php':

```
<form action="display-prep-stmt.php" method="post">
```

Then copy 'display.php' and name the copy 'display-prep-stmt.php'. Open this new file in Komodo Edit.

We connect and set the username and password exactly as before. We can delete the echo statements for the username and password – they will be exactly as we enter them and will be escaped later, before they go into the SQL query.

The table header is just as it was before.

Then we need to start our PHP section and set up our prepared statement.

We start out by giving the object a name; I'll call it `$stmt`. Then "equals" and then the "prepare" method, operating on the `$db` object:

```
$stmt = $db->prepare()
```

And in the prepare method's parentheses we put the SQL query to prepare. This is the same as before except instead of putting the user input variables straight into the query we replace them with question marks:

```
$stmt = $db->prepare("SELECT * FROM test_db
    WHERE name = ? && password = ?");
```

Now we need to specify what to replace those question marks with and we do this by binding parameters to the statement.

The two parameters we need are `username` and `password`, both of which are string variables.

We bind parameters using the method `bind_param()` working on `$stmt`:

```
$stmt->bind_param();
```

and inside its parentheses we will specify the variables we want to use. But before the variables we need to specify the data type of each of them. We do this using a series of one letter codes, each one corresponding to the variable in that position.

So for example, if we wanted to pass an ID number, which is an integer, then a username and a password, both of which are strings, we would use `'iss'`, the 'i' in the first position corresponding to the first variable, the integer, then the 's' in second position corresponding to the second variable, and the next s in third position, corresponding to the third variable.

The only ones we need to be concerned about are lower case 'i' for an integer and lower case 's' for a string. You can read about the two other data types on php.net: http://www.php.net/manual/en/mysqli-stmt.bind-param.php.

We just need two variables, both of which are strings, so we start with `'ss'`, in single quotes, and then the variables, username and password, with commas separating them:

```
$stmt->bind_param('ss', $username, $password);
```

Then we need to use `bind_result()` to link the statement to the database fields. Again this is a method, operating on `$stmt`:

```
$stmt->bind_result();
```

Inside the parentheses we need to put, in the order they appear in the MySQL database, variables corresponding to all the database fields: ‘id’, ‘name’, ‘password’, and ‘secrets’:

```
$stmt->bind_result($id, $name, $password, $secrets);
```

It is vital that all the database columns are included and that they are in the right order, as they appear in phpMyAdmin, so open up phpMyAdmin and check the order of the fields.

Then we execute the prepared statement using `execute()`:

```
$stmt->execute();
```

Printing the results out is easier than the previous method. We open a `while` loop,

```
while () {
}
```

and the loop is built using the `fetch` method:

```
while ($stmt->fetch()) {
}
```

Close off the PHP with `?>` and we are back to plain HTML.

Inside the loop we start a new HTML table row each time the loop runs

```
<tr>

</tr>
```

and then inside each row we return four table cells, containing the variables we want to display:

```
<tr>
    <td></td>
    <td></td>
    <td></td>
    <td></td>
</tr>
```

and then inside the table cells we echo out the variables we want, inside `<?php ?>` tags:

```
<tr>
    <td><?php echo $id; ?></td>
    <td><?php echo $name; ?></td>
    <td><?php echo $password; ?></td>
    <td><?php echo $secrets; ?></td>
</tr>
```

Re-start PHP and put in the closing brace of the ‘while’ loop:

```
...
</tr>
<?php
    }
?>
```

And, back in plain HTML, end the table with `</table>`:

```
...
</tr>
<?php
    }
?>
</table>
```

Finally, re-open PHP and close the prepared statement to free up resources:

```
</table>

<?php
$stmt->close();
?>
```

And that’s done. We now have a bullet-proof way of guarding against SQL injection: the tricky PHP snippet is now of no use to hackers.

Lesson 56: Add data through PHP interface

We saw earlier how to add, modify and delete database records using phpMyAdmin – now we'll do the same using a simple interface in PHP. I have prepared a file to get us going – this is 'display-and-insert.php', in the Working Files folder for this lesson. Copy this to 'php-demos' and then open it in Komodo Edit. You will see it contains much of the same code as the script 'display.php' from the lesson before last.

We connect to the database and then use the same method as we used in 'display.php' to display the data in the database. There is no need to use a prepared statement here because there is no user input to sanitise. Then as before we have a table header and then a loop using `fetch_object`, within which we echo out each row of data as a table row. Below that is a form pointing to 'insert.php', within which is a new table with input boxes in the table cells where we can type in the data we want to add, and below that a submit button. Now all we need to do is to write the SQL in 'insert.php' to insert this new data.

There is a partially prepared version of this in the Working Files. Copy it to 'php-demos' and open it in Komodo Edit.

We have already connected to the database and assigned the values of the POST superglobal to variables.

As before we will set up an object called `$stmt`:

```
$stmt = $db->prepare();
```

and inside the parentheses for the `prepare` method we will us a prepared SQL query. We saw the syntax for this in the chapter on SQL – it is `INSERT INTO`, then the table name, then in parentheses the list of affected field names, separated by commas, then `VALUES`, and then in parentheses a matching list of values. In the prepared statement the values are replaced by question marks:

```
$stmt = $db->prepare("INSERT INTO `test_db` (name, password, secrets)
VALUES (?, ?, ?)");
```

Then we use `bind_param` as before, with three strings whose data type is indicated by `'sss'`, and then the three variables to insert: `$username`, `$password`, and `$secrets`:

```
$stmt->bind_param('sss', $username, $password, $secrets);
```

We leave out the id, both here in the SQL and in the table. Because the field is set to auto_increment, its value will be automatically set.

Then we execute the statement and finally close it:

```
$stmt->execute();
$stmt->close();
```

We can redirect the browser back to the originating script, 'display-and-insert.php', by setting its header location using `header("Location: display-and-insert.php")`.

That's the basic way to insert data using a prepared statement in PHP.

In the next lesson we'll adapt these two scripts to delete data from the database.

Lesson 57: Delete data through PHP interface

We've made a simple PHP interface to add new data to the database and in this lesson we'll make a similar one to delete data. We'll do the HTML interface together and then I want you to adapt what we did in the previous lesson to write the code to delete data instead of inserting it. Most of the code is identical to that used in the previous lesson, so we will modify the two scripts we just used.

In 'php-demos', copy 'display-and-insert.php', paste the copy back into the same folder, and rename it 'display-and-delete.php'. Do the same for 'insert.php', naming the copy 'delete.php'. Open both up in Komodo Edit and switch to 'display-and-delete.php'.

To delete data, we need to be able to select each record individually, so we need another column in the table.

In the table header add a fourth column title inside the `<tr>` tag and put the title 'Delete' inside it inside `<th>` tags: `<th>Delete</th>`.

Inside the loop, add in the line to assign the value of the field named `id` to the variable named `$id`: `$id = $row->id`. There is no need to use `htmlentities()` because we know this field will always contain an integer and cannot have any problematic characters in it.

Matching the fourth column, 'Delete', in the table header, add a fourth column in the table body, with the text "delete" inside it: `<td>Delete</td>`.

Surround the word 'Delete' with an `a href` tag, pointing to 'delete.php':

```
<td><a href="delete.php">Delete</a></td>
```

We will use this to link to 'delete.php', and will tell 'delete.php' which record to delete by passing the id number of the record in the URL, using `?id=` and then the id number, using PHP to echo the value of the variable $id. It may be easiest, to get the syntax right, to temporarily put in any number in the link URL, I will put '1' here:

```
<td><a href="delete.php?id=1">Delete</a></td>
```

and then type over the number 1 with the PHP tags and echo command `<?php echo $id; ?>`:

```
<td><a href="delete.php?id=<?php echo $id; ?>">Delete</a></td>
```

We no longer need the second table, for inserting data, so delete everything after the closing table tag.

That's it for this file. Switch to 'delete.php'.

Now, using your knowledge of SQL, adapt this file to use the ID passed in the URL and delete the matching record in the database.

If you get stuck, the answer is in the Working Files for this chapter.

That concludes this chapter on simple PHP interfaces for MySQL queries. In the next chapter we will go back to our project files and put our knowledge on PHP and MySQL into practice, turning our present static framework into a dynamic site which takes its information from the database and allows the user to alter that information through the browser.

Chapter 9 - From static HTML to dynamic PHP

Lesson 58: Introduction to Section 9 – PHP

In this chapter I will take you, step-by-step, through the process of converting the static HTML we have written to a site which gets data from a database and uses just one page as the holder to display varying data, the data displayed depending on variables set in the URL as the user chooses various different movie-goers.

If you follow all the lessons in this chapter, you will get a thorough coverage of the programming logic involved, as well as the logical steps a coder might go through in this process.

At various points I will pause and ask you to complete the next small step, based on what we have just been doing.

There are also four major assignments, where I ask you to complete a whole piece of the project – these require more thought of the logic involved.

You may find the lessons too microscopic in their detail, in which case the Working Files allow you to skip forward. The Working Files corresponding to each lesson have the code as it stands at the END of that lesson, so if you find I'm going too slowly, simply move on, and use the appropriate version of the code.

Lesson 59: Efficient, reusable code with PHP includes

Open up all three HTML files in the 'favouritemovies' folder in Komodo Edit.

If we switch from one tab to another, we can see that the start of the three files is very similar in each – in fact everything is identical down to the closing header tag. We've already experienced some of the problems of working with multiple files with duplicate content – making the same changes repeatedly is a pain and likely to give rise to errors. It is also inefficient for the browser as it has to reload this content every time it loads a new page, even though the code is identical.

So the first thing we can do is to remove this duplicate content from these three files and put it in PHP include files. To do this, we need to convert our HTML files into PHP files, so close down Komodo Edit and change the .html extension to .php on all of them.

Make a new directory called 'php-includes' in the 'favouritemovies' folder, alongside the 'images' and 'css' folders. I use dashes, not underscores, for all my file system names, for consistency.

Open up the three PHP files in Komodo Edit again.

Now in 'index.php', select everything from the very top of the file including the doctype declaration, down to the closing head tag. Cut this out and paste it into a new code pane, and save this new file in 'php-includes' as 'head.inc.php'.

We will give all our include files the name '...inc.php', not just '...php', to make it clear that they're to be run as included files and not directly in the browser.

Now back in 'index.php', we need to include that file. In the previous chapter we looked at the difference between the `include` and `require` commands in PHP, and at `include_once` and `require_once`. In this case we'll use `require_once`, because the included files are essential, and each of them is only needed once.

Inside PHP tags, type in `require_once` and then in single quotes the relative path from this file to 'head.inc.php':

```
require_once 'php-includes/head.inc.php';
```

Select and copy this and do the same in the other two files, replacing everything from the top down to and including the closing head tag with this line.

Save and check the three files in your browser. Make sure you are using http://localhost in the browser address bar, not loading them as local files.

Now any changes we make in 'head.inc.php' will take effect in all three pages, making updating the website much easier.

The header section is an obvious candidate for the same treatment as it is identical in all three files, so we'll do the same for that.

Select the opening body and the whole of the header section, paste it into a new code pane, and save it as 'header.inc.php'. We'll maintain the names of the HTML tags, even though they are so similar, as this is the best way to show which bit of the document is in which included file.

In all three main files, copy the first `require_once` line and paste it in just below, changing it to include this file.

The main navigation section is identical in all three files so repeat the process with the `<nav class="navigation">` section, cutting it out and saving it as ‘navigation.inc.php’, again in the includes folder. Put in the `require_once` command, including ‘navigation.inc.php’, in all the files and cut out the duplicate HTML.

In the two movies pages, we can do the same with the `<nav class="favs_list">` section and save it as ‘favourites.inc.php’, add the `require_once` command, and cut out the now unwanted HTML.

The footer is also identical in all three files, so select everything in ‘index.php’ from the opening `<footer>` tag to the end, cut it out, and paste it into a new code pane and save it in ‘php-includes’ as ‘footer.inc.php’.

Replace the footer section in all three main PHP files with PHP tags enclosing a `require_once` command just as we did at the top of the file, but this time to ‘footer.inc.php’.

That’s all the duplicate code removed. The three main files now differ only in their central section. It would be even better if we had just one file for three types of display: the movie list page, the single movie page, and the admin page. But before we can do that, we need to pass a parameter in the URL, which decides which of these three pages the user has chosen to see.

This is the topic of the next lesson.

Lesson 60: One index file for single movie and movies list

We have moved the duplicated code for the head, header and footer out to PHP includes, so any changes we make in any of these included files will affect all the pages that use them. But we can do better than this and remove the parts of the movie display pages which are different so that we have just one index file for both the single movie and the movie list page.

In the stand-alone chapter on PHP I demonstrated the use of `$_GET`, to get a variable name and its value from the URL and display it in a webpage. We'll use `$_GET` now to check whether a user has chosen a particular movie or not, and combine it with an `if` condition to display the single movie page if a movie has been chosen or the movie list page if not.

To do this we'll use a variable called `$movieID`, which will eventually contain a unique identifying number linked to the database for each movie. For the moment we'll just look at whether it contains any value at all and if so we'll show the single movie page.

First of all, we need to get the variable name and value from the URL.

In 'php-includes', make a new text file and name it 'get-variables.inc.php'. We'll use this to get all variables from the URL.

Open 'get-variables.inc.php' up for editing and type in opening and closing PHP tags.

We learnt in the stand-alone chapter on PHP how to GET the name and value of a variable from the URL, so before I give you the answer, try now to type in the command necessary to get the value of a variable, specified in the URL with `?movieID =`, and assign this to a variable named `$movieID`.

The answer is:

```
$movieID = $_GET['movieID'];
```

I use camelCase for my PHP variables and all lower case with underscores for variables in the URL. I don't use camelCase in the URL because it might cause problems.

Just to check this is working, we'll temporarily echo this out in the script using `echo $movieID`.

We're not planning to run this script directly in the browser once we're finished but just for the moment, point the browser to it, in http://localhost/php-demos/php-includes/get-variables.inc.php?movieID=1, just to check that it works.

It should echo out the number 1 – change this and the number in the browser window should change, showing that it has GOT the value from the URL and assigned it to the variable `$movieID`.

Now remove `?movie_id=1` from the URL.

We get an error – "undefined index". This is because we are trying to get a non-existent index from an associative array. This is no good because we don't want to throw an error if no variable is specified.

We overcome this by running the code only if `movie_id` is actually given a value in the URL, using an `if` condition.

Type in `if` and then the parentheses which enclose the condition:

```
if () {
}
```

Then inside the parentheses `isset()`:

```
if (isset()) {
}
```

And then inside the parentheses for `isset()`, `$_GET['movieID']`:

```
if (isset($_GET['movie_id'])) {
}
```

Then assign the value of `$_GET['movie_id']` to the variable `$movieID`:

```
if (isset($_GET['movie_id'])) {
    $movieID = $_GET['movie_id'];
}
```

Now the variable assignment will take place only if `movie_id` is in fact set in the URL; otherwise nothing will happen.

After the closing brace of the `if` condition type in `else` – and within its braces type in `$movieID = "Movie ID not set"`:

```
if (isset($_GET['movie_id'])) {
    $movieID = $_GET['movie_id'];
} else {
    $movieID = "Movie ID not set"
}
```

Now if we refresh the page in the browser we will get this civilised message instead of a system error message.

Delete the `else` clause and the `echo` statement as we don't want them anymore.

Now we can use this condition to load the single movie page if `$movieID` is set and the movie list page if it is not set.

Open up 'index.php' in Komodo Edit and above the first `require_once` line, type in:

```
require_once 'php-includes/get-variables.inc.php';
```

Now we can use that code just as if it were part of 'index.php' and have an `if` condition to load the single movie page if `$movieID` is set, or the movie list page if it's not.

Now we'll do more of the cutting and saving as include files that we did in the previous lesson.

Go down in index.php and select and cut out the whole section

```
<section "class=movie_list">
    ...
</section>
```

Paste this into a new code pane and save this in 'php-includes' as 'movie-list.inc.php'.

Open up ‘movie-single.php’ and cut out the whole section

```
<section class= "movie_single">
    ...
</section>
```

and save this as ‘movie-single.inc.php’.

Now you can delete ‘movie-single.php’ – we don’t need that anymore – all movie displays will be done in the index file.

Back in ‘index.php’, all we need now is a condition to include the movie-single include file if `$movieID` is set and the movie list page if it is not, using an `if ... else` statement.

Try to write this yourself before I give you the answer.

The answer is:

```
if (isset($movieID)) {
    require_once 'movie-single.inc.php';
} else {
    require_once 'movie-list.inc.php';
}
```

Now load ‘index.php’ without assigning `movie_id` any value in the URL, and you should get the movie list page.

Assign it a value using `?movie_id=` and a value in the URL, and you should get the single movie page.

That’s much more efficient – now we have a single page to maintain rather than two, and any edits anywhere will affect both displays instantly.

In the next lesson we’ll follow the same principle to make our admin page similarly multipurpose, able to display both the users admin table and the movies admin table.

Lesson 61: One admin file for users and movies admin

In the previous lesson we got the value of a variable, `movie_id`, from the URL, and used an `if ... else` statement in 'index.php' to display different content according to whether or not `movie_id` had been set in the URL.

In this lesson we'll use a similar principle to display two different admin pages – one to add, modify and delete users and the other to add, modify and delete movies, according to the value of a variable set in the URL again.

This time we'll use `page` as the name of the variable in the URL and assign its value to a PHP variable of the same name, `$page`. If `$page` has the value 'users', 'admin.php' will display the users admin table. If it has the value 'movies', 'admin.php' will display the movies admin table.

Because the coding involved is very similar to what we have just done, I'll ask you to write this yourself and see if you can get it working.

Go to the Working Files for this chapter and inside the folder 'beginning' you will find the two new include files you need, 'admin-users.inc.php' and 'admin-movies.inc.php'. Before you start, copy these into your 'php-includes' folder.

Open up 'admin.php' and cut out the whole of the section

```
<section class="admin">
    ...
</section>
```

Now use your prior experience and in 'get-variables.inc.php', set `$page` to the value of `page` in the URL, just as we did for `movie-id`.

Then in 'admin.php', write an `if ... else` statement similar to the one we did earlier. But this time instead of simply testing whether a variable is set or not, you need to test whether the value of `$page` is 'users' or 'movies' and display the appropriate include file.

You will also need to have a way of catching errors in case the user is fooling around and enters something other than 'admin' or 'users' in the URL.

Pause the video and do your best with this now. At the end, or if you get into difficulties, go to the Working Files for this chapter, and in the folder called 'completed' you will find the prepared version.

There are also some changes to 'style.css' for the movies-admin page so select and copy everything in that folder and paste it into 'favouritemovies', overwriting all the files there.

I have written the IF statement so that it first checks whether `$page` equals 'users', and if so includes the users admin page:

```
if ($page=="users") {
    require_once 'php-includes/admin-users.inc.php';
}
```

Then if `$page` is NOT equal to 'users', it checks to see if it equals 'movies' and if so includes the movies admin page:

```
else {
    if ($page=="movies") {
        require_once 'php-includes/admin-movies.inc.php';
    }
}
```

Then if `$page` is neither “admin” nor “users” something must be wrong and the final `else` clause will run:

```
else {
    echo "<div class='message'>";
    echo "<h2 class='alert'>Please use the admin menus to navigate</h2>\n";
    echo "</div>";
    require_once 'php-includes/footer.inc.php';
    exit;
}
```

This prints out a bit of HTML to indicate that an error has occurred and asks the user to use the menus instead of fooling around in the URL. After that the program exits to prevent any more errors occurring. This sort of error catching is important to prevent the website from falling over if unexpected results are entered.

We need a bit more error catching; if a user goes to ‘admin.php’ and does not enter any value in the URL for `page`, we get an error. This is easy to do: in ‘get-variables.inc.php’

```
if (isset($_GET['page'])) {
    $page = $_GET['page'];
} else {
    $page = "error";
}
```

Finally, open ‘navigation.inc.php’ and notice that I have changed the `href` targets for the two list items to point to `admin.php?page=users` or `admin.php?page=movies` as appropriate.

Now our PHP code is efficiently written and we are ready to go on to connecting to the database and beginning to populate our website with data from it.

Lesson 62: Initialise and connect to the favourite movies database

Before we set up our database connection, let's check that the database is in its proper state, by reinitialising it.

Go to phpMyAdmin and click on 'Import' at the top. Click on 'Browse', and select 'initialise_movies_ database.sql' from the 'INITIALIZE DATABASE' folder in the Working Files.

Click 'Go', and the database should be restored to its original state.

Now we need to write the code to connect to it, using the mysqli connection we learnt about.

Go to your 'favouritemovies' folder and in Komodo Edit open up 'index.php' and 'admin.php' and in each of them, above the line requiring 'get-variables.inc.php', add a line requiring a new file in the php-includes folder, called 'connect.inc.php'.

This is where we will put our connection details, to keep them organised in a file which is clearly named for its purpose.

Remember we are using '...inc.php' in all the file names in this directory to make it clear that they're included files and should not be run directly in the browser. As before, we are dividing our script up into small individual files, each of which does a clearly identified job. In this way, we can reuse the same piece of code as many times as we want and the code will be much easier to manage because it doesn't become long and unwieldy. We also don't need much in the way of comments because it should normally be pretty clear what's going on.

Except for the database name, the connection details are exactly the same as those we used for the previous chapter so to save time you can copy 'connect.php' from 'php-demos' to 'favouritemovies/php-includes' and rename it 'connect.inc.php'.

Open it up and change 'test_db' to 'movies', the name of our project database, and that's our database connection done.

In the next lesson, I'll ask you to modify existing code to display a list of all the moviegoers in the database.

Lesson 63: Displaying dynamic data - the list of users

We are now working on the primary navigation for the site – the list of movie-goers.

At present the list is hidden in the stylesheet, so the first thing to do is to show this list instead of the admin menu.

Open up 'style.css', find the styles for `ul.users_menu` and `ul.admin_menu` and move `display: none` from the users menu to the admin menu. Save and refresh and you should see the dummy users menu saying "list of other users". Close 'style.css'.

We'll work first on displaying a drop-down list of movie-goers names from the database, replacing the dummy list we currently have.

Let's look at the finished site to see what we want to achieve – we will be able to switch from one movie-goer to another by clicking on their names in this pull down list and then once a movie-goer is selected we can add and delete their favourite movies by clicking on the favourite icon or dragging favourites to the trash can.

This will be our main navigation interface for the website, allowing us to switch quickly from one movie-goer to another so we can test the instant update effects easily.

We will not be doing a real login system with usernames and passwords because then it would be difficult to test the website out and switch quickly from one user to another to see the instant update effect in action; we would constantly have to remember and enter usernames and passwords, and also the database security issues are beyond the scope of this course.

As you can see, there are various states of the navigation menu depending on whether or not a movie-goer is selected. When a user first arrives at the site, no movie-goer is selected, and the drop-down list contains all the names in the database. Clicking on a movie-goer's name displays that movie-goer's favourites and the navigation list will change – the selected movie-goer's name appears as the heading and the list now contains all the movie-goers except the selected one.

In addition, a 'logout' link appears in a different style at the bottom. Clicking on this deselects the movie-goer.

We will get all of this done of the next few lessons, but we'll begin by replacing the dummy list with a dynamic list in the simplest way possible.

Open up 'navigation.inc.php' and make a new section of PHP code at the top.

We need an SQL query to select all the moviegoers from the 'movie_goers' table.

At the moment we are selecting all the records with no `WHERE` clause, so there's no user input. This means we don't actually need to use a prepared statement because there's nothing to sanitise – we could simply use a plain SQL statement, which would involve less work for the server.

But in the very near future we will be selecting particular records based on the `user_id` number passed in the URL, so to avoid making unnecessary changes later it makes sense to use a prepared statement right from the start.

Try now to write the prepared statement to select everything from the table named 'movie_goers' and assign this to an object called `$stmt`. If you have trouble, look back at 'display-prep.stmt.php' in 'php-demos' and adapt that.

The answer is:

```
$stmt = $db->prepare("SELECT * FROM `movie_goers`");
```

To order the results by the first name, add `ORDER BY `firstname``:

```
$stmt = $db->prepare("SELECT * FROM `movie_goers` ORDER BY `firstname`");
```

We are use the `prepare` method working on the `$db` object and assigning this to a new object called `$stmt`.

Then we need to bind the results from the table fieldnames, in the order they occur in the table structure, to PHP variables:

```
$stmt->bind_result($id, $firstname, $lastname);
```

I'm going to use a new variable, `$id`, to display the values we get from the `user_id` field. I will avoid using `$userID` for this because if we are not careful we will end up with the same variable name being used for two different things – the value set in the URL and the values coming from the database.

Can you remember how to execute the prepared statement?

The answer is: `$stmt->execute();`

Close the PHP section and move down into the relevant part of the HTML, where we start the users menu and re-open the PHP tags.

Delete the two dummy list items.

Convert the plain HTML into echo statements, changing the double quotes in the strings to single quotes so they do not interfere with the double quotes we are using as string delimiters.

Now move down to below the opening `<ul>` tag and start the `while` loop, so that the `<li>` list items are displayed repeatedly as the loop runs.

Can you remember, or look back at 'display-prep-stmt.php' again to refresh your memory, and try to write the loop we need here to display the data we need, the id, firstname, and lastname?

We'll use the fetch method:

```
while($stmt->fetch()){ }
```

Move the closing brace to after the closing 'li' tag:

```
while ($stmt->fetch()) {
   echo "<li><a href='index.php?user_id=$id'>$firstname
   $lastname</a></li>";
}
```

Every time the loop runs, another list item will be displayed.

We need `htmlentities()` to clear up any problems with special characters in the text fields `firstname` and `lastname` but not for the `user_id` field as this is an integer:

```
$firstname = htmlentities($firstname, ENT_QUOTES, "UTF-8");
$lastname = htmlentities($lastname, ENT_QUOTES, "UTF-8");
```

Then `echo` and `<li></li>` tags to format the output as a list item, then `$firstname $lastname` and make a link to 'index.php' with the current `user_id` appended to it:

```
<li><a href='index.php?user_id=$id'>$firstname $lastname</a></li>
```

so that we can select each user by clicking on their names.

And when we have finished using the prepared statement we should close it to free up server resources, so after the closing `</ul>` tag, put `$stmt->close()`.

Save and refresh and we get our first dynamic data from the database.

Lesson 64: Convert users list to function

Our first attempt at displaying dynamic data, the full list of movie-goers, appears fine on the surface.

But before we go further we need to change the way we organise our code because our next step will be to display the selected movie-goer in the heading – here and all the others in the list – or, if no movie-goer is selected, the complete list here and the message to select users at the top. This means we will be generating different lists of data according to whether or not a selection has been made. Remember that the values of variables set in include files are available everywhere throughout the project.

If we continue using nothing but include files as we have done so far, our variables will be available everywhere, even though the variables will mean different things in different parts of the project.

This would quickly lead to trouble. We would have to start using different variable names to get round this – instead of `$user_id` perhaps `$selected_user_id`, `$other_user_id`, things like that. We would also have to do the same thing with our prepared statements to distinguish one from another.

If we followed this way of doing things, our variable naming conventions would become extremely complicated and we would have a great deal of largely duplicated code, which is always a sign that something's wrong.

As we saw earlier in the lessons on functions, variables set inside functions are normally available only inside the function in which they are set, which keeps them under much better control and avoids this problem. If we run our prepared statements inside functions, then we can use the same variable names in different parts of the project with no conflicts.

So in this lesson we'll take the parts of 'navigation.inc.php' that do the work of getting and displaying results, and convert these into a function.

In Windows Explorer, make a new folder in 'favouritemovies' and call it 'functions'.

Just as we named all our include files '...inc.php', we'll name all our function files '...fn.php'.

Having clear naming and file location conventions like this makes it clear what does what in the project and reduces the need for comments in our code.

Some developers give their functions names beginning with 'function' or 'func' but I prefer to have the start of the file indicate its purpose, and also in this way the files are arranged alphabetically in the file manager and it's easier to find them.

Make a new file in the 'functions' folder and name it 'show-users.fn.php'.

We need to include this function file to make it available throughout the project.

The best place to do this is where we put all our other include files, in 'index.php' and 'admin.php'.

We'll keep our code neat by separating the include files and the function files and let's put in a comment to say what we're doing.

Enter `//` to start the comment line – everything up to the end of the line is then ignored by the server and we can write any notes we want.

Then `// Functions`

And add the `require_once` statement for 'show-users.fn.php'.

Then a new line and another comment: `// Includes`. And then the included files.

Do the same in 'admin.php'.

In 'show-users.fn.php', start out by entering the php tags.

So far we haven't used comments in our code because it hasn't seemed necessary – the code has been pretty self-explanatory. But here we will put in a comment telling us which file calls the function – 'navigation.inc.php' – because there's nothing else to tell us and we might get confused about that later.

Functions perform actions, so we'll start all our function names, and the name of the file they're in, with a verb, indicating the action they perform.

Start with the keyword `function`, name the function `showUsers`:

```
function showUsers
```

Function names should be camelCase – in other words if they consist of more than one word, the first letter of the second word should be uppercase, like a camel's hump.

Give it an empty set of parentheses `()` and open a set of braces to enclose the function code and indent the code inside the braces:

```
function showUsers(){
}
```

The main advantage of using functions is that they avoid variable conflict, but that also means they do not automatically get the values of variables set outside them. So this variable does not as yet have access to our connection object, stored in `$db`. To make `$db` useable inside the function we have to declare it as global, with `global $db`:

```
function showUsers() {
    global $db;
}
```

As I said before, making variables global is frowned on by purists, but then they are coming from a higher level of programming skill than we can possibly cover in a beginners tutorial. The alternative is to rewrite the code in object oriented style, which is conceptually far more difficult than anything we have been doing and is more suitable for advanced students who are familiar with the basics already.

Open up 'navigation.inc.php' in another tab and cut out the three lines setting up and executing the prepared statement, and paste them inside the function's braces:

```
function showUsers() {
    global $db;

    $stmt = $db->prepare("SELECT * FROM `movie_goers` ORDER BY
      `firstname`");
    $stmt->bind_result($id, $firstname, $lastname);
    $stmt->execute();
}
```

Now we'll cut the whole of the PHP code we wrote in the previous lesson out from the include file and paste this into the function file, inside the braces:

```
function showUsers() {
    global $db;

    $stmt = $db->prepare("SELECT * FROM `movie_goers` ORDER BY
      `firstname`");
    $stmt->bind_result($id, $firstname, $lastname);
    $stmt->execute();

    $output = "<ul class='users_menu'>";
    while ($stmt->fetch()) {
        $firstname = htmlentities($firstname, ENT_QUOTES, "UTF-8");
        $lastname = htmlentities($lastname, ENT_QUOTES, "UTF-8");
        echo "<li>
          <a href='index.php?user_id=$id'>$firstname $lastname</a></li>";
    }
    $echo "</ul>";
    $stmt->close();
}
```

Make sure the indentation is tidy.

But to get the data we want out of the function and into 'navigation.inc.php', where we want it, we have to change things a bit.

A function can only return to the script which called it one item of data, using the keyword 'return' with a variable name as its argument.

This can be a simple variable or we can loop through a database and push its contents into an array and return this array.

An array contains a whole set of results, although it's only one item itself.

We could use an array here, returning just the variables, and formatting them in 'navigation.inc.php', but if we did this we'd have to loop through it again in 'navigation.inc.php' so as to display its contents.

This wouldn't be a problem with the tiny set of example data we have here, but if we had a large database this extra loop could impact on performance, and causes unnecessary extra work for the server.

To avoid this, we'll return the data as a simple string variable, already formatted as an HTML list, to fit straight into place in 'navigation.inc.php'. This will be more efficient than looping through all the records again.

We need to convert all the `echo` statements into one string variable so that we can return just the one item as a single variable, which we'll name `$output`.

Start out by assigning the opening `<ul>` tag to `$output`, replacing `echo` with `$output =`:

```
$output = "<ul class='users_menu'>";
```

Then inside the loop we'll use using the concatenation operator, `.=`, which appends data onto the end of an existing variable, building `$output` up as the loop runs repeatedly to form eventually the complete, formatted, list item:

```
...

$output .= "<li><a href='index.php?user_id=$id'>$firstname
$lastname</a></li>";
...
```

After the closing brace of the loop, concatenate the closing `<ul>` tag to the variable:

```
$output .= "</ul>";
```

Then close the prepared statement – we don't need that any more.

Finally, return the value of `$output` to the calling script with `return($output)`, to send the string back to 'navigation.inc.php'.

Now at the top of the navigation file, we need to call the function: `showUsers()`.

We can at the same time assign the result it returns to a variable:

```
$usersList = showUsers();
```

... and then in the place where the list needs to appear, display the variable with `echo $usersList`.

Save and refresh and nothing should have changed – we get our complete list of movies as before – but the code is much better set up for the future.

In the next lesson we'll simplify our code in 'index.php' and 'admin.php' before we go on and transform the `showUsers()` function into a multi-purpose function that can generate all the different lists of users that we need.

Lesson 65: Set include paths in parent files

There's one small matter of housekeeping we can do now, which is to simplify things a bit by using `set_include_path()` in the two parent files, 'index.php' and 'admin.php', to tell the system where to look for include files and functions, which means we will no longer have to specify their paths.

Open up both 'index.php' and 'admin.php', and add as the first line of code:

```
set_include_path('./php-includes');
```

This adds the folder './php-includes' as one of the places to look for included files.

Then using concatenation dots, concatenate `PATH_SEPARATOR`:

```
set_include_path('./php-includes' . PATH_SEPARATOR .);
```

This is like a comma, to allow us to list a series of directories, but using the constant `PATH_SEPARATOR` allows it to work on a variety of operating systems.

Then add the functions directory:

```
set_include_path('./php-includes' . PATH_SEPARATOR . './functions');
```

Now we can include files and functions from the 'php-includes' folder and the 'functions' folder without specifying their path in the script, so in both 'index.php' and 'admin.php', remove 'php-includes' and 'functions' from the paths to all the included files – just the filename will be enough now.

Now we can go on with the next stage, improving our `showUsers()` function.

Lesson 66: User navigation - test if valid user set

So far we haven't written the code in 'get-variables.inc.php' to get the `user_id` from the URL.

Open up 'get-variables.inc.php' and copy the section of code which gets `movie_id` from the URL, paste it back in again changing `movie_id` to `user_id` in all three places, taking care of the naming conventions: lowercase and underscore in the URL, camelCase for the PHP variables:

```
if (isset($_GET['user_id'])) {
    $userID = $_GET['user_id'];
}
```

Now if `user_id` forms part of a link, as it will when a user chooses a movie-goer from the navigation menu, then the variable `$userID` will be set.

Now we'll write a function which tests first of all whether or not `user_id` is set in the URL and returns different codes to another function which displays the appropriate list of movie-goers. So if none is selected, the whole list will be displayed. If one is selected the others will be listed and the selected one will form the header at the top.

Make a new function in the functions folder and call it 'test-users.fn.php'. Include this function in both 'index.php' and 'admin.php', above 'show-users.fn.php':

index.php, admin.php:

```
require_once 'test-users.fn.php';
require_once 'show-users.fn.php';
```

Open 'test-users.fn.php', give it its PHP tags, make a comment saying where it is called, in 'navigation.inc.php, and start the function:

test-users.fn.php:

```
// Called in navigation.inc.php

function testUsers() {
}
```

We need access to the connection object `$db` and the variable `$userID`, so make them both global:

```
function testUsers() {
    global $db, $userID;
}
```

Now we'll check whether `$userID` has been set, or rather check whether it <u>hasn't</u> been set first, using the negation operator, `!`:

```
function testUsers() {
    global $db, $userID;

    if (!isset($userID)) {
    }
}
```

and in that case we'll return the code 'no_id' to the calling script, 'navigation.inc.php':

```
...
    if (!isset($userID)) {
        return("no_id");
    }
...
```

As we saw earlier, the keyword `return` returns one value to the calling script and also stops the function, so no more of the function code runs.

So we don't need an `else` clause – we can now go on and run a prepared statement to check whether the user ID set in the URL matches one of the movie-goers in the database.

You write the prepared statement to select all fields from the movie-goers table where the user ID matches that set in the variable `$userID`.

The answer is:

```
$stmt = $db->prepare("SELECT * FROM `movie_goers` WHERE `user_id` = ?"
);
```

and then bind the parameter to `$userID`:

```
$stmt->bind_param('i', $userID);
```

Now execute the prepared statement:

```
$stmt->execute();
```

To know whether any matching results were found in the database we need to first store the results using `store_result()`:

```
$stmt->store_result();
```

And then find out how many rows were returned using `num_rows`, working on `$stmt`:

```
$stmt->num_rows;
```

which we can assign to a variable, `$numrows`:

```
$numrows = $stmt->num_rows;
```

Then using another `if ... else` condition, if `$numrows` is less than 1, that means no matching user record has been found and we return the code 'invalid_id' and exit the function:

```
if ($numrows < 1) {
    return("invalid_id");
}
```

If `$numrows` is not less than 1, it means that a record has been found and we return the code 'id_set':

```
if ($numrows < 1) {
    return("invalid_id");
} else {
    return("id_set");
}
```

In 'navigation.inc.php', call the function and at the same time assign the result to a variable, `$testUsers`:

```
$usersList = showUsers();
```

For the moment, we'll just echo `$testUsers` out and exit so we can check that it works as expected if we alter `user_id` in the URL. Try it with no user ID set, with a valid user ID, and with an invalid user ID.

It works.

Back in ‘test-users.fn.php’, close the prepared statement just after we have assigned a value to `$numrows`:

```
$numrows = $stmt->num_rows;
$stmt->close();
```

Next we’ll use those codes to tell the navigation display function what to display according to whether `user_id` is set and valid or not.

Lesson 67: Set parameters for show users function

Working in 'navigation.inc.php', we now have a code coming in from the `testUsers()` function telling us whether a valid `user_id` has been set or not. We know that it works.

We'll use this code to send an appropriate parameter to the `showUsers()` function, telling it whether we want to display all the users, the currently selected user for the heading, or the non-selected users, and how we want to format them.

Delete the following lines:

```
echo $testUsers ;
exit;
$usersList = showUsers()
```

The only line left in the PHP code at the top should be `$testUsers = testUsers();`,

As we have several possible codes coming in from `testUsers()`, we'll use `switch`/`case`, as a series of `if` clauses would become cumbersome.

Remember the syntax – `switch` takes an argument in its parentheses and that argument is the code coming in from `testUsers()`, 'no_id', 'invalid_id' or 'id_set':

```
switch($testUsers) {
}
```

It has braces and inside the braces we have a series of `case` tests where we put the various values which the `switch` parameter may have, followed by a colon:

```
switch($testUsers) {
    case "no_id":

    case "invalid_id":

    case "id_set":

}
```

If the code is 'no_id', we want to display all the users in the pull-down menu, because no movie-goer is selected, which we'll do by running showUsers with the string 'all' as its parameter – we'll set up the `showUsers()` function shortly so that this returns all the movie-goers from the database:

```
...
    case "no_id":
        $usersList = showUsers('all');

...
```

We'll want to have the text 'Choose a movie-goer' as the heading for the user menu, which we can do by assigning this to a variable:

```
...
    case "no_id":
        $usersList = showUsers('all');
        $heading = "Choose a movie-goer";
...
```

Then we need `break` to break out of the switch clause, otherwise it will go on and run the next case as well:

```
...
    case "no_id":
        $usersList = showUsers('all');
        $heading = "Choose a movie-goer";
        break;
...
```

In the case that the code is "invalid_id", we want exactly the same response :

```
...
    case "invalid_id":
        $usersList = showUsers('all');
        $heading = "Choose a movie-goer";
        break;
...
```

Finally, in the case that the code is ‘id_set’, add:

```
...
    case "id_set":
        $usersList = showUsers('others');
        $heading = showUsers('current');
        break;
...
```

This will display the current movie-goer as the heading and all the others in the list.

Go down into the HTML in ‘navigation.inc.php’. We already have the PHP echo statement to output the users list – now replace the static text for the heading with `<?php echo $heading; ?>`.

That’s our parameterised function calls set up.

In the next lesson we’ll continue writing the function `showUsers()` so it’s able to make use of these parameters.

Lesson 68: Parameterised show users function

Now we'll make 'show-users.fn.php' able to make use of the parameters we pass to it when it is called in 'navigation.inc.php', displaying the various different user lists according to the parameter set.

We need to use not just `$db`, but also `$userID`, so make this a global variable too:

show-users.fn.php:

```
function showUsers() {
     global $db, $userID;
}
```

Again, 'switch' is more convenient and easier to read than a series of `if` clauses. We'll give the name `$data` to the parameter passed in 'navigation.inc.php' when we call the function:

```
function showUsers($data) {
     global $db, $userID;

     switch($data) {
     }

}
```

In 'navigation.inc.php' we set up three possible values for `$data`: 'all', 'current', and 'others', which are now our `case`s:

```
...
     switch($data) {
        case "all":
        case "current":
        case "others":
     }
...
```

Within each case, we need to set up the things to do differently. So in case "all" we want the prepared statement to select everything from the 'movie_goers' table and display them ordered by first name:

```
...
    case "all":
    $stmt = $db->prepare("SELECT * FROM `movie_goers` ORDER BY
    `firstname`");
...
```

We need to specify the HTML tags to use to format the output; in this case the output is part of an HTML list using `<li>` tags – but make sure you only put 'li', no HTML angle brackets:

```
...
    case "all":
    $stmt=$db->prepare("SELECT * FROM `movie_goers` ORDER BY
`firstname`");
    $tag = 'li';
...
```

and `break`:

```
...
    $tag = 'li';
    break;
...
```

Now you should be able to write all the code to go into the “current” and “others” cases.

In case ‘current’ we want to match the record with the variable `$userID`, using `WHERE `user_id` =` in the prepared statement, and remember the HTML tag is ‘h2’ with no HTML angle brackets:

```
...
    case "current":
       $stmt = $db->prepare("SELECT * FROM `movie_goers` WHERE `user_id` = ?");
       $stmt->bind_param('i', $userID);
       $tag = "h2";
       break;
...
```

In case ‘others’ we want to match all records except the one set in the URL, so we use the negation operator, `!=`, in the prepared statement:

```
...
    case "others":
       $stmt = $db->prepare("SELECT * FROM `movie_goers` WHERE `user_id` != ?
          ORDER BY `firstname`");
       $stmt->bind_param('i', $userID);
       $tag = "li";
       break;
...
```

Delete the old prepared statement.

We already have the `bind_results` and `execute` statements done, as well as the loop and most of the output.

At present we have opening and closing `<ul>` tags to make an unordered list to form the drop-down list of names. We only need these if the value of `$tag` is ‘li’; if `$tag` is ‘h2’ they should be omitted.

So put these inside `if` statements so that they are included only if the tag is ‘li’:

```
$stmt->execute();

if ($tag=="li") {
    $output = "<ul class='users_menu'>";
}

...

if ($tag=="li") {
    $output .= "</ul>";
}

$stmt->close();
```

We still need to start `$output`, so put in an `else` clause to start it off with an empty value if it is not a list:

```
if ($tag=="li") {
    $output = "<ul class='users_menu'>";
} else {
    $output = "";
}
```

Save and refresh, click on a movie-goer's name, and that works, but the style is wrong for the heading. In the loop, change `<li>` to `<$tag>` to output the variable. Space the lines out to make it easier to read:

```
$output .= "<$tag>";
$output .= "<a href='index.php?user_id=$id'>$firstname $lastname</a>";
$output .= "</$tag>";
```

Everything works except that we haven't got the logout link we want. This should appear only when a `user id` is set, in other words when the variable `$data` is set to 'others' in the function.

So, inside the loop, add:

```
while ($stmt->fetch()) {
   ...
   if ($data == "others") {
   }
}
```

And in that case, we append to the output string:

```
if ($data == "others") {
      $output .= "<$tag class='logout'>";
      $output .= "<a href='index.php'>Logout</a>";
      $output .= "</$tag>" ;
}
```

Now the logout link appears but with no special styling to distinguish it from the names in the list.

The styles you need for this are in the Working Files for this chapter in a text file in the 'NEW STYLES' folder.

Open the text file and copy the styles into 'style.css' – if you start them on line 69 this will put them in the right place to be with the other related styles.

These give the logout list item a top border, uppercase text, and different colours for both the normal and hover states so as to set the logout link off visually from the other list items.

At present the heading styles are wrong. In 'navigation.inc.php', in the two places where we set $heading, cases 'no_id' and 'invalid_id', add `<h2>` and anchor tags surrounding the heading text:

navigation.inc.php:

```
$heading = "<h2><a>Choose a movie-goer</a></h2>";
```

Finally, in 'navigation.inc.php', add `$loggedState = "logged_out"` to cases 'invalid_id' and 'no_id' and `$loggedState = "logged_in"` to case 'id_set'.

Remove 'logged_out' under `class="profile logged_out"` and change it to:

```
<div class="profile <?php echo $loggedState; ?>"></div>
```

Save and refresh. This will activate the ‘logged_in’ and ‘logged_out’ styles which are already set in the stylesheet. Now the colour ‘logged in’ and the grey ‘logged out’ profile images should appear at the top of the navigation area.

That’s our user navigation done. In the next lesson we’ll do the first of a series of error-catching operations, the first one to output a message telling the user to select a movie-goer when the user first arrives at the website and no movie-goer is yet selected.

Lesson 69: Catching missing and invalid user ID

When the user first arrives at the site and has not yet selected one of the movie-goers, we don't want to show any of the movies; until a movie-goer is selected, we cannot know which movies are favourites. Instead we want to display a message at the top instructing the user to choose a movie-goer, and the footer and then we want to exit the program. We can deal with this easily using the codes returned by the `testUsers()` function to 'navigation.inc.php'.

Open up 'navigation.inc.php' and go down to the bottom. Create another section of PHP code.

Make an `if` condition to catch the eventuality that no `user_id` is set in the URL:

```
if ($testUsers == "no_id") {
}
```

And output a new HTML div with the class "message" containing a level 2 heading instructing the user to choose a movie-goer from the menu on the right:

```
if ($testUsers == "no_id") {
   echo "<div class='message'>";
   echo "<h2>Choose one of the movie-goers from the menu on the
         right</h2>";
   echo "</div>";
}
```

Then include the footer include file and exit the program so that no more of the website appears:

```
if ($testUsers == "no_id") {
   echo "<div class='message'>";
   echo "<h2>Choose one of the movie-goers from the menu on the
         right</h2>";
   echo "</div>";
   include 'footer.inc.php';
   exit;
}
```

We can do a very similar job to produce an alert message if an invalid user ID is entered, one that has no matching record in the database.

Copy the `if` statement we just wrote and paste it in below, changing 'no_id' to 'invalid_id'. We'll make this message more insistent by using the 'alert' class as well, which will give us red text:

```
if ($testUsers == "invalid_id") {
   echo "<div class='message alert'>";
   echo "<h2> Invalid user id: Choose one of the movie-goers from the
         menu on the right</h2>";
   echo "</div>";
   include 'footer.inc.php';
   exit;
}
```

Save and refresh ... and that works. If the text isn't red, open up 'style.css' and add the class style declaration `.alert { color: red; }`.

But there is a further way the user could mess things up. Try appending `index.php?user_id=3` and then some text garbage as part of the `user_id` value in the URL: `index.php?user_id=3gshshsh`; you will find that user 3 is still selected.

What is going on?

The program is removing the textual bit from `user_id` and proceeding with just the numerical part.

This is a normal feature not of PHP but of mySQL – it is called 'type conversion' and involves MySQL doing the best it can with the data it gets. If it sees a value which consists partly of numerical data it will try to use that.

We don't want this – if a value entered is garbage we want to tell the user it's garbage.

To prevent type conversion we can check before the variable ever gets to MySQL that it is in fact numeric data. We can do this with a PHP function `is_numeric()`. If `is_numeric()` encounters a value like this it will return FALSE.

Can you think how and where to do this before I tell you?

In 'test-users.fn.php', after the `if` statement where we check whether `user_id` has been set in the URL, add another `if` statement testing whether or not it is numeric:

test-users.fn.php

```
if (!isset($userID)) {
    return("no_id");
}

if (!is_numeric($userID)) {
    return ("invalid_id");
}
```

If it fails that test, it 'invalid_id' is returned and the function exits there.

Only if it passes that test do we look in the database, and check if we get any results with `numrows`.

Refresh and you should get the 'invalid ID' message.

That's finished the movie-goer navigation panel.

We've now covered all the code needed to make the two lists of movies we need – the favourites on the left, the non-favourites on the right.

We did the SQL queries for this back in the chapter on MySQL, the formatting is all there in the PHP files, and acting on user input in the URL we have just covered.

All that's needed is to put all this together.

The best way to learn to code is to do it yourself, taking risks, getting it wrong, and tying to sort out the problems.

So your first major assignment is to stop listening to me and try to write the code for the dynamic movie display yourself.

Instructions are in the next lesson and my answer follows that.

Assignment 1: Write the favourite movies display function

You now know all you need to write the code to display the formatted list of favourite movies for a given movie-goer.

Your task is to follow the same principles as before to write a single movie display function which we will later be able to build on until it is capable of displaying the favourites, the non-favourites and a selected single movie. You can adapt the code in 'show-users.fn.php' to do this as quite a lot of the code is the same.

If you need it, you will find the SQL query you need in a prepared statement in Working Files for this chapter, in the folder 'SQL'.

Name the function file 'show-movies.fn.php' and the function itself `showMovies()`. Call the function with the parameter `'favs'` in 'favourites.inc.php'. The function should build a formatted output string named `$output` and return this to 'favourites.inc.php'.

Good luck!

Lesson 71: Favourite movies display function

This is my solution to the favourite movies display function. I recommend that you use this version so that your code and mine are identical as we go on.

We'll save some time and maintain a predictable coding style by building the favourite movies display function from 'show-users.fn.php'. Copy 'show-users.fn.php' and rename the copy 'show-movies.fn.php'.

In 'index.php' add another `require_once` command to include 'show-movies.fn.php'. This is not needed in 'admin.php'.

In 'show-movies.fn.php', change the comment line – this is called from 'favourites.inc.php'.

Rename the function `showMovies()` and keep `$data` as its parameter:

```
function showMovies($data) {
}
```

The `case`s we need are 'favs' and 'non_favs'. For the moment we'll just do the favourites so just put `case favs:` for now:

```
function showMovies($data) {
    global $db, $userID;

    switch($data) {
        case "favs":
    }
}
```

We'll do the 'non_favs' case in the next lesson, delete the SQL query, the tag assignments, the 'if li' clause, and the output in the loop. Leave the loop – we can use that. Delete two `if` clauses after the loop. Now we have the bare bones ready to build the new function:

The first thing we need to do is to make `$movieID` available inside the function by making it global:

```
function showMovies($data) {
    global $db, $userID, $movieID;
...
```

The nested `SELECT` query we'll use to select a movie-goer's favourites is in the Working Files for this chapter, in the 'SQL' folder in 'select-favs.sql'. You can copy it from there or type it in.

It's been a while since we saw this so let's remind ourselves about this query and how it works.

Open up phpMyAdmin and click to get into the movies database. Click on 'SQL' to get into the SQL command line window.

We start out with a `SELECT` query:

```
SELECT movie_id FROM favourites WHERE favourites.user_id = 9
```

We have to specify that this is the `user_id` field in the 'favourites' table because there is also a user_id field in the 'movie_goers' table.

We'll use '9' as the ID number; this gives us the list of movie IDs favourited by that user. But as humans that list of numbers is of no use to us - we don't know what those movie IDs relate to, so we

have to get the matching list of titles and descriptions out from the MOVIES table.

We wrap that query in parentheses and use the list of movie ids it returns as the input to an outer query:

```
SELECT * FROM movies WHERE movie_id IN (
    SELECT movie_id FROM favourites
        WHERE favourites.user_id = 9
)
```

This uses the `IN` clause to select all the fields of those records in the movies table where the movie ID matches any of the list of values returned by the inner select query.

Finally we order the list of movies by their titles:

```
SELECT * FROM movies WHERE movie_id IN (
    SELECT movie_id FROM favourites
        WHERE favourites.user_id = 9
) ORDER BY title
```

Copy that and click 'Go' and now we've got the information we want.

Now we'll convert that into a prepared statement by replacing the value of `user_id` with a question mark:

```
function showMovies($data) {
    global $db, $userID, $movieID;

    switch($data) {
        case "favs":
          $stmt = $db->prepare("SELECT * FROM movies
                       WHERE movie_id IN (
                         SELECT movie_id FROM favourites
                           WHERE favourites.user_id = ?
                       )
                       ORDER BY title");
          break;
    }
}
```

And bind the variable `$userID` to the prepared statement as before:

```
...
break;
$stmt->bind_param('i', $userID);
...
```

After the switch clause bind the fieldnames in the 'movies' table to PHP variables:

```
$stmt->bind_result($id, $title, $description);
```

Again, we have to use `$id`, this time instead of `$movieID`, to avoid it overwriting the value of `movie_id` set in the URL.

Execute the prepared statement:

```
...
$stmt->bind_result($id, $title, $description);
$stmt->execute();
...
```

Start the output variable with the empty string:

```
...
$stmt->execute();
$output = "";
...
```

Inside the loop, change the variable names to `$title` and `$description`:

```
$title = htmlentities($title, ENT_QUOTES, "UTF-8");
$description = htmlentities($description, ENT_QUOTES, "UTF-8");
```

And then we'll build `$output` up as we did before.

We want to make a list item in `<li>` tags with a hyperlink to 'index.php':

```
$output .= "<li>";
$output .= "<a href = 'index.php'>$title</a>";
$output .= "</li>";
```

But we need to pass both the current movie ID and the selected user ID. We do this by joining the two URL variables with an ampersand, for which we have to use the HTML entity `&`:

```
$output .= "<li>";
$output .= "<a href =
            'index.php?user_id=$userID&amp;movie_id=$movieID'>$title</a>";
$output .= "</li>";
```

Now edit 'favourites.inc.php'.

At the top, call the `showMovies()` function with `'favs'` as its argument, and assign the result to a variable called `$favsList`:

```
<?php $favsList = showMovies('favs'); ?>
```

And in the HTML below, replace the three dummy `<li>` list items with

```
<?php echo $favsList; ?>
```

Save and refresh and you should get the formatted favourites list on the left.

Click on different users in the navigation menu and the favourites list on the left should change, demonstrating that the query works.

In the next lesson we'll do the very similar code, but more involved formatting, for the non-favourites list.

Assignment 2: Write the non-favourites movies display function

You can now write the rather similar code to display the non-favourite movies from the database as a list with thumbnail images in the main section of the page.

You will find the images in the Working Files for this lesson in the 'NEW MOVIE IMAGES' folder. Copy them to the 'images-movies' folder in 'favouritemovies'. Delete 'movie.png' and 'thumbnail.png' from the 'images-movies' folder; I have renamed these as 'generic.png' and 'generic-tn.png' as this is a better description.

The images are named numerically, the numbers matching the IDs of the movies. '–tn' means an image is a thumbnail, which we will be using for this list.

Add the case `"non_favs"` to the function file and adapt it to display the appropriate list of movies.

Then cut out the plain HTML from 'movie-list.inc.php' to build the variable `$output` in the function and return this to 'movie-list.inc.php' as a variable.

You'll need to catch the possibility of there being no matching movie or thumbnail image in the images directory and display the appropriate generic image instead. You can achieve this using the function `file_exists()`.

Good luck!

Lesson 73: The non-favourites movie display

I'll go through my idea of how to write the code for the main, non-favourites, list of movies and you can compare how I did it with your version. At the end of this lesson you should copy my version from the Working Files so that we have identical code to go on with, so as to avoid any confusion later on in the course.

First you should have copied the movie images and thumbnails from Working Files for this lesson to the 'images-movies' folder in 'favouritemovies' and deleted 'movie.png' and 'thumbnail.png' as these are now called 'generic.png' and 'generic-tn.png'.

In 'show-movies.fn.php' in the comment line telling us where the function is called, add 'movie-list.inc.php'.

Add `case "non_favs":` – everything is exactly the same as the 'favs' case except that we want `NOT IN` instead of `IN` in the prepared statement:

```
...
    case "non_favs":
       $stmt = $db->prepare("SELECT * FROM movies
               WHERE `movie_id` NOT IN (
                  SELECT `movie_id` FROM `favourites`
                   WHERE `favourites`.`user_id` = ?
               )
               ORDER BY `title`");
            $stmt->bind_param('i', $userID);
            break;
...
```

Inside the `while` loop, just after the two lines with `htmlentitites`, add `switch($data)` and the two `case` statements again.

In `case "favs"`, put the lines building the variable `$output` which we wrote in the previous lesson and then `break`:

```
...
    case "favs":
       $output .= "<li>";
       $output .= "
         <a href='index.php?user_id=$userID&amp;movie_id=$id'>$title</a>";
       $output .= "</li>";
       break;
...
```

In `case "non_favs"`, we need to build the output variable by converting the plain HTML currently in 'movie-list.inc.php' to PHP.

First, replace all the double quotes with single, replace all the dummy text with variables in the two links, remembering to use `$id` instead of `$movieID` throughout. Put in the 'alt' text – we want the movie title to appear if a user cannot see the images. Put in the image name, which is made up of the `$id` variable, the title, the description, and `break`. The completed code should be:

```
...
   case "non_favs":
      $output .= "<li>";
      $output .= "<figure>";
      $output .= "<a href='index.php?user_id=$userID&amp;movie_id=$id'>";
      $output .= "<img class='thumbnail' alt='$title'
```

```
                    src='images-movies/$id-tn.png'>";
        $output .= "</a>";
        $output .= "<figcaption>";
        $output .= "<h3>";
        $output .= "<a
          href='index.php?user_id=$userID&amp;movie_id=$id'>$title</a>";
        $output .= "</h3>";
        $output .= "<div class='description'>$description</div>";
        $output .= "<div class='add_remove favourite'></div>";
        $output .= "</figcaption>";
        $output .= "</figure>";
        $output .= "</li>";
        break;
...
```

In 'movie-list.inc.php', call the function with `showMovies('non_favs')` and assign the result to a variable, `$nonfavsList`:

movie-list.inc.php:

```
$nonfavsList = showMovies('non_favs');
```

Replace the three dummy list items with `<?php echo $nonfavsList; ?>`.

Save and refresh and if that has all gone well you should see the non-favourites list, formatted with the correct thumbnail images.

Each time you click on a different movie-goer in the menu, you should get a mutually exclusive list of movies – the favourites on the left, the non-favourites in the main list.

There's one more job to do, though. At present if no file exists with a name matching the ID number of a movie in the list, we will get a broken image, which is no good. We should catch the possibility of a missing or misnamed image and load the generic image file, 'generic-tn.png' instead if the matching image is not found.

You can write the code to do this using the PHP function `file_exists()`.

At the top of `case "favs"`, add:

```
if (file_exists("images-movies/$id-tn.png")) {
    $image = "images-movies/$id-tn.png";
} else {
    $image = "images-movies/generic-tn.png";
}
```

Lower down in the HTML, replace the image path:

```
$output .= "<img class='thumbnail' alt='$title'
             src='images-movies/$id-tn.png'>";
```

with the variable name `$image`:

```
$output .= "<img class='thumbnail' alt='$title' src='$image'>";
```

Save and refresh and you should see the generic image in place of the broken image we had before. Rename the image back to what it was and refresh and you should get the correct movie image.

In the next lesson we will output a personal greeting in the form of the selected user's full name at the top of this page.

Lesson 74: Insert personal greeting on movie list page

We can take this moment to insert the personal greeting at the top of the movie list page, using the `showUsers()` function we have already written but with a new parameter.

Open 'show-users.fn.php' and under 'cases', add another `case`, called `"get_name"`, which we want to get the current user's details, so the prepared statement is the same as for 'current', and the output will be a 2nd level heading so we want `<h2>` as the tag:

```
case "get_name":
    $stmt = $db->prepare("SELECT * FROM `movie_goers`
                          WHERE `user_id` = ?");
    $stmt->bind_param('i', $userID);
    $tag = "h2";
    break;
```

Then in the `while` loop add an `if` clause which runs if `$data` is equal to 'get_name' and sets `$output` to a string within `<h2>` tags with the message "Hi" and the name of the selected movie-goer:

```
if ($data=="get_name") {
    $output .= "<$tag>";
    $output .= "Hi, $firstname $lastname";
    $output .= "</$tag>";
}
```

And in the `else` clause, put our original output formatted as a list:

```
...
else {
    $output .= "<$tag>";
    $output .= "<a href='index.php?user_id=$id'>$firstname $lastname</a>";
    $output .= "</$tag>";
}
```

Then in 'movie-list.inc.php', call `showUsers()` with the argument 'get_name' and assign the result to the variable `$fullname`:

movie-list.inc.php:

```
$fullname = showUsers('get_name');
```

and echo `$fullname` out in place of the dummy greeting and its tags:

```
<?php echo $nonfavsList; ?>
```

Save and refresh, and we have our personal greeting.

Add in the comment in 'show-users.fn.php' that the function is now also called in 'movie-list.inc.php'.

In the next lesson we'll adapt existing code again to produce the formatted single movie page.

Lesson 75: *The single movie display*

For the single movie display, we'll continue to use the `showMovies()` function, giving it a new parameter and a new case in the `switch` clause, called `'single'`.

This follows exactly the same principle as we've done before so if you want to save typing all those `$output .=` lines then you can copy the files from the Working Files for this chapter and go on.

If you want to type yourself, have a go first without watching this movie so as to reinforce what we've been doing.

I'll go quickly though this, using the finished files.

'Movie-single.inc.php' now calls `showMovies()` with the new parameter, 'single' `showMovies('single')` and assigns the result to the variable `$singleMovie`.

'Show-movies.inc.php' has a new `case`, `'single'`, in which the prepared statement gets the one movie matching the selected movie ID, so as to display the movie specified in the URL.

As before, we test for the existence of the movie image and substitute the generic image if it is missing.

The plain HTML is converted to PHP as before, building `$output` line by line.

I have also added a new style to 'style.css': `margin-top: 15px` to the `.movie_list` and `.movie_single` styles, so copy this or add this style yourself to give a little bit of space from the header which I missed.

Save and refresh and that works fine, but we need to do more work on it to guard against missing and invalid user input for `movie_id`, just as we did for `user_id`.

Lesson 76: Catch missing and invalid movie ID

So far we haven't taken any steps to catch errors in the value of `movie_id`, which the user enters into the URL. Let's see what happens if we deliberately enter a non-existent or non-numerical value as `movie_id`.

In the first case we get a messy empty movie list page and in the second MySQL's type conversion kicks in and ignores the textual part of `movie_id`, neither of which solutions we want.

We need to do the same checks on the `movie_id` input as we did on `user_id` and display a similar alert message if the tests are failed.

The first thing is that we're getting an unnecessary number of very small include files now, which are breaking up our code too much, and this causes a problem now we've come to the task of catching invalid input on the `movie_id`.

We can do all our movie displays better in just one include file now. Let's call this 'movies.inc.php'.

In 'index.php', remove all the `require_once` lines including favourites, movie-list and movie-single and the `if` clause, and replace them with just the one line: `require_once 'movies.inc.php';`.

We'll do the decision about which page content to display in the function.

Now we'll combine 'favourites.inc.php', 'movie-list.inc.php' and 'movie-single.inc.php' in this new file.

Make this new file, 'movies.inc.php', and save it in the 'php-includes' folder.

Cut everything out of 'favourites.inc.php' and paste it into 'movies.inc.php'.

Add `$nonfavsList = showMovies('non_favs');` in the PHP section at the top, and end the PHP section.

Leave the favourites display as it is, and after it we'll start a new PHP section and make the decision here about whether to display the movie list or single movie, based on whether `movie_id` is set in the URL:

```
if (!isset ($movieID)) {
    // show movie list
} else {
    // show single movie
}
```

In the movie list `if` clause, convert all the HTML from 'movie-list.inc.php' to `echo` statements:

```
if (!isset ($movieID)) {
    echo "<section class='movie_list'>";
    echo $greeting;
    echo "<p class='welcome'>Here are some movies you might like.";
    echo  "Click  on  the  heart  icon  to  add  them  to  your  favourites
list.</p>";
    echo "<ul>";
    echo $nonfavsList;
    echo "</ul>";
    echo "</section>";
}
```

and do the same for the HTML from ‘movie-single.inc.php’ in the `else` clause:

```
...
} else {
    echo "<section class='movie_single'>";
    echo $singleMovie;
    echo "</section>";

}
```

We can perform the validation on the `movie_id` input in exactly the same way as we did for `user_id`.

Copy ‘test-users.fn.php’ and rename the copy ‘test-movies.fn.php’. Change the comment line. Open ‘test-movies.fn.php’ and change every occurrence of ‘user’ to ‘movie’ and ‘users’ to ‘movies’.

The prepared statement we use to check if the movie ID is in the database should read:

```
SELECT * FROM movies WHERE movie_id = ?
```

Call ‘test-movies.inc.php’ at the top of ‘movies.inc.php’, assigning the result to the variable `$testMovies`:

```
$testMovies = testMovies();
```

Then set up the switch clause and cases, using $testMovies as the parameter:

```
switch($testMovies) { }
```

And we’ll have three cases, ‘invalid_id’, ‘no_id’ and ‘id_set’, just as before.

In the case ‘invalid_id’, we’ll echo an alert message, include the footer, and exit:

```
...
case "invalid_id":
    echo "<div class='message alert'>";
    echo "<h2>Invalid movie ID: Choose a movie-goer from the menu
          on the right</h2>";
    echo "</div>";
    include 'footer.inc.php';
    exit;
    break;
```

In case ‘no_id’ we have no movie selected so we’re going to display the movie list. So we get the name for the greeting by assigning the result of `showUsers('get_name')` to the variable `$greeting`, and `break`:

```
...
case "no_id":
    $greeting = showUsers('get_name');
    break;
```

In case ‘id_set’, we’ll get the formatted single movie output by assigning the result of `showMovies('single')` to `$singleMovie`, and break:

```
...
case "id_set":
    $singleMovie = showMovies('single');
    break;
```

This is very much the same as we did before with the users input. The main difference is that we put the alert message before the main output instead of after it. This is because in the navigation panel we wanted the users list to appear even when the alert message was triggered, so that site navigation would always be possible, so the navigation went before the alert and exit.

In this case we don't want the movies to display if there's any error in the user input, so we put the alert and exit before the code for the movie display.

In 'index.php', add `require_once 'test-movies.fn.php'` above 'show-movies.fn.php'.

Save and refresh, try putting in a movie ID which we know doesn't exist in the database and then try putting in a non-numerical movie ID, and we should get our alert messages.

Now we can delete those old include files, 'favourites.inc.php', 'movie-list.inc.php', and 'movie-single.inc.php'.

We've got one more piece of error catching to do – to ensure that the system does not fail if the database tables are empty, either the movie-goers table or the movies table. We'll deal with that in the next lesson.

Lesson 77: Catching empty movie-goers table

Our error catching for invalid and non-numerical user input works fine, but what would happen if the database contained no data? Let's find out.

In phpMyAdmin, go to the 'movie-goers' table. Select and delete all the movie-goer records.

Then go back and refresh the webpage. We've got a user ID selected in the URL at the moment, and that's caught by our previous error-catching methods. Delete the user ID from the URL so that it ends with `index.php`. Now we get the standard message telling us to select a movie-goer from the right, but over on the right we get an ugly little bit of non-existent list – not nice.

Reinitialise the database by importing the original data again, and then delete all the records in the 'movies' table.

Refresh the page in the browser, and now we get the movie list page but with no movies listed and a message telling us we might like some of them. That's not very good either.

Let's deal with this in two steps, tacking the movie-goers first.

First, where and how do we add a new catch to test for no data in the movie-goers table? Try to do it yourself before looking at my answer!

In 'test-users.fn.php', put a new SQL query, selecting all the records from the 'movie_goers' table:

```
$sql = "SELECT * FROM `movie_goers`"
```

There is no need for a prepared statement here as there is no user input to sanitise and plain SQL is faster and saves resources.

Then get the result:

```
$result = $db->query($sql);
```

and then the number of records returned, using the `num_rows` method:

```
$num_rows = $result->num_rows;
```

and then, if the value of `$num_rows` is less than 1, return the code 'no_data' to the calling file, 'navigation.inc.php':

```
if ($num_rows < 1) {
     return ("no_data");
}
```

In 'navigation.inc.php', add the matching `case`:

```
case "no_data":
```

Assign `<h2>` tags to `$heading`, the navigation heading on the right, and make a link not to one of the movie-goers' pages but to the admin page, set to load the manage users table:

```
case "no_data":
    $heading = "<h2><a href='admin.php?page=users'></a></h2>";
```

And give it a new title, so the website shows it is aware of the problem:

```
$heading = "<h2><a href='admin.php?page=users'>Add movie-goers</a></h2>";
```

Set `$usersList` to an empty string:

```
$usersList = "";
```

This will remove the ugly bit of non-existent list.

Set `$loggedState = "logged_out"` to display the logged out profile image, and break.

Then at the bottom of the page add another `if` clause and inside it echo a similar alert message, but this time include the 'admin-users' include file:

```
if ($testUsers=="no_data") {
    echo "<div class='message alert'>";
    echo "<h2>No movie-goers in database: Add movie-goers below</h2>";
    echo "</div>";
    include 'admin-users.inc.php';
}
```

Then include 'footer.inc.php' and exit:

```
...
    include 'admin-users.inc.php';
    include 'footer.inc.php';
    exit;
}
```

Repeat the re-initialisation of the database and again delete all the movie-goers from the table so that we can test this.

Save and refresh and you should get the admin table on the front page so new records can be conveniently added.

But the table heading appears twice, once in red and once in black.

How can you get rid of the second heading?

In 'admin-users.inc.php', in a new PHP block, type:

```
if ($testUsers != "no_data") {
    echo "<h2>Manage users</h2>";
}
```

so the normal black header will appear only if the admin page is loaded directly, when `$testUsers` will not equal 'no_data'.

Now that 'movies.inc.php' also displays the admin table when needed, its name is no longer accurate so rename it to 'main.inc.php'. Find all other places where the name exists using 'Find in Files' in Komodo Edit, check them and replace them with the new name.

Reinitialise the database.

As your next assignment, I'd like you to do the same for the movies table.

Assignment 3: Catching empty movies table

Follow the same principles as we have just used to catch the possibility of an empty movies table and display an alert message and the admin-movies page below.

Take appropriate action to deal with any variables which are not set and produce an error message.

Lesson 79: Catching empty movies table

Start by deleting all the records from the 'movies' table so that we can run our checks on this.

The process for catching an empty movies table is almost exactly the same so I'll speed things up by working with the completed files and highlighting the changes made. If you did this as an assignment you can check that you have made the same changes, or you can copy the files from the Working Files.

In 'test-movies.fn.php' we have exactly the same SQL select ending in a code of 'no_data' if no results are found:

```
$sql = "SELECT * FROM `movies`";
$result = $db->query($sql);
$num_rows = $result->num_rows;
if ($num_rows < 1) {
    return ("no_data");
}
```

In 'main.inc.php', we need to stop the favourites being displayed if there's no movie data, and return an alert message instead, so we have an `if` clause before we get to the favourites part of the page, which displays an alert message if the value of `$testMovies` is 'no_data', includes the 'admin-movies' include file and the footer, and then exits:

```
if ($testMovies == "no_data") {
    echo "<div class='message alert'>";
    echo "<h2>No movies in database: Add movies below</h2>";
    echo "</div>";
    include 'admin-movies.inc.php';
    include 'footer.inc.php';
    exit;
}
```

Only if the value of `$testMovies` is not 'no_data', do we display the favourites, by putting this section in the `else` clause, converted to `echo` statements:

```
...
else {
    echo "<nav class='favs_list'>";
    echo "<h2>Favourites</h2>";
    echo "<ul class='favs'>";
    echo $favsList;
    echo "</ul>";
    echo "<div class='trash'></div>";
    echo "</nav>";
}
```

As before, if it was left like that we would get the double heading for the table so in 'admin-movies.inc.php' we need to put our normal `<h2>` heading for the movies admin table inside an `if` cause.

This time our test variable may not have been set, so we need to do a double check in 'admin-users.inc.php', to display the normal, non-alert table heading if either $testMovies has not been set or if its value is not 'no_data':

```
if (!isset($testMovies) || $testMovies!="no_data") {
    echo "<h2>Manage movies</h2>";
}
```

Save and refresh and the movies table should appear when there is no data.

We have now completely obviated the need for ‘admin.php’ – ‘index.php’ does various different jobs according to the data it receives – so I have changed the links in ‘navigation.inc.php’ from ‘admin.php’ to ‘index.php’:

```
<li><a href="index.php?page=users">Manage users</a></li>
<li><a href="index.php?page=movies">Manage movies</a></li>
```

Delete ‘admin.php’.

Finish this lesson by re-initialising the database.

In the next lesson we will output different text in the favourites and movie-list headings according to whether a user has either chosen all the movies as favourites or has no favourites at all, or has some movies as favourites and some not.

Lesson 80: Data-dependent title for favourites list

At moment even if we select a movie-goer who has no favourites (*e.g.* Alfred the Awful) or has all the movies as favourites (*e.g.* Mr Movieman), we still get the same titles at the top of the favourites and the movie list.

It would be better to have the title of 'Favourites' change to 'You have no favourites' and have the trash can disappear if there are no favourites, and if all the movies are favourites have the top of the movie list change to show that there is nothing to choose from below.

Let's do this now.

First of all, the code in 'main.inc.php' is a bit messy, with a mixture of `switch` and `if` clauses – let's tidy it up by using `switch` instead of `if` throughout.

Move the `echo` statements from inside the `$testMovies == "no_data"` clause to a new case, `'no_data'`:

```
switch($testMovies) {
    ...
    case "no_data":
       echo "<div class='message alert'>";
       echo "<h2>No movies in database: Add movies below</h2>";
       echo "</div>";
       include 'admin-movies.inc.php';
       include 'footer.inc.php';
       exit;
```

Remove the `else` clause surrounding the favourites output.

And convert the 'if $movie_id is set' clauses to `switch ($testMovies)` with cases 'no_id' and 'id_set':

```
switch($testMovies) {
    case "no_id":
        // non favs list output
        break;

    case "id_set":
        // single movie output
        break;
}
```

Test in the browser.

That's more in keeping with the coding style elsewhere.

Put in a new `if` clause just after the function calls:

```
if ($favsList == "") {
}
```

When there are no favourites the variable `$favsList` will be empty, so we can use this to set up alternative titles and styles.

So when `$favsList` is empty, set a new variable, `$favsTitle`, to 'You have no favourites':

```
if ($favsList == "") {
    $favsTitle = "You have no favourites";
}
```

and set a new variable for a class, `$trashclass`, which we will use to apply a hidden style to the trashcan:

```
if ($favsList == "") {
    $favsTitle = "You have no favourites";
    $trashclass = "hidden";
}
```

In style.css, add a new class, `.hidden` and give it the display setting 'none':

style.css:

```
.hidden  {
    display: none;
}
```

And add the variable `$trashclass` as a second style for the trash can:

```
echo "<div class='trash $trashClass'></div>";
```

Then if `$favsList` is <u>not</u> empty, set `$favsTitle = "Favourites"` and set `$trashclass` to empty so the trash can still appears:

```
...
else {
    $favsTitle = "Favourites";
    $trashClass = "";
}
```

Save and refresh and select "Alfred the Awful" and you should get the new 'No favourites' title and no trash can.

Now for an exercise see if you can do the same for the movies list title, replacing the current welcome message with 'It looks like you love all the movies! Drag them to the trash can to delete them from your favourites' and dealing with any styling issues which arise.

Lesson 81: Data-dependent welcome in movie list display

Now for the welcome message to appear above the non-favourites list, we follow exactly the same procedure, this time setting up a custom title if `$nonfavsList`, the movie list, is empty.

In 'main.inc.php', find the movie list title area and type:

```
if ($nonfavsList=="") {
}
```

Again, `$nonfavsList` will be empty if there are no non-favourite movies.

Set up a custom welcome message:

```
if ($nonfavsList=="") {
    $welcome = "It looks like you love all the movies! ";
    $welcome .= "Drag them to the trash can to delete them from
    your favourites.";
}
```

and our standard welcome message in the `else` clause:

```
...
else {
    $welcome = "Here are some movies you might like. ";
    $welcome .= "Click on the heart icon to add them to your favourites
    list.";
}
```

and replace the message text further down in the file with the variable `$welcome`:

```
echo "<p class='welcome'>$welcome";
```

Save and refresh and the message is OK but we get an empty section below it. This is appearing because we've got opening and closing `ul` tags with nothing in between them - we should make these display only when there's some content in to go in the list.

Set the variables `$openTag` and `$closeTag` to empty if `$nonfavsList` has no content, so they do not appear, and to `<ul>` and `</ul>` respectively if `$nonfavsList` has content, so they do:

```
if ($nonfavsList=="") {
    // welcome message as above
    $openTag = "";
    $closeTag = "";
} else {
    // welcome message as above
    $openTag = "<ul>";
    $closeTag = "</ul>";
}
```

In the HTML for the non-favourites list, replace the hard-coded tags with `$openTag` and `$closeTag`:

```
echo $openTag;
echo $nonfavsList;
echo $closeTag;
```

Save and refresh – that removes the empty `<ul>`.

But it gives us another problem – the border is doubled now as we have a bottom border on the welcome paragraph and another on the surrounding box.

So when `$nonfavsList` is empty, assign to a new variable, `$welcomeClass` the class name `no_border_bottom`:

```
if ($nonfavsList=="") {
    // as above
    $welcomeClass = "no_border_bottom";
}
```

And in 'style.css' add the class style `.no_border_bottom` and style it with `border-bottom: none`:

```
.no_border_bottom {
    border-bottom: none;
}
```

If `$nonfavsList` has no content, set `$welcomeClass` to `no_border_bottom` , and if `$nonfavsList` has content, set `$welcomeClass` to empty so the original style is unaffected:

```
if ($nonfavsList=="") {
    // as above
    $welcomeClass = "no_border_bottom";
} else {
    // as above
    $welcomeClass = "";
}
```

Below, add `$welcomeClass` as a new style to the welcome paragraph:

```
echo "<p class='welcome $welcomeClass'>$welcome";
```

Save and refresh and that's sorted out.

Our next task is the slightly more difficult job of changing the dummy 'Add to/delete from favourites' link to 'Add to favourites' if the selected movie is not currently a favourite and 'Remove from favourites' if it is.

Lesson 82: Data-dependent link on single movie page

To get the 'Add to/remove from favourites' link on the single movie page to display the appropriate message according to whether the selected message is already a favourite or not, we need a new function, because as yet we don't have that information available. Make a new function file and name it 'test-fav.fn.php' and add it to the list of included functions in 'index.php'.

We want this function to return a string 'Add to favourites' if the currently selected movie is not a favourite and 'Remove from favourites' if it is. Try to write all the code to do this before I give you

First we need to have access to `$db`, `$userID` and `$movieID`:

test-fav.fn.php:

```
function testFav() {
    global $db, $userID, $movieID;
}
```

Then our prepared statement looks in the favourites table to see if there is any record which matches both the `user id` and the `movie id`. If there is, the current movie is a favourite; if not then it isn't:

```
$stmt = $db->prepare("SELECT * FROM favourites WHERE user_id = ? AND
movie_id = ?");
```

Both the variables are integers so the `bind_param` method needs `'ii'` as its data type setting:

```
$stmt->bind_param('ii', $userID, $movieID);
```

Then we execute the prepared statement:

```
$stmt->execute();
```

To get the results we have to use store_result:

```
$stmt->store_result();
```

Then to see if there are any results we use the method `num_rows` to count the number of records returned:

```
$numrows = $stmt->num_rows;
```

If numrows is zero there are no results and the movie is <u>not</u> a favourite; the only other possible result is 1, in which case it <u>is</u> a favourite.

Now we're finished with the prepared statement and can close it:

```
$stmt->close();
```

Then with a simple 'if' clause we can send back the appropriate string to the calling script:

```
if ($numrows<1) {
    $output = "Add to favourites";
} else {
    $output = "Remove from favourites";
}
return($output);
```

But where shall we call it and how can we make the new variable available where it is needed, in the `showMovies()` function?

We'll make sure we call it once only by putting the variable assignment along with the others at the top of 'main.inc.php' and in keeping with previous practice we'll use the function name for the variable name:

main.inc.php:

```
...
$testMovies = testMovies();
$testFav = testFav();
```

Then we need access to `$testFav` in the `showMovies()` function so add it to the list of global variables:

show-movies.fn.php:

```
function showMovies($data) {
    global $db, $userID, $movieID, $testFav;
}
```

And down at in the output section under `case single`, replace the dummy text with `$testFav`:

show-movies.fn.php:

```
case "single":
    // as before
    $output .= "<p>$testFav</p>";
...
```

Save and refresh and select favourite and non-favourite movies and you should get the appropriate link.

Our next task is to replace the current dummy data in the movies and movie-goers' admin tables with dynamic data from the database. I think this would make a nice assignment for you, reinforcing what we've been doing. Instructions are in the next lesson.

Assignment 4: Putting dynamic data into the movie admin table

Following the same principles as we have used so far, write the code to display the full list of movie titles and descriptions in the table in 'admin-movies.inc.php'.

No new functions are needed; just add a new `case` to each of the existing functions.

Lesson 84: Dynamic data in movies admin table

We can follow exactly the same principles as we used earlier to display the database records in the admin tables, using the two existing functions, `showMovies()` and `showUsers()`.

Let's do the movies page first.

Open up 'show-movies.fn.php' and add another `case`, called 'admin':

show-movies.fn.php:

```
switch($data) {
    // as before
    case "admin":
    ...
```

We want to display all the database records so the prepared statement is `"SELECT * FROM movies"`:

```
$stmt = $db->prepare("SELECT * FROM movies");
```

There are no parameters to bind to that so we go straight on to `break`.

The output variable should be a complete table row to slot into position in 'admin-movies.inc.php'.

Open 'admin-movies.inc.php' and cut out one of the dummy data rows, including the opening and closing `<tr>` tags. Delete the other dummy data row.

In show-movies.fn.php, add `break` to `case single`.

Paste the raw HTML you have copied into 'show-movies.fn.php', add a new `case`, called `admin`, and inside this convert it into the concatenated variable assignment as before, using `$output .=`. End with `break`:

show-movies.fn.php:

```
case "admin":
    $output .= "<tr class='datarow'>";
    $output .= "<td><input class='data' type='text' name='title'
                value='$title'></td>";
    $output .= "<td><input class='data' type='text'
                name='description' value='$description'></td>";
    $output .= "<td class='deletecell'><div class='delete'></div></td>";
    $output .= "</tr>";
    break;
```

On the second `<td>` add `description` as well as `data` to the class name so we can style the description column to be wider:

```
    $output .= "<td><input class='data description' type='text'
                name='description' value='$description'></td>";
```

Back in 'admin-movies.inc.php', add at the top:

```
<?php $dataRow = showMovies('admin'); ?>
```

and

```
<?php echo $dataRow; ?>
```

in place of the deleted table row.

Add a new parent-child selector, `.admin_table .description`, to 'style.css':

```
.admin_table .description {
   width: 700px;
}
```

to make the description column wider.

Save and refresh and the dynamic data is displayed.

To do the users admin table is a bit more involved because the logic of the `showUsers()` function isn't quite right, so I'll go through that in the next lesson.

Lesson 85: Dynamic data in users admin table

This is exactly the same principle as the previous lesson, except that the function `showUsers()` isn't as well organised as it might be. Let's improve it by converting `if` clauses to `switch`es.

First, add the new `case`, `'admin'`, and again the prepared statement needs to select all data. There are no variables to bind to it. The HTML tag should be set to empty, and we end with `break`:

show-users.fn.php:

```
case "admin":
   $stmt = $db->prepare("SELECT * FROM `movie_goers`");
   $tag="";
   break;
```

Then in the output part of the function, we can simplify things by getting rid of the `if/else` clause and using `switch/case` instead.

We need four cases, 'get_name', 'all', 'current', and 'others', but the last three all produce the same list output, so we can combine all three in one case using a semi-colon as the separator:

```
switch($data) {
    case "get_name":

    case "all"; case "current"; case "others":

}
```

Put the relevant output in the appropriate `case`s and add `break`s:

```
switch($data) {
    case "get_name":
        $output .= "<$tag>";
        $output .= "Hi, $firstname $lastname";
        $output .= "</$tag>";
        break;

    case "all"; case "current"; case "others":
        $output .= "<$tag>";
        $output .= "<a href='index.php?user_id=$id'>$firstname $lastname
                    </a>";
        $output .= "</$tag>";
        break;

}
```

Straight after the closing brace for the switch clause comes the closing brace of the `while` loop.

Make a new `case`, `'admin'`, and as before, cut out the dummy data row from 'admin-users.inc.php' and convert it to an output variable using `$output .=` again. End with `break`:

```
case "admin":
    $output .= "<tr class='datarow'>";
    $output .= "<td><input class='data' type='text' name='firstname'
                value='$firstname'></td>";
    $output .= "<td><input class='data' type='text' name='lastname'
                value='$lastname'></td>";
    $output .= "<td class='deletecell'><div class='success'></div></td>";
    $output .= "</tr>";
    break;
```

At the top of ‘admin-users.inc.php’, run the function:

```
<?php $dataRow = showUsers('admin'); ?>
```

Delete the dummy data row and replace it with the function output:

```
<?php echo $dataRow; ?>
```

Save and refresh and you should get the dynamic data in the table.

Change `<div class="success">` to `<div class="delete">`.

Make sure the comments in both ‘show-users.fn.php’ and ‘show-movies.fn.php’ are correct and tell us which files they are called in:

show-users.fn.php:

```
// called in navigation.inc.php and main.inc.php
```

show-movies.fn.php:

```
// called in main.inc.php
```

Everything’s working fine but before we go on we’ll check the website again for cross-browser compatibility and validate the HTML source.

Lesson 86: Cross-browser compatibility check

Check all the pages in all the major browsers and verify that there are no problems.

In Internet Explorer 10, you can press F12 and try it in various previous browser modes back to IE8. This no longer works once you have upgraded to IE11, but coping with old versions of Internet Explorer is becoming less important as time goes on.

Lesson 87: Validate HTML

Go to the W3C Markup Validation Service at validator.w3.org and choose ‘Validate by direct input’.

Right click in the various pages and choose ‘View page source’. Select everything and paste this in. Press ‘Check’ and we should get ‘This document was successfully checked as HTML5’.

Do the same for all the other pages and click ‘Revalidate’ and then check one of the error pages.

All OK – next we will format the HTML source to make it readable.

Lesson 88: Format HTML source

You can either follow the rather laborious process of formatting the HTML by hand as I do in the video, or you can try either of the automatic formatting methods suggested by one of my students: at validator.w3.org, jigsaw.w3.org/css-validator, and beta.phpformatter.com for PHP.

Now for the manual method.

I've left formatting the HTML until the end so that we know we are working on the final version of the code because as you will see it adds a load of verbiage to the code, which we wouldn't want while we were still working on it.

Our task is to go through the whole project inserting tab characters and new lines in the HTML and PHP output.

In PHP we use `\t` for tab and `\n` for new lines, and we'll work with the HTML source window open, constantly refreshing it, until it's neatly indented throughout.

It's an understatement to say that this is a somewhat lengthy and unexciting job so I'll demonstrate it for the whole of the movie list page and then leave you to do the rest if you really want to or, if you prefer, you can copy the files with formatting done from the last lesson in this chapter of the Working Files.

Let's start with the movie-list page, looking at the indentation of the source.

Select a movie-goer with both favourites and non-favourites, *e.g.* Wallace. It's OK down to the end of the head section. Then we need a new line before the body tag.

Open up 'head.inc.php' and enter two new lines at the end and remove the indentation from the last one. Refresh the source – that's dealt with that.

Put one new line at the end of 'header.inc.php' and remove the indentation.

The header is OK now – the navigation needs to be indented two tabs, throughout.

We could simply indent the whole of the block of plain HTML two tabs, but it would be rather odd in the code, so I'm going to rewrite that block as PHP.

Take out the closing PHP tag above it and follow the normal procedure for converting HTML to PHP echo statements. When that's done, refresh the source and you will find that all the formatting is gone and it's all one line, because PHP doesn't put in tabs or new lines unless we tell it to.

Now we need to indent the opening nav element two tabs – enter `\t\t` inside the double quotes. Save and refresh.

Then `\n` at the end of the opening `<nav>` element, as always inside the double quotes.

As we go on we need to insert tabs as needed to get the indentation right – three tabs `\t\t\t` before `<div class = "select users">`, and a new line `\n` at the end

For tabs `\t\t\t\t` before `$heading`, new line `\n` at the end

Three tabs before the closing `</div>` to match the opening one. Two new lines at the end.

Three tabs `\t\t\t` before `<div class = "profile">`, and before `<div class = "admin button">`, and a new line after each – let's have two after 'admin button'.

Three tabs before the opening `<ul>` and a new line at the end.

Four tabs before the opening `<li>` and a new line at the end. And the same for the next `<li>`.

Three tabs for the closing `<ul>` and a new line.

And two for the closing `</nav>` element, and two new lines.

But the users list is all on one line still – this is generated by 'show-users.fn.php', so open that up.

We need a new line after the opening `<ul class = "users_menu">` tag.

Save and refresh.

And then the `<li>` list items need to be indented four tabs, and we need a new line at the end of the closing list tag `</li>`.

Save and refresh.

The logout list item also needs four tabs and a new line at the end of the closing `</li>` tag.

Save and refresh .

And the closing `</ul>` tag needs three tabs and a new line at the end.

Now we're down to the favourites list so open up 'main.inc.php'.

Find `<nav class="favs_list">` and add two tabs to the start of it and a new line at the end.

Add three tabs to the start of `<h2>` and two new lines at the end.

Add three tabs to the start of `<ul class="favs">` and a new line at the end.

In 'show movies.fn.php' in the 'favs' case, add four tabs to the start of the `<li>` tag and a new line at the end.

Back in 'main.inc.php', add three tabs to the closing `</ul>` tag and two new lines at the end.

Add three tabs to the start of `<div class = "trash">` and a new line after the closing `</div>`.

Add two tabs to the start of the closing `</nav>` tag and two new lines at the end.

Add two tabs to the start of the opening `<section = "movie_list">` tag and a new line at the end.

Save and refresh.

In 'show-users.fn.php' in the 'get_name' case add three tabs to the start of the opening tag and a new line after the closing tag.

In 'main.inc.php' find `<p class = "welcome">` and put three tabs in front of it and two new lines at the end, putting in the missing closing `</p>` tag I have just noticed before the new line.

In ‘main.inc.php’, find `$openTag`, surround it in double quotes and inside the double quotes put three tabs at the start and a new line at the end. This is the opening `<ul>` tag for the movies list. Save and refresh.

In ‘show-movies.fn.php’, find the opening `<li>` tag in the ‘non_favs’ case and put four tabs in front of it and a new line at the end. Save and refresh.

And five tabs in front of the opening `<figure>` tag and a new line at the end. Save and refresh.

Put six tabs in front of the opening anchor tag and a new line after the closing anchor tag.

Put six tabs again in front of the opening `<figcaption>` tab and a new line straight after it.

Put seven tabs in front of the opening `<h3>` tag and a new line after the closing `</h3>` tag.

Put seven tabs in front of the opening `<div class = "description">` and a new line at the end.

Put seven tabs in front of the ‘Add to/remove from favourites’ div and a new line at the end.

Now we start unindenting.

Put six tabs in front of the closing `</figcaption>` tag and a new line at the end, five in front of the closing `</figure>` tag and a new line at the end, four in front of the closing `</li>` tag and a new line at the end.

In ‘main.inc.php’ find `echo $closeTag` and add double quotes and three tabs in front of it and a new line after it.

Add two tabs before and two new lines after the closing `</section>` tag. Save and refresh.

There – look at that beautiful HTML!

Now I think you are very lucky to have me here because I have performed this thankless task throughout all the other files.

So you can either do all that yourself if you like that sort of thing or you can copy the Working Files from this chapter and overwrite your files.

Now we’re finished with this chapter, we’ve got our dynamic data and our nicely formatted HTML and we’re ready to start the real fun – jQuery.

Chapter 10: Essential jQuery and AJAX

Lesson 89: Introduction to jQuery

If you've followed this course from the beginning, you've heard me talk a lot about jQuery and the wonderful things it can do; so what is jQuery?

jQuery is a single Javascript file consisting of ready-made functions designed to make it easy to manipulate an HTML document on the fly, meaning we can easily change styles, make things disappear or reappear, move around, fade them in and out, and create a host of other effects.

It's an extension of Javascript specially designed to work in conjunction with styles, such as those we set up in our stylesheet, 'style.css'.

The jQuery motto is "write less, do more", and it certainly achieves this goal – you can do more in one line of jQuery than in a hundred or more of Javascript written from scratch.

Before we can begin to see how it works, we need to do two things – make jQuery available in our pages, and install Firebug, the developer tools for Firefox, if you haven't already got it.

There are two ways of making jQuery available – you can either download it to your computer and link to it, using `<script src = path/to/script>` in the `<head>` part of an HTML file as we did for the html print shiv script earlier, or you can link to a version hosted online at Google, Microsoft or jquery.com.

These online hosts are called CDNs or content distribution networks, and unless you have a specific reason not to, which I'll come to in a moment, it is better to link to a hosted version than to download your own copy.

The reasons for this are threefold:

- first, CDN servers are located all around the globe so when a user loads up your site, jQuery will probably come from a server physically much closer than your web host, meaning faster page loading;
- second, having an external server provide jQuery means your own server has less work to do, which also speeds up loading times;
- third, and most importantly, you have probably already visited other websites using jQuery from the same CDN, in which case jQuery is already cached on your computer so your browser won't even need to load it again.

These benefits are lost if you use a local copy.

There is just one disadvantage of using a CDN – it obviously won't work if the site you are developing will run on a machine which has no access to the internet.

If you need to develop an offline site, go to the Downloading jQuery page, and choose the current compressed, production version, save it in a folder called 'scripts', and then link to it using `<script src = "scripts/name-of-script.js">`.

For our purposes, its best to use Google's CDN as this is the most popular one. I have put a link in the bookmarks to Google hosted libraries, https://developers.google.com/speed/libraries/devguide, where you will find jQuery and other links we'll need later.

Elsewhere, you can also find links to the “latest” version of jQuery which might seem like a good idea to make sure you’re up to date. But this a changing version, being continuously developed, so it’s safer to decide on a specific version and develop your site in that and stick to it because otherwise changes made to the latest version of jQuery may break the site.

We link to jQuery using the script tag:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js">
</script>
```

‘Min’ in the file name means this is the minimised version, with all comments, newlines and formatting stripped out. It is completely unreadable for humans but its file size is considerably smaller than the non-minimised version so you should use that.

Now jQuery is available for use in our pages.

In the next lesson we’ll download and install Firebug. In Firefox, press F12, and if a panel pops up you already have and can skip to the following lesson. If nothing happens when you press F12 then you need to install Firebug.

Lesson 90: Install Firebug

We need Firebug, the web development toolkit for Firefox, from now on. This allows us to see what's going on, or what's going wrong, with our scripts, because unlike PHP, which fails dramatically and obviously if there's an error in the code, jQuery tells us nothing – it simply doesn't work. This is just what we want in production but can be rather frustrating in development.

If you haven't done so recently, it would be a good idea first to update Firefox. Go to mozilla.org and click on the Download Firefox button and follow the instructions to install it.

Once that's done, go to getfirebug.com and download the latest version of Firebug and follow the prompts to install that in Firefox.

Now start up Firefox and press F12 and the debugging panel should appear at the bottom of the window.

This is where we'll find information about the changing structure of our HTML documents as jQuery alters them, and about things which have gone wrong.

Lesson 91: Basic jQuery syntax

As jQuery scripts are a variety of Javascript, there are two ways of loading them; they can either go in the HTML file inside `<script></script>` tags or they can be put in an external javascript file with a '.js' extension and linked to using `<script src = "(path to external Javascript file)"></script>`.

When we come to our project we'll put them in external Javascript files but for the moment, for ease of editing and because we're working with very simple demos, we'll put them in our HTML documents.

Whichever way we do it, the link to jQuery has to come before any custom scripts or jQuery will not be loaded so they will not function.

Now let's look at the syntax of a jQuery script.

The heart of any jQuery script is the jQuery object. This consists of the keyword "jQuery" followed by parentheses which enclose, in quotes, a selector:

```
jquery(selector)
```

As virtually every line starts with this, it's fortunate that there's an abbreviation for it, the dollar sign, which is nothing to do with the dollar sign we used to start a variable name in PHP:

```
$(selector)
```

I will always use `$` instead of the word 'jQuery' to start my jQuery scripts as it is exactly equivalent and is shorter to type.

The selector in parentheses corresponds to an HTML element or a style selector, just as it appears in our styling.

Following the selector comes an event, things like mouse clicks, mouseover actions, key presses, mouse ups are all events:

```
$(selector).event
```

The event starts a jQuery function, which carries out one or more actions on any elements we specify inside the function:

```
$(selector).event(function(){
    // Action takes place
});
```

So for instance having `'button.hide'` as the selector would target a button or buttons on the page with the class name 'hide':

```
$('button.hide').event(function(){
    // Action takes place
});
```

The selector `'h1'` targets all `<h1>` elements:

```
$('button.hide'). event(function(){
    $('h1')
});
```

So this script would hide all `<h1>` elements on the page when the button is clicked:

```
$('button.hide').click(function(){
    $('h1').hide();
});
```

But before we get that far, we have to that the elements we want to target are actually available and we do this by making sure that the whole page has loaded before we perform any actions on it.

This is done by starting our jQuery with the `document` selector, followed by the event 'ready' to check that the document has fully loaded:

```
$(document).ready(function() {
    // jQuery script goes here
});
```

Only when that is done do we have the function inside which goes all of the rest of our jQuery script.

Look at the syntax, particularly the braces and parentheses.

After 'ready' we open parentheses, then start a function, which has an empty set of parentheses, then we start the function's braces, inside which our script will go.

So now we have a set of braces to close first and then a set of parentheses. Then the closing semi-colon.

This syntax reoccurs constantly throughout jQuery scripts, so make sure you understand why we have that pattern of braces and parentheses.

Here is an example of a fully functioning little bit of jQuery acting on an HTML button. We start with the document ready event. Then select the button, and when it is clicked, run the function which follows:

```
$(document).ready(function() {
    $('button').click(function() {
        // Function to run
    });
});
```

This function selects the `<body>` tag and uses CSS to turn its background color red:

```
$(document).ready(function() {
    $('button').click(function() {
        $('body').css('background-color', 'red');
        $('button').text("It's all gone red!");
    });
});
```

It also selects the text of the button and changes this to "It's all gone red!".

You can try this by opening up 'button-demo.html', which is in the Working Files for this chapter, in your browser and clicking on the button.

This is a simple example of one of the most important features of jQuery – you can perform an action on one element or on a whole group of elements, on the basis of an event which takes place on a different element.

That’s the basics of jQuery syntax. Apart from having to keep track of a great number of nested braces and parentheses as we type, it’s not hard to learn.

In the next lessons we’ll have a deeper look at selectors and events which will be useful for our project.

Lesson 92: The $(this) selector

We have just seen how jQuery allows us to do something we cannot achieve with CSS – we can set up an event on one element which triggers an action on another element.

Frequently however we want the event and action to coincide on the same element.

Open up 'this-selector.html' in Komodo Edit and in Firefox.

In the HTML there is a paragraph `<p class = "add_to_favs">Add to favourites</p>`.

Inside the document ready function, see if you can write the jQuery to make this change to "Remove from favourites" when it is clicked on, using `.text("Remove from favourites")` as the action.

The first thing to remember is to use dot and then the class name as the selector:

```
$(document).ready(function() {
    $('.add_to_favs').click(function() {
    });
});
```

Then, unless you've done this before, you probably repeated the class selector for the action.

```
$(document).ready(function() {
    $('.add_to_favs').click(function() {
        $('.add_to_favs').text("Remove from favourites");
    }):
});
```

Try it in Firefox and it works, but it's not very good programming.

When the selector for the action is the same as the selector for the event which triggers the action, we shouldn't repeat the selector - we should use `$(this)` – with no quotes around it – because the selector is already known to the jQuery script. This saves unnecessary load and speeds up page execution:

```
$(document).ready(function() {
    $('.add_to_favs').click(function() {
        $(this).text("Remove from favourites");
    });
});
```

Try it again in Firefox and it should change to "Remove from favourites". In the next lesson we will make this toggle between "Add to favourites" and "Remove from favourites".

Lesson 93: Add-remove class and the dynamic handler 'on'

It would make sense to be able to click on the text again and have it change back to "Add to favourites".

One way to achieve this would be to use jQuery to dynamically change the class name on the paragraph tag using the actions `addClass` and `removeClass`, so that when the text says "Add to favourites" the class is changed to 'add_to_favs' and when the text says "Delete from favourites" the class is changed to 'delete_from_favs'.

Then we could write a second function targeting the 'delete_from_favs' selector to change the text and class back again.

Save 'this-selector.html' as 'add-remove-class.html' and we'll go on editing it.

We can remove a class name using `.removeClass()` and the class name, but in this situation we do not use the dot in the class name, just the class name itself without the dot prefix:

```
$(document).ready(function() {
    $('.add_to_favs').click(function() {
        $(this).text("Remove from favourites")
            .removeClass('add_to_favs');
    });
});
```

Then we can add on `.addClass('remove_from_favs')`:

```
$(document).ready(function() {
    $('.add_to_favs').click(function() {
        $(this).text("Remove from favourites")
            .removeClass('add_to_favs')
            .addClass('remove_from_favs');
    });
});
```

So you can see that we are performing more than one action simply by adding them on the end in a list. This is called 'chaining' and it's a key feature of jQuery, which makes it possible to perform any number of actions triggered by an event simply by adding them on like this.

So the idea is to target the paragraph using its class name and not only change its text but also the class name.

Then we can then write another function doing the same thing in reverse, acting on the new 'remove_from_favs' class:

```
...
    $('.remove_from_favs').click(function() {
        $(this).text("Add to  favourites")
             .removeClass('remove_from_favs')
             .addClass('add_to_favs');
    });
```

changing the text back to "Add to favourites" and the class name back to 'add_to_favs'.

Save and refresh and click – and it doesn't work.

To see what is going on, open up Firebug, choose 'HTML' and you you will see that the HTML is being changed. 'Add_to_favs' changes to 'remove_from_favs', just as we'd like, but the second

click doesn't work – it will not change back to 'Add to favourites'.

The problem is that the `click` event is not responding to the new class 'remove_from_favs' because this class has only just been added on the fly by jQuery.

To target newly added elements like this one we have to use different syntax to set up the event, using the event handler `on`:

Replace the selector with 'document', with no quote marks:

```
$(document)
```

then the event handler `on`:

```
$(document).on
```

then in the parentheses we have a comma-separated list. First we have the event, which is `click`, then a comma:

```
$(document).on('click',
```

then the class selector, `'.add_to_favs'`, and a comma:

```
$(document).on('click', '.add_to_favs',
```

and then the function as before:

```
$(document).on('click', '.add_to_favs', function() {
    // Action as before
});
```

Do the same again for the `'.remove_from_favs'` class:

```
$(document).on('click', '.remove_from_favs', function() {
    // Action as before
});
```

Save and refresh – and now it works.

When we want to target elements which have just been added by jQuery, we have to use the `on` event handler and this syntax, and we'll be using it a lot in the project.

Lesson 94: Retrieving and using HTML attributes

We saw the difference between classes and IDs in the chapter on styling. Can you spot what is wrong with this snippet of HTML?

```
<ul class="favs">
    <li id="movie">Movie 1</li>
    <li id="movie">Movie 2</li>
    <li id="movie">Movie 3</li>
    <li id="movie">Movie 4</li>
</ul>
```

It's incorrect, and would not pass validation, because the purpose of IDs is to uniquely identify particular elements on the page – we cannot use the same ID name more than once on a page.

IDs are central to jQuery precisely because they allow us to target individual elements.

Open up 'retrieve-attibutes.html' in Komode Edit and Firefox. The HTML provided consists of two lists, a list of favourite movies and a list of non-favourites, distinguished by class names on divs.

In the style section the 'favs' and 'non_favs' divs are positioned using float left so that they are alongside each other. Each list item is given the class 'movie_list' and a unique ID name consisting of 'movie_' and an ID number. In the style section, the '.movie_list' class is given a background-colour on its hover state to give us a bit of visual feedback.

So you can see that this is a simplified version of the favourites and non-favourites lists in the project; we're working towards a script which will remove a list item from one list and add it to the other when it is clicked.

Inside the `document ready` function, write the code to select any of the movie list items using the `on` event handler we learnt about in the previous lesson.

The answer is:

```
$(document).ready(function() {
    $(document).on('click', '.movie_list', function() {
    });
});
```

Now we can perform actions on the list item we clicked on using `$(this)` as the selector. We can remove it from the page using `.remove()`:

```
...
    $(document).on('click', '.movie_list', function() {
        $(this).remove();
    });
...
```

Save and refresh, and any item we click on will disappear. But how do we identify which item we've clicked on in one list and add that particular item to the other list? We want to get get jQuery to look into the HTML and get the ID number of the list item clicked.

Start out with a variable assignment `$id =`:

```
...
    $(this).remove();
    $id =
...
```

Variables in Javascript do not need to begin with a dollar sign, unlike those in PHP, which do, but it's common in jQuery to prefix them with a dollar sign, so this is what I will do.

Now we can assign to the variable `$id` the value of the ID for the item clicked on by using the attribute method, `attr()`:

```
...
    $(this).remove();
    $id = $(this).attr('id');
...
```

Unlike in PHP, we cannot check that our variable assignments are as we expect by using `echo` to print them temporarily on the webpage. Instead we'll use the Firebug console.

Adding `console.log($id)` will output the value assigned to `$id` in the console window:

```
...
    $id = $(this).attr('id');
    console.log($id);
...
```

In Firefox, press F12 to open Firebug. Click on 'Console' and then on 'All'.

Save and refresh and click on the various movie list items and you should see the changing values of `$id`.

It would be more convenient if we could get rid of the prefix – the `movie_` part and use just the numerical part.

We can do this using the `split()` method with '_' as its argument:

```
...
    $id = $(this).attr('id').split('_');
    console.log($id);
...
```

Save and refresh and click on the movie names and you will get the two parts `movie` and the ID number separated.

These are now array values and we can get just the numerical value by specifying the index position we want.

As we saw in the lesson on arrays, array fields are indexed from position zero, so the field we want is position 1. So to assign just this to `$id` we can add a new line, `$id = $id[1]`:

```
...
    $id = $id[1]:
    console.log($id);
...
```

Save and refresh and now we have the index value we need.

Now we can make the correct movie list item from the favourites list appear in the non-favourites list when we click on it, using the `append()` method:

We need to append it to the `<ul>` inside the `non_favs` div, so the selector is a parent-child selector `'.non_favs ul'`:

```
...
    console.log($id);
    $('.non_favs ul')
...
```

Then we append a formatted string consisting of a list item, whose class is `movie_list`:

```
...
    $('.non_favs ul').append("<li class='movie_list')
...
```

and then give it the ID `'movie_'` :

```
...
    $('.non_favs ul').append("<li class='movie_list' id='movie_")
...
```

with the index number we have just got from the jQuery above concatenated to it.

In Javascript we use a plus sign for concatenation, not the dot concatenator we used in PHP, so add a 'plus' sign before and after `$id`.

```
...
    $('.non_favs ul').append("<li class='movie_list' id='movie_" + $id +)
...
```

Then close the html tag:

```
...
    $('.non_favs ul').append("<li class='movie_list' id='movie_"
    + $id + "'>
...
```

Put in the text, which is 'Movie' and a space:

```
...
    $('.non_favs ul').append("<li class='movie_list' id='movie_"
    + $id + "'>Movie "
...
```

to which we append the ID number again:

```
...
    $('.non_favs ul').append("<li class='movie_list' id='movie_"
    + $id + "'>Movie " + $id +
...
```

and close off the `<li>` tag:

```
...
    $('.non_favs ul').append("<li class='movie_list' id='movie_"
    + $id + "'>Movie " + $id + "</li>");
...
```

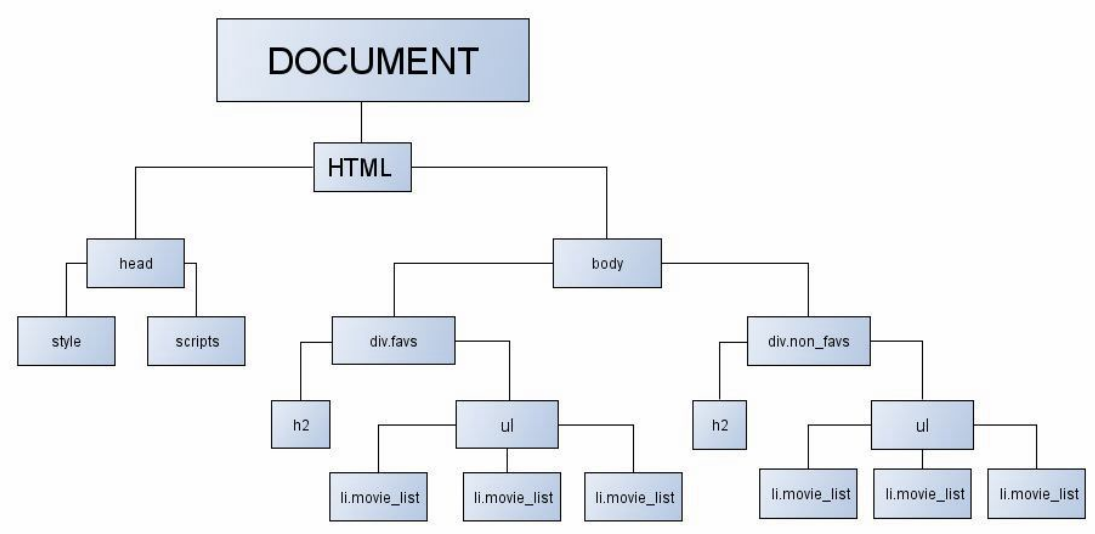
That's OK except it only works one way – deleting from the list on the left and adding to the one on the right.

To make it work both ways we could duplicate the whole lot and change ‘favs’ to ‘non_favs’ and vice-versa throughout but that would be inelegant and repetitive. Instead, let’s find out which list the user has clicked on by using `attr()` again.

To do this we need to look from the list item clicked on back in the HTML structure until we find the enclosing div, which has the class name we are looking for.

JQuery provides very powerful and convenient tools for doing this sort of thing, called DOM traversal. DOM stands for Document Object Model, which is a representation of an HTML document as a tree, with a root and branch structure.

This HTML document would be represented like this, with two major branches, the `<head>` and the `<body>`, and in the body the two divs, each with an `<h2>` and a `<ul>` tag, each `<ul>` having three `<li>` tags:



We need to look for the closest div in the HTML structure to the item clicked, which we can do using `closest('div')`:

```
...
    $id = ($(this).attr('id').split("_"))[1];
    $(this).closest('div')
...
```

and then getting one of its attributes, its class name:

```
...
    $(this).closest('div').attr('class');
...
```

and assigning this to a variable, which we’ll call `$list`:

```
...
    $list = $(this).closest('div').attr('class');
...
```

With a simple `if` clause we can assign the name of the list found to a variable, putting a dot in front of it because it is a class selector:

```
if ($list=="favs") {
    $addList = ".non_favs";
} else {
    $addList = ".favs";
}
```

and then use this as the selector by concatenating the variable name and `' ul'`, with a space in front of it, to form a parent-child selector:

```
$($addList + ' ul')
```

and then using this as the selector for the `append()` statement we made earlier:

```
$($addList + ' ul').append("<li class='movie_list' id='movie_"
+ $id + "'>Movie " + $id + "</li>");
```

Save and refresh and that works both ways. We're well on the way to success.

Delete the `console.log($id)` line – we don't need that – and cut out the line where we remove the list item and paste it into just above where we append it to the other list. Save and refresh, and that works both ways.

It would be nice to sort the newly added list item so it appears in the right place straightaway, and we can do this with a jQuery plugin called 'tinysort'.

I have downloaded this and saved a copy in the 'scripts' folder in the Working Files, one folder up from where we are. Add a relative link to it just after the link to jQuery and before our custom script:

```
<script src="../../SCRIPTS/jquery.tinysort.min.js"></script>
```

Then to sort the list on the fly we add `$('li').tsort()` just after we have added the new item to the list. The whole jQuery script should be:

```
$(document).ready(function() {
    $(document).on('click', '.movie_list', function() {
        $id = ($(this).attr('id').split("_"))[1];
        $list = $(this).closest('div').attr('class');
        if ($list=="favs") {
            $addList = ".non_favs";
        } else {
            $addList = ".favs";
        }
        $(this).remove();
        $($addList + ' ul').append("<li class='movie_list' id='movie_"
        + $id + "'>Movie " + $id + "</li>");
        $('li').tsort();
    });
}); // End doc ready
```

That's a simplified version of the 'click to add to/delete from favourites' jQuery effect. In the next lesson we'll look how to make the drag-and-drop version of this.

Lesson 95: One-way drag-and-drop effect using jQuery UI

Our project features a one-way drag-and-drop effect, which serves as the way to remove items from the favourites list. The user drags a list item to the trash can, the trashcan hover effect kicks in to provide visual feedback that an action has been recognised, and the item is removed from the list. In the background, the corresponding item is removed from the favourites table so that when the page is refreshed the changes persist.

Let's see first how to achieve just the visual effect in simple terms before we do it in the project.

Open up 'drag-and-drop.html' in Komodo Edit and Firefox.

This is similar to the file we used for the previous lesson except we now only have one list, for the favourites, and the trash can below it.

The 'trash' div style gives it a background image using the CSS sprites we made earlier, and this changes to a trash can with things in it when we hover over it.

We have the link to jQuery in place as before but we also need a new link, to the jQuery UI library, which I have put in. This adds user interface interactions and effects which we need to make the drag-and-drop effect work.

JQuery UI is dependent on jQuery so the link to it must come after the link to jQuery. And we need it for our script so it must come before that.

To make the list item draggable we use the 'draggable' widget:

```
$('.favs li').draggable({
)};
```

Notice the parentheses and braces.

Make a comment `// end draggable` after the closing braces and parentheses. Comments are needed in jQuery to show us where all the sets of braces and parentheses end.

Save and refresh and hold the mouse down over the list items and you can drag them anywhere on the page, leaving a gap where the item was. When you let go they will stay in that position, and the gap remains.

To prevent the list item from staying in any position it is dragged to, instead of dragging the item itself, we'll drag a copy, or 'clone' of it, using `helper: clone`:

```
$('.favs li').draggable({
    helper: 'clone'
)}; // End draggable
```

Save and refresh. And now when we drag items, no gap opens up and the dragged item doesn't stay where we dragged it.

That looks the part but doesn't do anything when we drag an item to the trash can; it doesn't delete the item from the original list. We need to set up the corresponding 'drop' event on the trash can.

This is done with matching syntax – the 'droppable' widget on the trash class selector:

```
$('.favs li').droppable({
)}; // End droppable
```

We want this div to accept `<li>` items from the 'favs' class so we use `accept: '.favs li'`:

```
$('.favs li').droppable({
    accept: '.favs li',
)}; // End droppable
```

Then we set up a function which takes place when the drop event takes place:

```
$('.favs li').droppable({
    accept: '.favs li',
    drop: function() {
    }
)}; // End droppable
```

This function takes two arguments, 'event' and 'ui', for 'user interface':

```
$('.favs li').droppable({
    accept: '.favs li',
    drop: function(event, ui) {
    }
)}; // End droppable
```

The selector 'ui.draggable' can now be used to select the dragged item:

```
$('.favs li').droppable({
    accept: '.favs li',
    drop: function(event, ui) {
        $(ui.draggable)
    }
)}; // End droppable
```

and we can use the same technique we used in the previous lesson to get the ID of the item dragged from its attribute, with `attr()`:

```
$('.favs li').droppable({
    accept: '.favs li',
    drop: function(event, ui) {
        $id = $(ui.draggable).attr('id')
    }
)}; // End droppable
```

again using `split()`, splitting the ID name on the underscore:

```
...
    drop: function(event, ui) {
        $id = $(ui.draggable).attr('id').split("_")
    }
...
```

and getting just the value in index position 1 in the array and assigning this to `$id`:

```
...
    drop: function(event, ui) {
        $id = $(ui.draggable).attr('id').split("_");
        $id = $id[1];
    }
...
```

Now we know which item to remove from the favourites list and we can select it using a selector made up of `movie_` concatenated to the ID number with the + sign:

```
...
    drop: function(event, ui) {
        $id = $(ui.draggable).attr('id').split("_");
        $id = $id[1];
        $('li.movie_' + $id)
    }
...
```

and remove it from the HTML with `.remove()`:

```
...
    drop: function(event, ui) {
        $id = $(ui.draggable).attr('id').split("_");
        $id = $id[1];
        $('li.movie_' + $id).remove();
    }
...
```

Save and refresh and that all works.

A visual enhancement would be to make the hover state of the trash can appear during the drag action.

Add a comma after `helper: clone` as this is a comma separated list of options:

```
$('.favs li').draggable({
    helper: 'clone',
)}; // End draggable
```

We can do this by adding a function to the drag event:

```
$('.favs li').draggable({
    helper: 'clone',
    drag: function(){
    }
)}; // End draggable
```

selecting the ‘trash’ div:

```
...
    drag: function(){
        $('trash')
    }
...
```

making a new style called ‘trash_hover’ in ‘style.css’, exactly the same as the hover state style, by adding it to the hover state:

```
.trash:hover, .trash_hover {
     background-position: 0 -61px ;
}
```

And adding the class ‘trash_hover’ using `addClass()`:

```
...
    drag: function(){
        $('.trash').addClass('trash_hover');
    }
```

Save and refresh and we have the hover effect as well.

As a final touch, add ‘cursor: pointer’ to the movie list hover state style to give us a hand when we mouse over:

```
.movie_list:hover {
    background-color: grey;
    cursor: pointer;
}
```

We can add more niceties, like more emphatic styling to emphasise that an item has been added to a list, but these two lessons have covered the basics of making our ‘click to add’ and ‘drag-and-drop to delete’ effects.

But they don’t actually do anything – they don’t add or remove the favourites from the database because we’re not even connected to the database.

In the next lesson we’ll look at the basics of making AJAX calls and how we use these to update the database in the background with no page refresh.

Lesson 96: AJAX - Update database with no page refresh

We have now made simplified versions of the click-to-add and drag-to-remove jQuery effects but with no database interaction. In this lesson we will write the code for a basic 'click to add to favourites' which updates the database with no page refresh.

In the next lesson you can put into practice what you have learnt in this one in an assignment to do the 'remove from favourites' interface.

It will lack some of the adornments of the final project but the functionality will all be there. When we refresh the page after adding or deleting favourites, the change has been made in the database.

JQuery alone does nothing to the database; to alter database records we must use PHP. AJAX simply calls an ordinary PHP script and passes it variables, in this case `$userID` and `$movieID`. The PHP script does the work in the background as jQuery performs visual effects.

The visual effects and the database update are actually not linked. If we wanted to, we could have jQuery make changes to the HTML which are completely unrelated to the changes made to the database records. But in practice we want to make sure that the HTML changes reflect the database changes so as to give the illusion of an instant update.

Let's look at the syntax of an AJAX call.

The AJAX call goes inside the triggering event:

```
$(document).on('event', 'selector', function() {
    $.ajax({
    });
});
```

and needs as a minimum three pieces of information, which we enter as key/value pairs within the braces, `url`, the url of the PHP script called by AJAX relative to 'index.php':

```
...
    $.ajax({
        url: "path_to/php_script.php"
    });
...
```

`type`, the method used to send the data, which is either POST or GET:

```
...
    $.ajax({
        url: "path_to/php_script.php",
        type: "POST", // post or get
    });
...
```

and `data`, the data to send to the PHP script:

```
...
    $.ajax({
        url: "path_to/php_script.php",
        type: "POST", // post or get
        data:
    });
...
```

Data is itself a comma-separated list of key/value pairs inside its own set of braces:

```
...
    $.ajax({
        url: "path_to/php_script.php",
        type: "POST", // post or get
        data: {
            'variable1': $variable1,
            'variable2': $variable2
        }
    });
...
```

In our case the items of data will be `'movie_id'` and `'user_id'`.

That's a basic AJAX call.

As we are now working in PHP we need to have our files back in 'htdocs' and load them through http://localhost in the browser.

Go to the Working Files for this chapter and select the folder 'ajax-demo'. Copy this folder, and paste it into 'htdocs', alongside 'favouritemovies' and 'php-demos'.

We need to have a lot of our database functionality working again, so this folder contains simplified versions of a number of files from the project. Start up Komodo and let's have a look at what we've got.

'Index.php' starts out with `set_include_path()` as before and then includes vital files – the database connection, get-variables, the head section, and a simplified version of the `showMovies()` function.

We get the data for our two lists using `showMovies('favs')` and `showMovies('non_favs')`, as before.

Then we have the HTML for the two lists and echo the database records out as list items.

I have added a div above the trash can which is styled in 'style.css' to clear the float so the trash can is unaffected by the two column layout produced by the floats.

'Head.inc.php' links to the stylesheet, jQuery and the jQuery UI.

Then we have a script of just one line:

```
<script>var $userID  = "<?php echo $userID; ?>"</script>
```

In this we echo a PHP variable into jQuery, assigning the echoed PHP variable `$userID` to the jQuery variable of the same name.

This is needed because otherwise jQuery knows nothing about PHP or its variables – as far as it knows, all we have is the HTML on the page – it has no idea how this was generated.

Then we have the links to the jQuery scripts at the heart of the lesson, 'remove-favs.js' and 'add-favs.js'. These are both familiar.

‘Add-favs.js’ reacts to a click on a ‘non_favs’ list item:

```
...
$(document).on('click', '.non_favs .movie_list', function() {
}); // End movie_list click function
...
```

It gets the ID number using `attr().split("_")` as before:

```
...
    $id = ($(this).attr('id').split("_"));
    $id = $id[1];
...
```

It gets the movie title using the method `text()`:

```
...
    $title = $(this).text();
...
```

removes it from the ‘non_favs’ list:

```
...
   $(this).remove();
...
```

and adds a new version of it to the favourites list:

```
...
   $(".favs ul").append("<li class='movie_list' id='movie_" + $id + "'>"
   + $title + "</li>");
...
```

‘Remove-favs.js’ is the drag-and-drop interface we made in the previous lesson.

Let’s work on ‘add-favs.js’ first.

Type the shorthand for jQuery AJAX, `$.ajax` and its parentheses and braces, with a comment after the closing parentheses and braces:

```
...
   $(".favs ul").append("<li class='movie_list' id='movie_" + $id + "'>"
   + $title + "</li>");

   $.ajax({

   }); // End AJAX
...
```

Now for our three pieces of information,

The URL is the PHP script, relative to ‘index.php’, which is ‘ajax/add-favs.ajax.php’:

```
...
   $.ajax({
      url: 'ajax/add-favs.ajax.php',
   }); // End AJAX
...
```

For the method used to send the data, we'll use POST:

```
...
   $.ajax({
      url: 'ajax/add-favs.ajax.php',
      type: "POST",
   }); // End AJAX
...
```

And the data is a comma-separated list of key-value pairs inside their own set of braces:

```
...
   $.ajax({
      url: 'ajax/add-favs.ajax.php',
      type: "POST",
      data: {
      }
   }); // End AJAX
...
```

We'll use the variable and field name naming conventions that we used earlier, setting the variable `$id` to be posted to the PHP script as `user_id` and `$movieID` as `movie_id`. Each item needs a comma after it except the last one, which must not have a comma:

```
...
  data: {
    'movie_id': $id,
    'user_id': $userID
  }
...
```

The closing brace for `data {}` does not have a comma because it is the last in this list of three options.

That's our AJAX call.

But we've put in a link to a non-existent PHP file in the URL option.

Navigate to http://localhost/ajax-demo/index.php?user_id=2 in your browser. Save and refresh and try the click-to-add interface. Refresh again afterwards.

The jQuery effect works but the database records, unsurprisingly, are unchanged. We weren't given any error messages to tell us anything was wrong.

Make sure Firebug is open and set to 'Console – All' and when you click you should get red error messages each time you click telling you that there has been a Network error – '404 Page not found'.

Because AJAX requests work silently, errors are not reported in the browser window as they are in regular PHP – to find errors you have to look in Firebug.

We still have to write the PHP script which actually does the work. So make a new file in the 'ajax' folder named 'add-favourites.ajax.php').

This script is free-standing from the rest of the project so it has not yet connected to the database. Open it in Komode Edit and add `require_once '../php-includes/connect.inc.php';` The path has to be relative from this file.

Then we receive the data posted in the URL section of the ajax request using the POST method:

```
$userID = $_POST['user_id'];
$movieID = $_POST['movie_id'];
```

There'll be no need to sanitise it in the project as we will have already done that in our error checking and we're using a prepared statement anyway.

Then we set up a prepared statement just as before, inserting a new record matching `movie_id` and `user_id` into the favourites table.

You should be able to write all the lines needed to write the prepared statement, bind the parameters, execute the prepared statement, get the results and assign them to a variable `$result`. And end by echoing `$result` so this is sent back to the AJAX call. Don't forget to close the prepared statement.

Try to do this yourself before you look at the answer:

```
$stmt = $db->prepare("INSERT INTO `favourites` (movie_id, user_id)
        VALUES (?, ?)");
$stmt->bind_param('ii', $movieID, $userID);
$stmt->execute();
$result = $db->query($stmt);
$stmt->close();
echo $result;
```

Save and refresh and the 'click-to-add' interface should work. Refresh after clicking and the newly added movie should remain in the list on the left, showing that the database has been updated.

Now you should be able to use the same principle to write the corresponding AJAX and PHP for the drag-and-drop function and this is your next assignment.

The corresponding PHP script 'remove-favs.ajax.php' is identical to 'add-favs.ajax.php' except for the SQL in the prepared statement.

In 'remove-favs.js' add the AJAX call and its three options, URL, TYPE, and DATA.

I'll give you the answer in the next video and in the Working Files.

Assignment 5: The drag-to-delete AJAX call and PHP script

Using the principles we have just learnt, add the basic AJAX call to ‘remove-favs.js’ and write the new PHP file ‘remove-favs.ajax.php’.

The PHP file is identical to add-favs.ajax.php except for the SQL, and the AJAX call needs only slight modification.

When you have done this, try both add and delete features in your browser.

Lesson 98: The drag-to-delete AJAX call and PHP script

The answer to the assignment is simple – you can copy the AJAX call we just wrote in 'add-favs.js' and save it as 'remove-favs.js'.

It needs to go inside the 'droppable' so it is fired when the item is dropped onto the trashcan. Change the name of the PHP script to 'remove-favs.ajax.php':

```
$('.trash').droppable({
...
    $.ajax({
        url: "ajax/remove-favs.ajax.php",
        type: "POST",
        data: {
            'user_id' : $userID,
            'movie_id' : $id
        } // End data
    });  // End AJAX
...
```

Copy 'add-favs.ajax.php' and save it as 'remove-favs.ajax.php'.

Open it up in Komodo Edit and change the SQL in the prepared statement:

```
$stmt = $db->prepare("DELETE FROM favourites WHERE movie_id = ?
    && user_id = ?");
```

Save and refresh and that appears to be done.

But if you've tried it in the browser carefully you may have found it's not quite as done as it appears. Try clicking on the list on the right to add several items to the favs, and then, without refreshing the page, try dragging any of the items you have just added to the trashcan. It won't work – they aren't draggable.

We can solve this in the same way as we did before by making them clones using `helper: clone`.

After the `append()` line in 'add-favs.js', add the `helper clone` clause we used earlier, acting on the draggable favourites movie list item:

```
$(".favs ul").append("<li class='movie_list' id='movie_" + $id + "'>"
+ $title + "</li>");
$(".favs li#movie_" + $id).draggable({
    helper: 'clone'
});
```

Save and refresh and try adding favourites and then immediately dragging them to trash – problem solved.

At the moment our two scripts remove and append the list items in a way which isn't linked to the execution of the AJAX call. The visual change happens immediately the moment we click or drag. It would be better if the if the removal from the original list took place before the AJAX call is executed and the addition to the new list took place upon a successful database update.

We can specify the timing of the jQuery effects relative to the execution of the AJAX call by using two more options in the AJAX call, `'beforeSend'`, and `'success'`:

```
$.ajax({
      url: "path_to/php_script.php",
      type: "POST",
      data: {
            'variable1': $variable1,
            'variable2': $variable2
      }, // End data
      'beforeSend' : function() {
            // Action(s) to perform before execution of PHP script
      },
      'success': function(result) {
            // Action(s) to perform after execution of PHP script
      }
}); // End AJAX
```

`'BeforeSend'` triggers a function to run effects we want to occur before the PHP script is executed, and `'success'`, triggers a function to run effects we want to occur after the PHP script is successfully executed. So using `'beforeSend'` and `'success'`, we can control the timing of the jQuery effects to match the database update.

In 'add-favs.js' first, add a comma and then add `'beforeSend'` and its function and braces:

```
$.ajax({
    url: "ajax/add-favs.ajax.php",
    type: "POST",
    data: {
        'user_id' : $userID,
        'movie_id' : $id
    }, // End data
    'beforeSend': function(){

    }, // End beforeSend
});
```

Move the `remove()` line from its present location down to inside the `beforeSend` function:

```
...
    'beforeSend': function(){
        $(this).remove();
    }, // End beforeSend
...
```

We can no longer use the `$(this)` selector here, as its reference here is no longer the same as it was in its previous position, so up where we set the variables `$title` and `$id`, we'll make another variable called `$this` and set its value to the jQuery selector `$(this)`:

```
$title = $(this).text();
$this = $(this);
```

then the variable `$this` will hold that value throughout the rest of the script, no matter where we use it.

Change `$(this).remove()` to `$this.remove()` – now the reference is correct:

```
...
    'beforeSend': function(){
        $this.remove();
    }, // End beforeSend
...
```

Move the `append()` line and the function following it, which `clone`s the appended item, to inside the `success` function:

```
...
    'success': function() {
        $(".favs ul").append("<li class='movie_list' id='movie_"
          + $id + "'>" + $title + "</li>");
        $(".favs li#movie_" + $id).draggable({
            helper: 'clone'
        }); // End draggable
    } // End success
...
```

Now we'll do the equivalent in 'remove-favs.js'.

First set the value of `$this` to the jQuery selector `$(ui.draggable)` to set the value of the variable to the dragged item:

```
$this = $(ui.draggable);
```

Then, inside the AJAX call, make a `'beforeSend'` option and in its function put `$this.remove()`:

```
'beforeSend': function() {
    $this.remove();
}, // End beforeSend
```

Make a `'success'` option and move the append line into there:

```
'success': function() {
    $(".non_favs ul").append("<li class='movie_list' id='movie_"
       + $id + "'>" + $title + "</li>");
} // End success
```

The complete AJAX call, with its five options `url`, `type`, `data`, `'beforeSend'` and `'success'` is now:

```
$.ajax({
    url: "ajax/remove-favs.ajax.php",
    type: "POST",
    data: {
        'user_id' : $userID,
        'movie_id' : $id
    }, // End data
    'beforeSend': function() {
        $this.remove();
    }, // End beforeSend
    'success': function() {
        $(".non_favs ul").append("<li class='movie_list' id='movie_"
        + $id + "'>" + $title + "</li>");
    } // End success
}); // End AJAX
```

Now in the browser make sure you have some non-favourites in the right hand list and then click on them very rapidly: Another problem! We get duplicate items being added to the favourites list.

This is a problem known as event bubbling, where the clicks are happening faster than the database updates can cope.

To prevent this we can set a variable which checks whether an AJAX request is running and if so, prevent any more clicks until it has completed.

Near the top of 'add-favs.js', just inside the `document ready` function, before the `click` event, assign the value 'false' to a variable named `$requestRunning`:

add.favs.js:

```
$(document).ready(function() {
    $requestRunning = false;
...
```

Then in the `'beforeSend'` function set the value of `$requestRunning` to 'true':

```
'beforeSend': function(){
    $requestRunning = true;
    $this.remove();
}, // End beforeSend
```

In the `success` function set it back to 'false':

```
'success': function() {
    $requestRunning = false;
...
```

Then just below the click event put in an `if` clause with `return` if `$requestRunning` is 'true'. `Return` exits the function and prevents the click event from doing anything if an AJAX call is in process:

```
$(document).on('click', '.non_favs .movie_list', function() {
    if ($requestRunning) {
        return; // Don't do anything if an AJAX request is running
    }
...
```

Save and refresh, and the response to clicks will be much slower, allowing time for each update to take place before the next one is allowed.

Later we will add some niceties like an AJAX loader image, but that for the moment is the database functionality sorted out and the effects properly working and timed.

In the next chapter we will put these principles into practice in the project.

Chapter 11: jQuery effects and AJAX interaction

Lesson 99: Adapt jQuery for project

Our task in this chapter is to write the code for all the jQuery effects and AJAX calls in the project. By the end we will be able to add and delete favourites, using these heart icons, which will appear when we hover the mouse anywhere over the list items.

When we drag movies to the trash can they are removed from favourites and a little success icon appears to show that a new item has been added to the list on the right.

Our navigation will work fully, giving lists which appear when we move the mouse over them, allowing us to get to the admin pages, where we will be able to add, delete and modify movie and movie-goer records, all without refreshing the page.

At present we've got a prototype version of our jQuery and AJAX interface working but before we continue we need to make a few minor alterations to adapt it to work with the more complicated HTML in the project.

Our two PHP scripts called by AJAX need no changes – so copy the 'ajax' folder and its contents from 'ajax-demo' into the 'favouritemovies' folder. We'll adapt our Javascript files from the previous chapter, so copy the 'scripts' folder and its contents from 'ajax-demo' into 'favouritemovies' as well.

Open up the version of 'head.inc.php' in 'ajax-demo', and copy and paste into the project version of 'head.inc.php' after the 'print-shiv' link the following lines:

```
<script src =
http://ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js">
</script>
<script src =
"http://ajax.googleapis.com/ajax/libs/jqueryui/1.10.3/jqueryui.min.js"></s
cript>
<script>var $userID  = "<?php echo $userID; ?>"</script>
<script src = "scripts/remove-favs.js"></script>
<script src = "scripts/add-favs.js"></script>
```

Load up the project in your browser, and select a user who has both favourites and non-favourites so that we can see the full source code.

Your database contents and mine might not be the same now as we altered the records in the previous chapter.

Right click in the browser window and choose 'View Page Source' to open up the HTML source.

We'll need to check throughout this chapter in the HTML to see the style selectors we're working with in the HTML and jQuery to make sure they match, so you can see how important it is to have nicely formatted HTML.

Looking at the non-favourites list in the source code, we can see that the opening `<ul>` tag has no class name – I missed this earlier. This is set in main.inc.php.

In 'main.inc.php', find the 'else' clause and change `$openTag = "<ul>"` to `$openTag = "<ul class='non_favs'>"`.

In the two Javascript files, in the `append()` lines where we add new list items, the selectors for the `<ul>` elements are wrong– they're currently parent-child selectors because in the demo chapter I had the class names set on the divs, above the `<ul>` elements in the DOM heirarchy. In the project, the class names are on the <ul> elements themselves so the selectors we use on the append lines have to be changed to combination selectors.

Find the `append()` lines and change `$(".non_favs ul")` to `$("ul.non_favs")` and change `$('.favs ul')` to `$('ul.favs')`.

At the moment we've got a lot of selectors not doing much on the div with the heart icon on it. We'll use this as the element the user clicks to add an item to favourites. I think we can simplify things by simply naming this 'add'.

There are three changes to make to do this:

1) In 'show-movies.fn.php', change `<div class = "add_remove favourite">` to `<div class = "add">`.

2) In 'style.css', change `.movie_list li .add_remove` to `.non_favs .add`.

3) In 'add-favs.js', change the click selector to `'.non_favs  .add'` to match.

As we've seen, JQuery relies on unique ID numbers and we have no ID numbers for our movie list items yet. We need to add these in the `showMovies()` function.

In 'show-movies.fn.php' in the 'favs' output, change `$output .= <li>` to `$output .= "<li id='fav_$id'>";`.

And in the 'non_favs' area add change the `<li>` output to `"<li id = 'nonfav_$id'>\n"`.

In the prototype I used 'movie_' as the ID prefix everywhere, which is OK, but it means we need to use parent-child selectors to distinguish the two lists. It's good practice to minimise the use of complicated selectors in jQuery as much as possible because they do take resources and if there are too many of them in a large project this will have an impact on performance.

So we'll use 'fav_' and 'nonfav_' as prefixes to distinguish the two lists without needing parent-child selectors. I've used the singular, 'fav_' and 'nonfav_' for the IDs, not 'favs_' and 'nonfavs_' because the name is more accurate as each ID refers to one movie only.

Also make sure there is no underscore between the 'non' and the 'fav' in the 'nonfav_' ID prefix or the use of the array position to set the variable will not work.

Refresh and double-check the source code and in the browser; make sure there are no red bits highlighting errors. You can even validate it to make sure.

In next lesson we'll continue setting up the click-to-add interface by making the little heart icons appear when we mouseover them so we can click on them, and disappear when we move the mouse off them.

Lesson 100: Toggle background image on mouseover

There are too many little heart images in the main movie list – not at all tasteful. My idea is to have these appear one at a time only when you mouseover each movie list item.

The heart comes from the class style 'favourite' in 'style.css', which gives a two-color effect when you hover over it.

We can use `toggleClass()` in jQuery to switch the 'favourite' class on and off.

In 'add.favs.js', just inside the `document ready` function and above the `$requestRunning` variable assignment, add an `on` event handler using `'mouseover mouseout'` working on the `'non_favs'` list item, `'.non_favs li'`:

```
$(document).ready(function() {
    $(document).on('mouseover mouseout', '.non_favs li', function(){
    });
$requestRunning = false;
...
```

This will run its function whenever we mouseover or mouseout from a 'non_favs' list item.

But we don't want the effect in the function to take place on the whole of the list item – just on the 'add' div within it.

There's a very convenient way in jQuery to target a particular style selector within the one we are using to trigger the event. We add a second parameter in the jQuery constructor, in this case `'.add'`, then a comma, and then `'this'`. So `$('.add', this)` will target the '.add' style inside the 'non_favs' list item:

```
...
    $(document).on('mouseover mouseout', '.non_favs li', function(){
        $('.add', this)
    });
...
```

`ToggleClass()` adds the class name if it is not already on the item targetted and removes it if it is:

```
...
    $(document).on('mouseover mouseout', '.non_favs li', function(){
        $('.add', this).toggleClass('favourite');
    });
...
```

Save and refresh, and that's fine – the heart icons are not there until we mouse over them, so we only see one at a time. Now we're ready to do the click-to-add interface.

Lesson 101: The click-to-add interface

Now we're going to adapt 'add-favs.js' for use in the project so that when we click on one of the heart icons, that movie is added to the movie-goer's favourites list and and removed from the non-favourites list, and the database is updated in the background.

We're going to need to set a number of variables including the movie title and description. It's good practive to use variables as much as possible instead of repeatedly using jQuery selectors to save resources and give faster processing.

We'll start out by assigning the currently selected element to the variable `$this`, referring to the div named 'add' which we click on to add movies to favourites:

```
$(document).on('click', '.non_favs .add', function() {
    ...
    $this = $(this);
    ...
```

On its own that's not much use – we need to get from that 'add' selector back up the DOM heirarchy to find useful information, like the ID of the selected item.

We can use `closest('li')`, finding the closest list item back up in the page structure, and assign this to a variable, `$this_li`, as well:

```
$this_li = $this.closest('li');
```

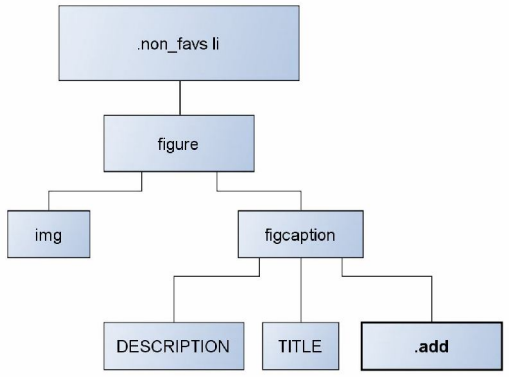
Then we can use the variable `$this_li` instead of another jQuery selector to set the variable `$id`, using the `attr('id')` method with `split("_")`, and then using just position 1 of the resulting array, just as before:

```
$id = ($this_li).attr('id').split("_");
$id = $id[1];
```

When we remove items from the favourites list later, they will appear back in the main movies list, and this includes the description of each movie, so we need to know the movie description for each record.

To get that information we need to have the description available somewhere in the favourites list items. We could do this using hidden text, but a way I like is to have the description of the movies as the `<title>` tag of the favourites list items. In this way, when a user mouses over each favourite, the description will appear, giving more details about the movie.

But how can we get the title and description? Both are siblings of the selected element, the 'add', alongside it in the page heirarchy, with the same parent, the `<figcaption>` element:



To traverse the DOM from the add div to its siblings we use the jQuery method `siblings()`, working on `$this`:

```
$title = $this.siblings()
```

For the title we are looking for the `<h3>` element, which is a sibling of the 'add' div:

```
$title = $this.siblings('h3')
```

and then getting its text:

```
$title = $this.siblings('h3').text();
```

For the description, we want to get the 'description' class with the same relationship to the 'add' div, another sibling:

```
$description = $this.siblings('.description').text();
```

Then using that information we need to build up a string to append onto the favourites list building up a list item. We'll do this as we did earlier in the PHP functions, forming a variable named `$output` by concatenating parts to it.

You will remember that in Javascript the concatenation operator is a plus sign, not a dot as in PHP.

You can copy the lines which build up `$output` from 'show-movies.fn.php', changing the concatenation dots to plus signs throughout. So we have `<li`, the opening list item, `title=` and then in single quotes `$description`, concatenated with plus signs:

```
$output = "<li title='" + $description +
```

Then a space, then `id=` and then in quotes `'fav_'` concatenated to `$id`:

```
$output = "<li title='" + $description + "' id='fav_" + $id
```

And finally close off the quotes and the HTML tag, `'>`:

```
$output = "<li title='" + $description + "' id='fav_" + $id + "'>";
```

We concatenate the rest of the hyperlinked list item in the same way, making a link to the currently selected movie using the ID `movie_$id`:

```
$output += "<a href='index.php?user_id=" + $userID
  + "&amp;movie_id=" + $id + "'>";
$output += $title;
$output += "</a></li>";
```

Then we append `$output` to the favourites unordered list, `ul.favs`:

```
$('ul.favs').append($output);
```

We need to change the ID prefix from 'movie_' to 'fav_':

```
$(".favs li#fav_" + $id).draggable({
    helper: 'clone'
});
```

Finally, in ‘show-movies.fn.php’, in `case "favs"`, add the title name ‘description’ to the list item tag we are building up:

```
<li title='$description' id='fav_$id'>";
```

Save and refresh and check that the HTML is error-free. In the browser, refresh the page and try clicking on a list item. The movie in question is added to the favourites list but is not deleted from the main movies list.

In ‘add-favs.js’ we need to change `$this.remove()` to `$this_li.remove()` to remove the whole of the list item.

I have also just noticed an error in the styling. If you click next to the title you are taken to the single movie page, which could be confusing. I would prefer it is just the title text itself were active, not the whole space next to it.

To achieve this, go to ‘style.css’, find the selector `.movie_list h3 a` and change `display: block` to `display: inline-block`.

Save and refresh and that problem is solved – now we have to click on the title text itself to be taken to the single movie page.

That’s the click-to-add interface complete, and in the next lesson we’ll do the more complicated HTML for the drag-to-remove version.

Lesson 102: The drag-to-remove interface

Now for the drag-and-drop effect the job is much the same except that we have to reproduce the more complicated formatting for the main movie list item.

Open up 'remove-favs.js'; we need to set the variables as before.

The currently selected item, the dragged item, is `'ui-draggable'`, so, inside the `droppable()` event, we can set the variable `$this` to the jQuery `$(ui.draggable)`:

```
$('.trash').droppable({
    accept: '.favs li',
    drop: function(event, ui) {
        $this = $(ui.draggable);
...
```

Then we set the ID number as before, using the variable `$this`, not the jQuery selector:

```
$id = $this.attr('id').split("_");
$id = $id[1];
```

Setting `$title` is simpler this time – it's the text of the selected item:

```
$title = $this.text();
```

And `$description` is its title attribute:

```
$description = $this.attr('title');
```

Now we need to recreate the whole of the 'non_favs' list item with its figure, figcaption, image *etc*..

Let's give ourselves a headstart by selecting all of the output variable assignment from 'show-movies.fn.php' and change `.=` to `+=` for Javascript concatenation throughout, except for the first line of the variable assignment, which should be `=`:

```
$output = "<li id='nonfav_" + $id + "'>";
```

We don't need the `\t`s and `\n`s so delete these.

In PHP I decided not to use concatenating dots but to enclose the variables in double quotes, running them together with the plain text, because it made the code easier to read and write.

We can't be lazy in Javascript – we cannot just run the variables and text together in this way– you have to concatenate the variables, using `+` and quotes all the way through.

We also need to change the way we get the image, using `src=`, then the path 'images-movies/', then concatenate the ID number with `+ $id +` and finally append `-tn.png` to it:

```
$output += "<img class='thumbnail' alt='" + $title +
"' src='images-movies/" + $id + "-tn.png'></a>";
```

Once `$output` is built up, change `append()` to `prepend()` and replace the string with `$output` to make the newly added item appear at the top of the list, where it's more obvious:

```
$("ul.non_favs").prepend($output);
```

We also want the trash can to display its hoverstate when we drag a list item into it and to revert to normal when the removal of the favourite is over.

In the `'beforeSend'` function we need to add the class ‘trash_hover’, to the trash div:

```
'beforeSend': function() {
    $this.remove();
    $('.trash').addClass('trash_hover');
}, // End beforeSend
```

and then remove it in the `'success'` function:

```
'success': function() {
    // $output concatenation
    $("ul.non_favs").prepend($output);
    $('.trash').removeClass('trash_hover');
...
```

Make sure the class selector ‘.trash_hover’ is present in ‘style.css’:

```
.trash:hover, .trash_hover {
      background-position: 0 -61px ;
}
```

Save and refresh and now both add and delete interfaces should work.

Lesson 103: Load generic thumbnail image in case of error

Earlier in the course in the chapter on PHP we made sure that if a movie thumbnail image did not exist in the file system, a generic image would be loaded in its place.

We have not dealt with this possibility for items newly added by jQuery, as I will demonstrate.

The ID number of the first movie in the favourites list on the left is 14. If I go to the images-movies folder and rename thumbnail image 14 to something else – anything else, just so the expected file does not exist and then drag that movie item to the trashcan, I get a broken image on the page, and in Firebug we get an error in red.

We can deal with this easily using the event handler `onError`. We do this with the syntax `onError=this.src=` and then the path to the generic thumbnail:

remove-favs.js:

```
$output += "<img class='thumbnail' alt='" + $title +
"' src='images-movies/" + $id + "-tn.png'
onerror=this.src='images-movies/generic-tn.png'></a>";
```

`OnError` loads an alternative image if the first one is not present. It works also for any link where an error is encountered.

Click to add the same movie back to favourites. Save and refresh and drag it back to the trashcan. And now we get the generic thumbnail image. Rename the thumbnail image back to its original name.

Next we need to do the add and remove script for the single movie page. I thought this would make another good assignment for you, so there are instructions on how to proceed in the following lesson and my solution in the one after that.

Assignment 6: The add/remove interface for the single movie page

Write the code, in two new Javascript files, for the add and remove favourite interface on the single movie page.

You will be targetting the div with the class name ‘add_remove’ for both add and delete actions.

The value of `$testFav`, in ‘show-movies.fn.php’, tells us whether the selected movie is currently a favourite or not.

Use this information to add a second class, named either ‘.add’ or ‘.remove’ as appropriate, to the ‘add_remove’ div and then adapt ‘add-favs.js’ for both new scripts, which you should name ‘add-fav-single.js’ and ‘remove-fav-single.js’.

Make the text in the ‘add_remove’ div change as appropriate and the trashcan change to its hover state as an item is removed and back to normal when the removal is completed.

Don’t forget to call the scripts in ‘head.inc.php’ – and Good Luck!

Lesson 105: Add-remove interface for single movie page

This is my solution to the single movie add/remove interface.

In the single movie page we have a single place to click to add and remove that image from favourites. If the movie is currently a favourite, this says “Remove from favourites” and if not it says “Add to favourites”. This changing text is set in ‘show-movies.fn.php’ using the `$testFav` variable.

In ‘show-movies.fnp’, just below the code setting the generic image, add two new `if` clauses, setting a new variable, `$action`, to either ‘add’ or ‘remove’:

```
if ($testFav=="Add to favourites") {
    $action = "add";
}
if ($testFav=="Remove from favourites") {
    $action = "remove";
}
```

Add this variable `$action` as an extra class name on the ‘add_remove’ div, to specify which job it should do according to the state of the data:

```
$output .= "<div class='add_remove $action'>";
```

I have omitted the `\t`s and `\n`s from the example code as they are not important for our examples and just add visual clutter.

Then in the single movie output area, so far we have no ID for the movie so add `$id= single_$id` to the ‘add_remove’ div:

```
$output .= "<div id='single_$id' class='add_remove $action'>";
```

Also, for clarity, I added `class= "title"` to the h3 tag – this is not strictly necessary because it is the only `<h3>` tag on the page – but it makes it easier to read the code:

```
$output .= "<h3 class='title'>$title</h3> ";
```

Then I adapted ‘add-favs.js’, copying it and saving it as ‘add-fav-single.js’. Open ‘add-fav-single.js’ and remove the `toggleClass()` clause – that’s not relevant.

The click selector should be `'.actions .add'`, a parent-child selector:

```
...
$requestRunning = false;
$(document).on('click', '.actions .add', function() {
...
```

As always, set `$this` to the currently selected jQuery object:

```
$this = $(this);
```

Delete the `$this_li` variable assignment.

And as the ID is on the selected div we can get the ID more simply than before from `$this`:

```
$id = $this.attr('id').split("_");
$id = $id[1];
```

I could have just used `'h3'` to get the title but using the name `'title'` is a bit clearer:

```
$title = $('h3.title').text();
```

`$('p.description')` uniquely identifies the description of the selected movie as none of the other descriptions are inside paragraph tags:

```
$description = $('p.description').text();
```

In the `'beforeSend'` function, delete the line in which we remove `$this_li`:

```
'beforeSend': function(){
    $requestRunning = true;
    $this_li.remove();  // Remove this line
}, // End beforeSend
```

We are working on `$this`, and as the 'add-remove' div contains a `<p>` tag, we have to use manipulate its HTML not its text value, and include these paragraph tags:

```
$this.html("<p>Remove from favourites</p>")
```

Then we want to remove the class 'add':

```
$this.html("<p>Remove from favourites</p>").removeClass('add')
```

and add the class 'remove':

```
$this.html("<p>Remove from favourites</p>").removeClass('add')
.addClass('remove');
```

– I hope that's not too confusing!

And then the text will be immediately updated, ready to accept an add or remove click as appropriate.

The remove single favourite script is very similar to this one, so copy 'add-fav-single.js' and save it as 'remove-fav-single.js'.

This script is going to act on the 'remove' class, so change `'.add'` to `'.remove'`:

```
$(document).on('click', '.actions .remove', function() {
...
```

We don't need to know the title or description, so delete those variable assignments:

```
$this = $(this);
$id = $this.attr('id').split("_");
$id = $id[1];$description = $('p.description').text(); // Delete
$title = $('h3.title').text(); // Delete
```

Change the ajax call to 'remove-favs.ajax.php' instead of 'add-favs.ajax.php':

```
$.ajax({
    url: "ajax/remove-favs.ajax.php",
...
```

We want the trash can hover state to appear as the item is being deleted so add the class 'trash_hover' in the `'beforeSend'` callback:

```
'beforeSend': function(){
    $requestRunning = true;
    $('.trash').addClass('trash_hover');
...
```

and remove it in the `'success'` callback:

```
'success': function() {
    $requestRunning = false;
    $('.trash').removeClass('trash_hover');
...
```

This script is removing favourites from the list, so select all the lines where we set up `$output`, append `$output`, and create the clone, and delete them.

Set the HTML of the 'Add/remove' div back to "Add to favourites" in paragraph tags:

```
$this.html('<p>Add to favourites</p>')
```

and change its class back to 'add', ready to receive an 'add' click:

```
$this.html('<p>Add to favourites</p>').
removeClass('remove').addClass('add');
```

Remove the corresponding list item from the favourites list:

```
$("li#fav_" + $id).remove();
```

Finally, I am going to put a comment at the top of all four Javascripts to say what they do.

Add the two new scripts to 'head.inc.php':

head.inc.php:

```
<script src = "scripts/remove-favs.js"></script>
<script src = "scripts/add-favs.js"></script>

<script src = "scripts/remove-fav-single.js"></script>
<script src = "scripts/add-fav-single.js"></script>
```

Save and refresh and try adding and removing single movies and movies from the lists, and check that everything works.

You'll notice as you do this that if you add all the movies to a movie-goer's favourites or remove them all and don't refresh the page, the welcome message at the top doesn't change accordingly and the favourites title and trashcan are always there.

We need to mirror what we did in the PHP to make these change to reflect the state of the data. If I press F5, "Favourites" turns to "You have no favourites" and the trash can disappears, and if all the movies are favourites welcome message also changes to reflect the fact that there is no data below it.

We are very nearly at the end of this chapter, and the main part of the course, so I will set this as a series of major assignments, and will give you detailed instructions and hints in the PDF files followed by my solutions.

Assignment 7: Data dependent headings (Modify PHP)

Our task is to allow for three possible states of the the favourites data:

1) that there are no movies in the favourites list,
2) that all the movies are in the favourites list, or
3) that there are some movies in both lists.

We want to change what is displayed at the top of the page as follows:

1) no movies in favourites list:

Favourites heading:	"You have no favourites"
Trashcan:	Hidden
Welcome message:	"You haven't got any favourite movies yet! Mouse over the movies and click on the heart icon to add them to the list on the left."

2) all movies in favourites list:

Favourites heading:	"Favourites"
Trashcan:	Visible
Welcome message:	"It looks like you love all the movies! Drag them to the trash can to delete them from your favourites."

3) some movies in each list:

Favourites heading:	"Favourites"
Trashcan:	Visible
Welcome message:	"Here are some movies you might like. Click on the heart icon to add them to your favourites list."

These changes should take effect immediately without a page refresh.

Your first task is to modify the PHP in 'main.inc.php' because at present it is only capable of displaying two of these three possibilities.

I would suggest setting default values for the variables and then overriding some of the defaults as necessary.

Lesson 107: Data-dependent headings - Modify PHP

In 'main.inc.php' we have already done some of the PHP work on this, but I only allowed for two states - I did not allow for there being no favourites, so your task was to modify 'main.inc.php' to allow for this. This is my way of doing it.

I set default values first:

```
$welcome = "Here are some movies you might like. ";
```

and concatenated onto that:

```
$welcome .= "Click on the heart icon to add them to your favourites list.";
```

I set the favourites title to 'Favourites':

```
$favsTitle = "Favourites";
```

And didn't add any extra classes to the class 'trashclass':

```
$trashClass = "";
```

And I set the opening HTML tag for the unordered list to a default class of 'non_favs':

```
$openTag = "<ul class='non_favs'>";
```

We didn't need a `$closeTag` variable because the closing tag is always `</ul>`.

I didn't give any value to `$welcomeClass`:

```
$welcomeClass = "";
```

All these default values apply when there are movies in both lists.

Then, using the value of `$favsList` and `$nonfavsList`, I overrode these values as necessary. If `$favsList` is empty, it means there are no favourites. I changed the welcome message, the title of the favourites list, and added the class 'hidden' to the trash can div to hide the trash can:

```
if ($favsList=="") {
    $welcome = "You haven't got any favourite movies yet! ";
    $welcome .= "Mouse over the movies and click on the heart icon to
                 add them to the list on the left.";
    $favsTitle = "You have no favourites";
    $trashClass = "hidden";
    $openTag = "<ul class='non_favs'>";
}
```

If `$nonfavsList` is empty that means that all the movies are in the favourites list. Again, I changed the welcome message to make it appropriate, and this time added the class 'hidden' to the non-favourites list to hide the main list of movies. I also set `$welcomeClass` to the style 'no_border_bottom' to avoid the problem we saw earlier with the doubled bottom border:

```
if ($nonfavsList=="") {
    $welcome = "It looks like you like all the movies! ";
    $welcome .= "Drag them to the trash can to delete them from your
                 favourites.";
    $openTag = "<ul class='non_favs hidden'>";
    $welcomeClass = "no_border_bottom";
}
```

Save and refresh and choose movie-goers with all favourites, no favourites, and some of each. Now we have all three sets of output correct whatever the state of the favourites is, but a page refresh is necessary to see them.

For your next assignment, try to mirror this effect in jQuery. There are instructions and some new code on the next page.

Write the code to create an identical effect using jQuery so that the favourites heading, trashcan, and welcome message change without a page refresh according to whether a movie-goer has no favourites, all favourites, or some of each.

Here are some hints to get you started.

First, a more elegant way of changing the welcome message than including the long strings directly in jQuery is to set three styles in the stylesheet, `.like_some`, `.like_all`, and `.like_none`, and use the `:after` pseudo-selector to set the content of the paragraphs like this:

```
.like_all:after {
    content: "It looks like you love all the movies(...)";
}
```

and then use `addClass()` and `removeClass()` in jQuery.

Second, you can tell whether there are any favourites using the Javascript length property (*e.g.* `$('.favs li').length` and set the various strings and styles on the basis of that.

Third, the same code will work unaltered in both 'remove-favs.js' and 'add-favs.js'; a simplified version of it with only a deletion will work in 'add-fav-single.js'; and a further simplified version of that, again with only a deletion, will work in 'remove-fav-single.js'.

Good luck!

Lesson 109: Data-dependent headings- jQuery version

This is my solution to getting the data-dependent headings to come into effect without a page refresh, using jQuery. We could put the various welcome messages in the text attribute in jQuery but as its rather long it's neater to use classes and then add and remove the classes in jQuery as we did earlier.

In 'style.css', add three new class selectors: `.like_none`, `.like_some`, and `.like_all`.

We can set the text each of these displays using the pseudo-element `:after`, and set the `content` to the text that we want to display:

style.css:

```
.like_none:after {
    content: "You haven't got any favourite movies yet! Mouse over the
movies and click on the heart icon to add them to the list on the left.";
}

.like_some:after {
    content: "Here are some movies you might like. Click on the heart icon
to add them to your favourites list.";
}

.like_all:after {
    content: "It looks like you love all the movies! Drag them to the
trash can to delete them from your favourites.";
}
```

We'll start the jQuery in 'remove-favs.js'.

These changes need to go in the `'success'` callback, so they kick in as the database update completes. We'll do the equivalent of what we did in PHP, setting default values, for when there are movies in both lists, and then selectively overriding them to deal with the other possibilities.

We need to remove the text already set in the welcome message by the PHP, or we will get both messages at once:

```
$('p.welcome').text("")
```

We want to remove the classes 'like_all', 'like_none' and 'no_border_bottom' – all the non-default classes:

```
$('p.welcome').text("")
.removeClass('like_all like_none no_border_bottom')
```

and add the default class, 'like_some':

```
$('p.welcome').text("")
.removeClass('like_all like_none no_border_bottom')
.addClass('like_some');
```

Using a comma-separated list of class names, we want to remove the class 'hidden' from 'trash', 'favs' and 'non_favs':

```
$('.trash, .favs, .non_favs').removeClass('hidden');
```

And we want to select the `<h2>` inside the 'favs_list' and set its text to the default, "Favourites":

```
$('.favs_list h2').text("Favourites")
```

Those are the defaults, which we'll now override in the cases of there being no favourites or no non-favourites.

If there are no favourites, the length of the favourites list item output by the function will be zero because the PHP loop does not run. We can use this fact to set the output for this case:

```
if ($('.favs li').length == 0) {
    // Set output for case of no favourites
}
```

In that case, remove the welcome text:

```
if ($('.favs li').length == 0) {
    $('p.welcome').text("")
}
```

and remove the class 'like_some':

```
if ($('.favs li').length == 0) {
    $('p.welcome').text("").removeClass('like_some')
}
```

and add in its place the class 'like_none':

```
if ($('.favs li').length == 0) {
    $('p.welcome').text("")
    .removeClass('like_some').addClass('like_none'):
}
```

Using another comma-separated list of selectors, hide both the trash can and the favourites list:

```
...
    $('.trash, .favs').addClass('hidden');
...
```

Select the favourites list heading and change its text to "You have no favourites":

```
...
    $('.favs_list h2').text("You have no favourites");
...
```

Select the non-favourites class and remove the class 'hidden' from it to show it:

```
$('.non_favs').removeClass('hidden');
```

If there are no items in the non-favourites list, the length of that list item will be zero:

```
if ($('.non_favs li').length == 0) {
}
```

As before remove the welcome text set in the PHP, remove the 'like_some' class from it, and add the two classes 'like_all':

```
if ($('.non_favs li').length == 0) {
    $('p.welcome').text("").removeClass('like_some').addClass('like_all');
}
```

When the non-favourites list is empty we need to remove the bottom border from the welcome paragraph, which we do by adding the class 'no_border_bottom':

```
if ($('.non_favs li').length == 0) {
    $('p.welcome').text("")
    .removeClass('like_some').addClass('like_all no_border_bottom');
}
```

Remove the hidden style from both the favourites list and the trash can:

```
$('.trash, .favs').removeClass('hidden');
```

and add it to the ‘non_favs’ list to hide that:

```
$('.non_favs').addClass('hidden');
```

Set the favourites title back to ‘Favourites’:

```
$('.favs_list h2').text("Favourites")
```

That’s all we need.

Copy that and paste it in at the end of the `'success'` callback in ‘add-favs.js’ – it needs no changes.

To get it to work in the single movie version, first copy it all into the end of the `'success'` callback in ‘add-fav-single.js’ and then remove the lines setting the welcome messages – they are not present in the single movie page.

And then copy that into ‘remove-fav-single.js’ and remove the if clause and its contents setting the variables when the ‘non_favs’ list length is zero – there’s no such list on this page.

Save and refresh and try the various states: no favourites, all favourites, and some favourites and check the welcome message, favourites title, and trash can.

In the next lesson we will add visual enhancements, including a highlight state for newly added favourites, a ‘success’ icon for movies just added to the main list, and an AJAX loader icon to show there is a database update in progress.

Lesson 110: Visual enhancements and AJAX loader icon

It's important in AJAX-powered websites to provide visual feedback that a database update has actually taken place, because otherwise it can escape the user's attention.

In this lesson we'll put in a little success icon, replacing the heart icon, when a movie has just been added to the non-favourites list, we'll highlight newly added favourites with a different color, and we'll put in an AJAX loader icon as the database update takes place, dimming the rest of the page at the same time.

We'll do the success icon first.

The style needs to be altered slightly. In 'style.css', change the selector '.success' to a parent-child selector '.non_favs .success'. Remove its left margin and change its width and height to 20px:

```
.non_favs .success {
    width: 20px;
    height: 20px;
}
```

Put the background image in a new '.success' selector of its own and remove it from '.non_favs .success' and '.admin_table .success':

```
.success {
    background-position: -42px -177px;
}
```

In 'remove-favs.js', just after the `prepend()` line in the `'success'` callback, create a new variable, `$this_added`, and set this to the value of the curently active 'add' div:

```
$("ul.non_favs").prepend($output);
$this_added = $('li#nonfav_' + $id + ' .add');
```

On the next line, using the variable `$this_added` as the jQuery selector, `$($this_added)`, remove the class 'favourite' from this and add the two classes 'add' and 'success':

```
$this_added = $('li#nonfav_' + $id + ' .add');
$this_added.removeClass('favourite').addClass('add success');
```

We need the class 'add' for positioning and 'success' for the image.

Save and refresh. The success icon appears – but it stays there in a rather ugly way. It would be better if it disappeared when we mouseover any of the 'non_favs' list items.

Set a mouseover function on all 'non_favs li' items:

```
...
$this_added.removeClass('favourite').addClass('add success');
$('.non_favs li').mouseover(function(){
});
```

Within that, we want to remove the class 'success' wherever it exists:

```
...
$('.non_favs li').mouseover(function(){
    $('.success').removeClass('success');
});
```

Save and refresh. Now if we move the mouse over any of the other non-favourites list items, the 'success' icon disappears immediately

At the moment if we keep on dragging items to the trash can and don't refresh the page, we'll get a series of 'success' icons. We want it to be possible to have only one 'success' icon at a time. We can achieve this by adding that same jQuery command in the `'beforeSend'` function so that when the database is initiated any success icons are removed from the page:

```
'beforeSend': function() {
    $this.remove();
    $('.trash').addClass('trash_hover');
    $('.success').removeClass('success');
    ...
```

Now we'll highlight newly added favourites list items. The process for this is very similar.

Add a new parent-child class selector, '.favs .highlight', to 'style.css':

```
.favs .highlight {
      background-color: orange;
      border: 1px solid red;
}
```

or any other colours you prefer.

And then in 'add-favs.js', in the `'success'` callback just under the cloning function, make a new variable called `$this_added` again and set this to the value of the currently active favourites list item:

add-favs.js:

```
$this_added= $('li#fav_' + $id);
```

Add the class 'highlight' to it:

```
$this_added.addClass('highlight');
```

As before, prevent this highlight from remaining when the list is moused over by adding another moseover function which takes effect on the favourites list items and removes the class 'highlight', wherever it exists:

```
$('.favs li').mouseover(function(){
  $('.highlight').removeClass('highlight');
});
```

And we can prevent more than one highlighted item appearing at a time in the same way as we did before. Copy that line and paste it into the `'beforeSend'` callback, just below the line `$this_li.remove()`:

```
$this_li.remove();
$('.highlight').removeClass('highlight');
```

Now in the `'success'` function, copy the five lines we have added in this lesson, and paste these into 'add-favs-single.js', just below the 'helper clone; function in the `'success'` callback:

```
$this_added = $('li#fav_' + $id);
$this_added.addClass('highlight');

$('.favs li').mouseover(function(){
```

```
    $('.highlight').removeClass('highlight');
});
```

Save and refresh and click on a single movie. Click on “Add to favourites” and we should get the highlight state. When we mouseover it, it should disappear. Click on a movie-goer and add a movie and again that highlight state should appear and disappear if we mouse over it.

Now we’ll put in an AJAX loader icon.

Copy ‘loader-large.png’ from the ‘NEW LARGE LOADER ICON’ folder in the Working Files for this chapter to the ‘images’ folder in ‘favouritemovies’.

In style.css define a new class selector ‘.loader_large’ and give it a background image using the relative path to the image file. Give it absolute positioning, set that to left 500px, top 430px. Give it a height and width of 50px:

```
.loader_large {
   background-image: url('../images/loader-large.png');
   position: absolute;
   left: 500px;
   top: 430px;
   height: 50px;
   width: 50px;
}
```

In ‘main.inc.php’ after the closing section in two places, on the line below the closing `</section>` element in cases ‘no_id’ and ‘id_set’, add:

```
echo "<div class='loader_large hidden'></div>";
```

adding this loader icon as a hidden div which we’ll reveal using jQuery.

In all four Javascript scripts in the `'beforeSend'` callback, add a line removing the ‘hidden’ class:

```
$('.loader_large').removeClass('hidden');
```

so that when the database update starts, the loader icon is visible and is revealed during the AJAX call.

In the `'success'` callback in all four scripts, add its counterpart:

```
$('.loader_large').addClass('hidden');
```

To hide the loader icon once the update is complete.

Save and refresh, and each time we perform a database update we can see the loader icon is appearing, but it's not very clear. It would look better if everything on the page except the loader icon dimmed during the update. We can achieve this using the CSS property ‘opacity’.

In ‘style.css’, add a new class selector, ‘.dim’ and set its opacity to 0.5:

```
.dim {
    opacity: 0.5;
}
```

Opacity ranges from 0 to 1, 0 being invisible and 1 being normal, 100% opaque. 0.5 sets it to 50% opacity. You can experiment with this value to find the level you prefer.

In all four scripts in the ‘beforeSend’ callback add the 'html' selector:

```
$('html')
```

filtered by the `not()` selector:

```
$('html').not('.loader_large')
```

`$('html')` selects all the HTML on the entire page, and then the selector `not()` removes items from the selection, so in this case everything is selected except the div containing the loader icon.

To that, in the `'beforeSend'` callback, add the class ‘dim’:

```
$('html').not('.loader_large').addClass('dim');
```

and in the `'success'` callback add its counterpart:

```
$('html').not('.loader_large').removeClass('dim');
```

Copy that and paste it into all four scripts, near the other lines to do with the loader icon to keep related things together.

Save and refresh. That’s all fine and we’re ready to do the admin pages now.

Assignment 9: Visibility of admin and movie-goer menus

At the moment only the movie-goers navigation menu is visible – the admin menu is hidden in 'style.css', so we have no way of navigating to the admin pages.

It is a relatively simple task to write a script to make each of these menus appear when we mouse over it, but we also have to make sure that the menus do not disappear too easily before a user has managed to click on one of the items.

Try to write the code, in a new script called 'navigation.js', to achieve this. Don't forget to include the link to this script in 'head.inc.php'.

As always, good luck!

Lesson 112: Control visibility of admin menus

Start by putting in a link to the new script, 'navigation.js', in 'head.inc.php':

```
<script src="scripts/navigation.js"></script>
```

In 'style.css', remove `display: none` from the selector `ul.admin_menu` – that was there merely for convenience as we were working on the site.

We want both menus to be invisible when the page is loaded, so add the class 'hidden' to `<ul class='admin_menu'>` in 'navigation.inc.php':

navigation.inc.php:
```
echo $usersList;

echo " <ul class='admin_menu hidden'>";
```

And add the class 'hidden' to `<ul class='users_menu'>` in 'show-users.fn.php':

show-users.fn.php:
```
$output = "<ul class='users_menu hidden'> ";
```

Adding the class 'hidden' in the PHP files instead of relying on jQuery means that this will work even if Javascript is not enabled in the browser, which is what we want – otherwise a user without Javascript would see both menus on top of each other.

Make the new Javascript file, 'navigation.js', add the `document ready` function, and put in a comment saying what this script does:

```
// Shows and hides admin menus on mouseover

$(document).ready(function(){
});
```

Then, upon document ready, hide both the user_menu and the admin_menu so neither of the menus appear:

```
$(document).ready(function(){
    $('.users_menu, .admin_menu').addClass('hidden');
});
```

We want the users_menu to be visible when the mouse is over the 'select_users' class or over the users menu itself. Mousing over these should remove the 'hidden' class from the users menu:

```
$('.select_users, .users_menu').mouseover(function(){
    $('.users_menu').removeClass('hidden');
});
```

and at the same time add it to the admin menu, hiding that:

```
$('.select_users, .users_menu').mouseover(function(){
    $('.users_menu').removeClass('hidden');
    $('.admin_menu').addClass('hidden');
});
```

Then the reverse for the admin button and the admin menu:

```
$('.admin_button, .admin_menu').mouseover(function(){
    $('.admin_menu').removeClass('hidden');
    $('.users_menu').addClass('hidden');
});
```

Save and refresh and try it and the menus should first be hidden and they should appear when we mouseover them or their headings.

At present the menus remain visible even when we have moved the mouse off them.We need a mouseout function to hide them. We can do this in one go using two class selectors separated by commas and the `$(this)` selector, which will apply to whichever one is active:

```
$('.users_menu, .admin_menu').mouseout(function(){
    $(this).addClass('hidden');
});
```

Save and refresh and the menus now appear and disappear as intended.

Now we want to go on to the admin pages, but before we do that there is one piece of housekeeping we need to do.

At present, in ‘head.inc.php’ we load all the Javascript files regardless of whether or not they are required for the particular page we are viewing.

It would be more efficient if we put these links to the various scripts in `if` conditions so as to run only those needed at a paricular moment. This would make a nice assignment for you.

Assignment 10: Conditional loading of Javascript files

At present, all the Javascript files are loaded in 'head.inc.php'. This is inefficient and could potentially cause errors.

Modify 'head.inc.php' so that only those Javascript files which are needed for the current page view (*e.g.* movie list, single movie, user admin, *etc.*) are loaded.

Lesson 114: Conditional loading of Javascript files

This is my way of loading only the Javascript files needed for each particular page.

Open up 'head.inc.php'. The link to 'navigation.js' must always be there, so we keep this out of the conditions which follow. Then I opened PHP and tested whether `$userID` is set:

```
<script src="scripts/navigation.js"></script>

<?php
if (isset($userID)) {
    // Must be a movie page not an admin page
}
?>
```

Then I close off the PHP tags, go back to plain HTML, and open the PHP tags again later.

If `$userID` is set, then we know that we are looking at a movie page, not an admin page, and we need to load the one-line script that echoes `$userID` into Javascript and we need 'remove-favs' and 'add-favs.js':

```
<?php
if (isset($userID)) { ?>
    <script>var $userID  = "<?php echo $userID; ?>"</script>
    <script src = "scripts/remove-favs.js"></script>
    <script src = 'scripts/add-favs.js'></script>
<?php }
?>
```

Then within that condition, we further test to see if `$movieID` is set, and if it is then we know we are looking at the single movie page:

```
if (isset($userID)) {
    // As above – must be a movie page

    if (isset($movieID)) {
        // Must be the single movie page
    }
}
```

And we load 'remove-fav-single.js' and 'add-fav-single.js':

```
if (isset($userID)) {
    // As above – must be a movie page

    <?php
    if (isset($movieID)) { ?>
       <script src = 'scripts/remove-fav-single.js'></script>
       <script src = 'scripts/add-fav-single.js'></script>
    <?php }
}
```

That's as far as we've got so far, but we'll put in the next condition in place, testing whether or not `$page` is set, and if it is then we're viewing one of the admin pages and we should load the admin scripts that we're about to write:

```
if (isset($page)) { ?>
    // Must be admin page
<?php } ?>
```

Lesson 115: Movie-goer deletion interface – preliminaries

For your next assignment, I want you to write all the code for the interface we'll use to delete movie-goers from the database.

Looking at the finished version, the aim is to be able to click on the 'delete' icon, which appears only when we mouseover it, and the movie-goer will be deleted immediately.

As the database updates a small loader icon should appear in the table-cell in place of the delete icon.

In a real-world application, you might want to put in a confirmation prompt before the deletion takes place, but for this AJAX demo, in which the data is of little importance, I've opted for a more rapid interface.

Before you go on to writing the Javascript and AJAX scripts to achieve this, there are a couple of housekeeping jobs to do.

Click on the menu to go to the users admin page.

Click 'View page source' – the HTML for the admin table is badly formatted - I have missed off the tabs and new lines.

Open 'show-users.fn.php', and go down to the 'admin' `case`. We need six tabs, `\t`, in front of each of the `<td>`s and a new line, `\n`, after each of them, and five tabs before the closing `</tr>` and a new line after it.

And in 'admin-users.inc.php', put in four tabs before `<h2>Manage users</h2>` and a new line after it.

Refresh the source – now we can see what we're doing.

In 'head.inc.php', inside the condition we just set, add a link to the new Javascript file you are about to write. Call it 'delete-user.js':

```
if (isset($page)) { ?>
    <script src = "scripts/delete-user.js"></script>
<?php } ?>
```

Now create 'delete-user.js' in the 'scripts' folder.

Now look in the HTML source – our click is to take place on the 'delete' div.

So far, we have no ID number – we need to set this. We'll use 'user_' as the ID prefix.

In 'show-movies.fn.php', add `id = user_` and the user ID on the opening `<tr>` element:

```
$output .= "\t\t\t\t\t<tr id='user_$id' class='datarow'>\n";
```

Save and refresh and check the HTML source is OK.

You will also need the small loader icon, which is in the Working Files for this chapter in the 'NEW SMALL LOADER ICON' folder.

In 'style.css', you will also need to make a new class selector, '.loader_small', with the following properties:

```
.loader_small {
   background-image: url('../images/loader-small.png');
   margin-left: 40px;
   width: 25px;
   height: 25px;
}
```

That’s the preliminaries done.

No new knowledge or techniques are required to write the AJAX code needed to delete a user from the database, although there is a trick to watch out for.

Nothing new is needed for the Javascript to manage the interface, hiding the delete icons until the mouse moves over the ‘deletecell’, deleting the affected line from the table and inserting the animated loader icon in place of the ‘delete’ icon as the database update takes place.

There are instructions in the next lesson. Do your best at writing this yourself as it gives practice in putting together all the skills we’ve been working on.

Assignment 11: Movie-goer deletion interface

The movie-goer deletion interface needs no new techniques – just correct application of what we've done so far.

There is just one important trick to take note of when you are writing the PHP file called in AJAX – I will give you the hint that if you do not notice this, you will soon end up with a very messy database, with lots of irrelevant records in it.

Good luck!

Lesson 117: Movie-goer deletion interface

This is my solution for the movie-goer deletion interface.

Looking first at the AJAX script, 'delete-user.ajax.php', this is very familiar – we delete from movie-goers where user_id = ? and then bind the parameter, execute and close the prepared statement:

delete-user.ajax.php:

```
$stmt = $db->prepare("DELETE FROM movie_goers WHERE user_id = ?");
$stmt->bind_param('i', $userID);
$stmt->execute();
$stmt->close();
```

The trick is that if you leave it at that, you will not be deleting the corresponding records in the 'favourites' table and you will end up with meaningless, orphan, records.

You need to do the same thing on the 'favourites' table as well:

```
$stmt = $db->prepare("DELETE FROM favourites WHERE user_id = ?");
//As above
```

Then before doing the jQuery file, in 'show-users.fn.php', I first added the class 'hidden' to the 'delete' div, so that I could use the same method as before to show the delete icon only on mouseover:

show-users.fn.php:

```
...
case "admin":
    $output .= " <td class='deletecell'>
    <div class='delete hidden'></div></td>\n";
    ...
```

Then the jQuery code.

I used exactly the same `toggleClass()` effect to show and hide the delete icon:

```
$(document).on('mouseover mouseout', '.deletecell', function(){
    $('.delete', this).toggleClass('hidden');
});
```

Using document – on – click, I set the variable `$this` as usual:

```
$(document).on('click', '.delete', function(){
    $this = $(this);
```

And got the current ID from the closest `<tr>`, tablerow:

```
$(document).on('click', '.delete', function(){
    $this = $(this);
    $id = $(this).closest('tr').attr('id').split("_");
    $id = $id[1];
```

I called the AJAX script and in the `'beforeSend'` function I removed the class 'delete' and added the class 'loader_small' to show the loader icon:

```
'beforeSend': function() {
    $this.removeClass('delete').addClass('loader_small');
},
```

In the `'success'` function I removed the currently selected table row:

```
'success': function(result) {
    $('tr#user_' + $id).remove();
},
```

Now we'll try this out. Open up phpMyAdmin, and reinitialise the database. Have a look in the database and in the 'movie_goers' table we can see that Mr Movieman has user ID no. 9. If we look in the favourites table we will find that he has a lot of favourite movies. Now try out the interface by deleting Mr Movieman and checking that the records are deleted in the database – and indeed, user ID 9 has gone from both tables, so we know what works.

And that's it.

We can now do exactly the same for the delete movies interface.

Lesson 118: The movie deletion interface

I'll go through this quickly as the procedure is identical to the delete-user interface. First the HTML needs minor tidying. In 'admin-movies.inc.php', put four tabs `\t` in front of `<h2>Manage movies</h2>` and a new line `\n` at the end.

Below that, find `<th class='data_col'>Description</th>` and add the class 'description':

```
<th class='data_col description'>Description</th>
```

In 'show-movies.fn.php' add a new line at the end of the closing tablerow tag: `</tr>\n`.

Still in 'show-movies.fn.php', in `case: admin`, add the IDs:

```
case "admin":
    $output .= "\t\t\t\t\t<tr id='movie_$id' class='datarow'>\n";
...
```

And add the class 'hidden' to the 'delete' div:

```
$output .= "\t\t\t\t\t\t<td class='deletecell'>
           <div class='delete hidden'></div></td>\n";
```

Copy 'delete-user.ajax.php' and rename the copy 'delete-movie.ajax.php'. In 'delete-movie.ajax.php', change every occurrence of 'user' to 'movie'.

In 'head.inc.php', change "if isset page" to "if $page==users" and "if $page==movies" and add the links to the appropriate Javascript files:

```
if ($page=="users") { ?>
    <script src = "scripts/delete-user.js"></script>
<?php }

if ($page=="movies") { ?>
    <script src = "scripts/delete-movie.js"></script>
<?php } ?>
```

Copy 'delete-user.js' and rename the copy 'delete-movie.js'. In 'delete-movie.js', change 'user' to 'movie' throughout. Double-check it; there are three occurrences, in the PHP file name:

```
...
    url: "ajax/delete-movie.ajax.php",
...
```

in the AJAX data:

```
...
    data: {
        'movie_id': $id
...
```

and in the jQuery `remove()` line:

```
...
    $('tr#movie_' + $id).remove();
...
```

Try it out ...

The visual effects work, but the database deletion does not take place. Can you see what is wrong?

In ‘delete-movie.ajax.php’, change the table name from ‘movie_goers’ to ‘movies’:

```
$stmt = $db->prepare("DELETE FROM movies WHERE movie_id = ?");
```

Save and refresh and try it again. Press F5 and this time the data has been deleted from the database.

Reinitialise the database in phpMyAdmin.

Next we’ll do the very similar job of coding the interfaces to insert new movie-goers and movies.

Lesson 119: Add new user interface (1)

The code for adding new movie-goers to the database is a bit more complicated than anything we've done so far.

Before we begin, let's look at the completed version to see in more detail at what we want to achieve.

When we enter a name in at least one of the name fields and click on the 'insert' icon, the loader icon appears, the record is inserted into the database, and it moves up one row in the table, leaving the bottom row empty again for the next 'insert' action.

Immediately a new record is inserted, we are able to delete it using the delete interface.

The delete, add, success and loader icons all have to appear and disappear at the appropriate moments.

If the user menu is visible when a database update takes place, this will be updated immediately with inserted and deleted records as well.

If we don't enter any names at all and try to add a record, a dialog will alert us and no empty record will be added.

By the way, from now on we will be adding, deleting and altering records repeatedly as we test the website. You'll need to reinitialise the database as often as you need – I won't mention this every time.

If you want to have a go at this, by all means go ahead – it will be very good practice at thinking all the way through a puzzle. In the end the Working Files are there if you need them, or alternatively uou can continue with my explanations.

The PHP file starts out much the same as before, so copy 'delete-user.ajax.php' and rename the copy 'add-user.ajax.php'.

In 'add-user.ajax.php', the data fields are `$firstname` and `$lastname`, so edit the variable assignments to read:

```
$firstname = $_POST['firstname'];
$lastname = $_POST['lastname'];
```

This is an `INSERT INTO` operation:

```
$stmt = $db->prepare("INSERT INTO movie_goers (firstname, lastname)
                      VALUES (?, ?)");
```

The data types are both strings:

```
$stmt->bind_param('ss', $firstname, $lastname);
```

After executing and closing the prepared statement, delete the other one (deleting from favourites) as this is not relevant.

In the 'scripts' folder, create the new Javascript file, 'add-user.js'.

And add a link to it to 'head.inc.php', inside the `if page == users` condition:

```
if ($page=="users") { ?>
    <script src = "scripts/delete-user.js"></script>
    <script src = "scripts/add-user.js"></script>
<?php }
```

Open it up and insert the comment and document ready function:

add-user.js:

```
// Inserts a new movie-goer into the database

$(document).ready(function(){
});
```

Let's get the basic data entry working first and think about the other things later.

Looking at the HTML source, the bottom row of the table is already differentiated from the other rows with different classnames, 'insertcell' and 'insert' instead of 'deletecell' and 'delete', so we'll be using these as the selectors.

In 'admin-users.inc.php', add the class 'hidden' to the insert div, as we did before, so that the insert icon does not appear until jQuery does something to make it appear:

```
echo "\t\t\t\t\t\t<td class='insertcell'>
      <div class='insert hidden'></div></td>\n";
```

We need to see the insert icon when the focus, be it the mouse or the cursor, is anywhere on the newdatarow, so for our jQuery object, we'll use document – on – focus, working on the 'newdatarow' selector:

```
$(document).on('focus', '.newdatarow', function() {
});
```

Which we do by removing the 'hidden' class on it.

```
$(document).on('focus', '.newdatarow', function() {
    $('.insert').removeClass('hidden');
});
```

Start the on – document – click function, working on the 'insert' class, and set `$this` as before to the element clicked:

```
$(document).on('click', '.insert', function(){
    $this = $(this);
});
```

Now we want to get the value of the names entered in the input boxes. We can do this using the jQuery method `val()`, for value.

Our selector is an input box, its class name is 'newdata' and we want its `val()`:

```
$(document).on('click', '.insert', function(){
    $this = $(this);
    $firstname = $('input.newdata')
});
```

We can distinguish the two `input`s by specifying their names in the selector, with square brackets and `name=` and its name in the HTML:

```
...
    $firstname = $('input.newdata[name="firstname"]')
...
```

And finally we want its value:

```
...
    $firstname = $('input.newdata[name="firstname"]').val();
...
```

Copy that and change ‘firstname’ to ‘lastname’ to get the `$lastname` variable:

```
...
    $firstname = $('input.newdata[name="firstname"]').val();
    $lastname = $('input.newdata[name="lastname"]').val();
...
```

Let’s put in an `if` condition here to catch the possibility that both `$firstname` and `$lastname` are empty, *i.e.* that nothing has been entered:

```
if ($firstname=="" && $lastname=="") {
});
```

The Javascript `alert()` function pops up a dialog box with the message in the parentheses and an OK button:

```
if ($firstname=="" && $lastname=="") {
    alert("Please enter at least one name");
});
```

`Return` stops the function so no completely empty record can be inserted:

```
if ($firstname=="" && $lastname=="") {
    alert("Please enter at least one name");
    return;
});
```

This allows one of the variables to be empty but not both.

Now we know there is some data, we can start the AJAX call with its three main options, `url`, `type`, and `data`:

```
$.ajax({ // Begin insert ajax
    url: "ajax/add-user.ajax.php",
    type: "POST",
    data: { // begin insert data
            'firstname': $firstname,
            'lastname': $lastname
    }
});
```

Then the `'beforeSend'` callback:

```
$.ajax({ // Begin insert ajax
    url: "ajax/add-user.ajax.php",
    type: "POST",
    data: { // Begin insert data
            'firstname': $firstname,
            'lastname': $lastname
    },
```

```
    'beforeSend': function() {
    },

});
```

When the AJAX call starts, we want the loader icon to appear in place of the ‘insert’ icon, so we remove the ‘insert’ class and add the ‘loader_small’ class:

```
...
    'beforeSend': function() {
        $this.removeClass('insert').addClass('loader_small');
    },
```

And we also want to do as we did before and make it impossible for more than one success icon to appear at once, by removing any existing success icon when a new AJAX call starts:

```
...
    'beforeSend': function() {
        $this.removeClass('insert').addClass('loader_small');
        $('.success').removeClass('success insert');
    },
```

Let’s pause there and see if that works so far. Enter a name and click the insert icon.

Of course the page is not updated and the loader icon is not removed as we haven’t done that yet, but if we press F5 we can see that the record has been successfully inserted.

That’s enough for one video. I’ll stop here and we’ll do the `'success'` callback in the next lesson.

Lesson 120: Add new user interface (2)

Now for the `'success'` callback, where we set up the events to happen once the database update has completed.

So as to avoid typing over the inserted name each time, we'll remove the value from the input boxes as soon as the database update is complete by setting the value of the two input boxes to empty. We don't need to distinguish them:

```
...
'success' : function() {
     $('input.newdata').val('');
}
...
```

Then we'll add the new table row with the new record in it. We'll do as we did before and build up a variable called `$output`. You can copy and paste the string to add from 'show-users.fn.php', removing the tabs and newlines and changing `.=` to `+=` again, and concatenating the text and variables with plus signs and double quotes:

```
...
'success' : function() {
     $('input.newdata').val('');

     $output = "<tr id='user_$id' class='datarow'>";
     $output += "<td><input class='data' type='text' name='firstname'
            value='" + $firstname + "'></td>";
     $output += "<td><input class='data' type='text' name='lastname'
           value='" + $lastname + "'></td>";
     $output += "<td class='deletecell'><div class='delete'></div></td>";
     $output += "</tr>";
...
```

For the moment we need to remove `id='user_$id'` as this is a newly inserted record and we have no way of knowing the value of its user ID yet:

```
$output = "<tr class='datarow'>";
```

We want to add this to the table in such a way that it jumps up once row, leaving the bottom row empty again. Like many things, jQuery makes it easy to do this.

Select the last of the table rows:

```
$('table tr:last')
```

and then insert `$output` before it using the jQuery `before()` function:

```
$('table tr:last').before($output);
```

Save and refresh and add a new record. When you click 'insert' it will jump up one row in the table, exactly as we want.

But the loader icon is never going to disappear, so we need to remove that and replace it with the success icon. It is now on the last of the 'delete' divs' so we can target it with the `:last` pseudo-selector. Remove the class 'loader_small', and add the class 'success':

```
$('.delete:last').removeClass('loader_small').addClass('success');
```

At the same time we need to remove the loader icon and show the insert icon on the clicked div, `$this`, ready for another insert:

```
$('.delete:last').removeClass('loader_small').addClass('success');
$this.removeClass('loader_small').addClass('insert');
```

We'll use `mouseover` to hide both the 'hidden' and 'success' classes on the 'deletecells':

```
$(document).on('mouseover', '.deletecell', function(){
    $('.delete', this).removeClass('hidden success');
});
```

Save and refresh and now if we add a new record – click on it – the loader icon disappears, and the 'delete' icon appears when we mouseover it where the 'success' icon used to be.

Lastly, we want to update the user menu on the fly as well. We can use the same variable name, `$output`, overriding its previous value,

And we can copy the value string from 'show-users.fn.php' again and make the necessary changes, changing `.=` to `+=`, change `.=` on the first line to `=`, removing `\t`s, changing `<$tag>` variable to `<li>` , put the link to 'index.php', and change `</$tag>` to `</li>`. It should look like this:

```
$output = "<li>";
$output += "<a href='index.php'>";
$output += $firstname + " " + $lastname + "</a>";
$output += "</li>\n";
```

And then append `$output` to the users menu:

```
$('.users_menu').append($output);
```

Save and refresh and try it, and the user is added both in the table and in the users menu at once. But in the user's menu it's not in order. Can you work out how to solve this problem and insert new records into this list in alphabetical order?

The answer lies in the script we used earlier, 'tinysort'. Copy it from Working Files 'scripts' folder to 'favouritemovies/scripts'. Add a link to it in 'head.inc.php' just under the link to 'navigation.js', so that it is available throughout the project:

```
<script src="scripts/navigation.js"></script>
<script src="scripts/jquery.tinysort.min.js"></script>
```

Back in 'add-user.js', we can now use the `tsort()` function, working on the users menu list item, and that will sort the listinto alphabetical order:

```
$('.users_menu li').tsort();
```

Save and refresh, and insert some new records, and indeed they are now in order in the menu list.

But our work isn't over yet – if we try to delete newly added records, before refreshing the page, it doesn't work. To do this, we need to have a way of distinguishing the various records just added, using their ID numbers. In the HTML source we can see that newly inserted records, unlike the others, have no ID numbers.

The ID numbers have been automatically assigned by the database and at the moment we don't know what they are. That's for the next lesson.

Lesson 121: Delete newly inserted database records

To be able to delete records which have only just been inserted into the database, before a page refresh gets the full details of the record, we need to get the newly assigned ID number back from the database and insert this into the DOM using jQuery.

The ID number is set in the database to auto-increment, so each time a new record is inserted, the value of the ID number increases by one. We're interested in the most recently inserted record each time, which will always be the record with the highest ID number.

We can get this highest value in 'add-user.ajax.php', and return it to the calling script. Open up 'add-user.ajax.php'. We can get the highest value in a field using the MYSQL command `SELECT MAX()`, with the fieldname that we're looking in in parentheses:

```
SELECT MAX(user_id)
```

We then assign this to a fieldname with `AS user_id`:

```
SELECT MAX(user_id) AS user_id
```

We could use any name for this but we'll use the same name for clarity.

And then we specify the table that we are looking in:

```
SELECT MAX(user_id) AS user_id FROM movie_goers
```

We do not need a prepared statement here as there is no user input.

We assign all of that to a variable which I have named `$maxIdSQL`:

```
$maxIdSQL = "SELECT MAX(user_id) as user_id FROM movie_goers";

$maxIdResult = $db->query($maxIdSQL);
$maxIdNumrows= $maxIdResult->num_rows;
```

Then we get the results by looping through using `fetch_object()`:

```
while ($row = $maxIdResult->fetch_object()) {
}
```

assigning the result to the variable `$userID`,

```
while ($row = $maxIdResult->fetch_object()) {
    $userID = $row->user_id;
}
```

And finally assign `$userID` to `$result` and return the value we have got as `$result` to the calling script:

```
while ($row = $maxIdResult->fetch_object()) {
    $userID = $row->user_id;
    $result = $userID;
}

echo $result;
```

Now that value is available to 'add-user.js' for use in the HTML.

To get it we need to add the response from the AJAX call as the argument for the `'success'` function:

```
...
'success' : function(response) {
}
...
```

and then assign this response to `$userID`:

```
...
'success' : function(response) {
    $userID = response;
}
...
```

And use this in the added HTML, adding it to the opening `<tr>` as `id= 'user_`, concatenated to the value of `$userID`:

```
$output = "<tr id='user_" + $userID + "' class='datarow'>";
```

And again adding an ID in the opening list item tag:

```
$output = "<li id='userlist_" + $userID + "'>";
```

We need to use a different prefix in the users list from that used in the table because the ID names all have to be unique, so I'll use `userlist_` for the list

And we need to do the same thing in 'show-users.fn.php' in `case`s 'all', 'current' and 'others' in the opening `<$tag>` element, so that the ID is also present on the existing users list items:

```
$output .= "\t\t\t\t<$tag id='userlist_$id'>";
```

Finally, in the `'success'` callback in 'delete-user.js', add `, li#userlist_' + $id` to the remove line, which adds the `#userlist` ID name and number on in a comma-separated list:

```
$('tr#user_' + $id + ", li#userlist_" + $id).remove();
```

Save and refresh, and now newly added and deleted records should appear and disappear from the users list simultaneously with the table update.

Lesson 122: Escape HTML output in jQuery

You will remember that we escaped HTML output in PHP using `HTMLentities()`, allowing us to display database entries with apostrophes and even HTML tags with no problem.

Go to the users admin page and add a new movie-goer with an apostrope in the name, like O'Reilly.

When the jQuery has run the name appears in the table truncated before the apostrophe, so all we see is 'O'. Press F5 to refresh the database and now it's OK because now it is running through PHP and `htmlentities()`.

Now click on one of the movie-goers to get the movie list.

If the movie *The Dot Matrix* is in the favourites list, drag it to remove it, or if it is in the main list, click on it to add it to the favourites and then do the reverse, to put it back where it was.

Now, look at the description – it has the same problem – truncation at the apostrophe.

This is because we have not done the equivalent of `htmlentities()` in jQuery to insert a backslash before the apostrophe to escape it.

This is an easy problem to deal with, using Javascript's `replace()` method.

To replace a string in a variable, append `.replace()` onto the end of the variable and then inside its parentheses put in quotes the text you want to replace, then a comma, and then in quotes what you want to replace it with.

```
$description = $description.replace("'", "&apos;");
```

Using double quotes as the string delimiter means we avoid having to escape the single quote, which is our target for replacement.

The replacement text, `&apos;`, is the HTML code for apostrophe.

Let's try this first with the output of the movie description – add this to 'add-favs.js' in the `'success'` callback just before we set up the output string so that the modified version is output to the browser:

```
'success': function() {
    $requestRunning = false;
    $description = $description.replace("'", "&apos;");
```

Save and refresh. Add and remove *The Dot Matrix* from favourites again.

There's still a problem – the description is no longer truncated at the first apostrophe, but at the second.

The `replace()` function as it stands only works once, replacing the first apostrophe it finds. We need to replace them wherever we find them.

To do this, we need to use a regular expression as the text to be replaced.

Regular expressions are a big subject, virtually a language of their own, but what we need is simple.

First, we tell the `replace()` function that we are using a regular expression by replacing the quote marks surrounding the text to be replaced with forward slashes:

```
$description = $description.replace(//, "&apos;");
```

Then the text to be replaced, which is a single quote mark:

```
$description = $description.replace(/'/, "&apos;");
```

There's no need to escape it now it is a regular expression.

Then after the closing forward slash the regular expression flag `g`, which stands for 'global':

```
$description = $description.replace(/'/g, "&apos;");
```

Try that again – and it's OK now.

We need to do the same everywhere HTML output is being generated by jQuery, so copy that and do the same for the title:

```
$title = $title.replace(/'/g, "&apos;");
$description = $description.replace(/'/g, "&apos;");
```

Copy and add the same two lines to 'add-fav-single.js' again just before `$output` is set up.

And for 'add-user.js', change the fieldnames to `$firstname` and `$lastname`.

Save and refresh and try both the add-remove favourites interface and the manage users interface and that problem should be solved.

In the next lesson we'll adapt what we've already done so as to add and delete movies.

Lesson 123: Add new movie interface

Building the interface to add new movies is a routine matter of copying the various scripts we have already written for the add-users interface and adapting them to work with the movies admin page.

The only significant change is to change the Javascript prompt to require a title as a minimum, but no description.

It would be good practice for you to do this yourself – pause the video now and make all the changes needed and check it.

If it works and you can add and delete movies, then go on to the next lesson.

The steps are:

In ‘head.inc.php’, add the link to ‘add-movie.js’:

```
if ($page=="movies") { ?>
    <script src = "scripts/delete-movie.js"></script>
    <script src = "scripts/add-movie.js"></script>
<?php } ?>
```

In ‘admin-movies.inc.php’, add the class ‘hidden’ to the ‘insert’ div (tabs omitted for clarity):

```
echo "<td class='insertcell'><div class='insert hidden'></div></td> ";
```

Copy ‘add-user.js’ and rename the copy ‘add-movie.js’.

In ‘add-movie.js’, change ‘firstname’ to ‘title’ and ‘lastname’ to ‘description’ throughout the file. Change ‘user’ to ‘movie’ throughout. Change the prompt to require a title as a minimum and change the prompt text:

```
if ($title=="") {
    alert("Please enter a movie title");
return;
}
```

Delete the lines adding entries to the users menu.

Copy ‘add-user.ajax.php’ and rename the copy ‘add-movie.ajax.php’.

Change the variable names as before.

Change the database field names and the table name.

And that should be that.

In the next lesson we’ll do the more involved job of adding the facility to alter existing records by typing directly in the user admin table.

Lesson 124: Update existing user – firstname

So far we can add and delete movies and movie-goers.

To complete the admin interface, we want to be able to update existing records by typing directly in the input boxes, so that when we mouse out the change takes place immediately, without clicking on any buttons, and the loader and success icons appear, to give us feedback that the update has taken place. The users list navigation should update simultaneously. This is a nice modern interface which you'll find on many websites.

Before we get started, open up phpMyAdmin and import the SQL file 'initialise-test-database.sql' from the Working Files for this lesson in the 'INITIALISE TEST DATABASE' folder.

Save and refresh and go to the users admin page and you will see that my entertaining names have been replaced with simple set of test data, consisting of 'Firstname 1 Lastname 1', 'Firstname 2 Lastname 2', *etc.* This is so that we can easily check that our records are OK and that the firstname and lastname are not getting jumbled or anything like that. Once we have completed the admin scripts, we'll reinitiase the database with the proper data.

We'll work first of all just on the 'firstname' field in the users admin to get it working and then do the lastname in the next lesson.

Make a new script file called 'update-user.js' and link to it in 'head.inc.php':

```
if ($page=="users") { ?>
    <script src = "scripts/delete-user.js"></script>
    <script src = "scripts/add-user.js"></script>
    <script src = "scripts/update-user.js"></script>
<?php }
```

To give you more practice at writing jQuery, I'll pause throughout this video for you to write the next section as I explain what it needs to do, before I give you the answer.

Write the `document ready` function and put in a comment telling us what this script does.

```
// Updates existing movie-goers

$(document).ready(function(){
});
```

Our event will be triggered by the user putting the focus, either from the keyboard or the mouse, onto one of the firstname input boxes.

Look at the HTML and work out the selector you need to target this, using 'focus' as the event.

The class name is `.data`, and the name of the input is 'firstname', so using the syntax we used earlier this is:

```
$(document).on('focus', '.data[name="firstname"]', function() {
})
```

Then set up `$this` and the ID number as before:

```
$(document).on('focus', '.data[name="firstname"]', function() {
    $this = $(this);
    $id = $(this).closest('tr').attr('id').split("_");
    $id = $id[1];
})
```

We want to check whether the user has made any changes to the value of the input box so we need to find its current value.

How do you get the value of the selected input box and assign this to a variable called `$firstname`?

```
...
    $id = $id[1];
    $firstname = $this.val();
```

The trigger for action to be taken will be when the focus goes off the selected input box – when the mouse is removed from it or the user tabs out of it. This is `focusout`:

```
$this.on('focusout',function() {
});
```

At that point the comparison should be made between what has been entered into the input and what was in it at the start. We can look again at the value of `$this`, the input box, and assign this to the variable `$newName`.

```
$this.on('focusout',function() {
    $newName = $this.val();
});
```

Then we check to see if oldName and newName are the same or not:

```
$this.on('focusout',function() {
    $newName = $this.val();
    if ($newName != $oldName) {
    }
});
```

and if they are not the same, the user has made a change and the script needs to take action, to update the database with the value of `$newName`.

If they are the same, the user has not altered the name, nothing needs to be done, and the rest of the script should not run.

So now we are in the not equal clause, so we know that a change has taken place, so we can run the AJAX call.

You write this, calling the script 'ajax/update-user.ajax.php', and posting the variables `$id` and `$new_Name` to it.

```
$.ajax({
    url: "ajax/update-user.ajax.php",
    type: "POST",
    data: {
          'id' : $id,
          'new_name': $newName
    }
})
```

Now in the ‘ajax’ folder, create a new file named ‘update-user.ajax.php’.

You know everything you need to write this file yourself, using the MySQL `UPDATE SET` query.

Everything is exactly as before until we get to the prepared statement:

```
require_once '../php-includes/connect.inc.php';
$id = $_POST['id'];
$newName = $_POST['new_name'];
$stmt = $db->prepare("UPDATE movie_goers SET firstname = ?
                      WHERE user_id = ?");
$stmt->bind_param('si', $newName, $id);
$stmt->execute();
$stmt->close();
```

Save and refresh, and enter a new value in one of the ‘firstname’ input boxes. Press F5. The new value remains, so `$firstname` has indeed been updated in the database.

But of course we have no visual feedback yet that anything has happened. Now we need the icons to give us that feedback.

Try to write the code for this. First, in the 'beforesend' callback, you need to remove the ‘hidden’ class from the div inside the selected ‘deletecell’, and replace it with the ‘loader_small’ class.

Our target with the ID number is the ID `user_` with the current ID number concatenated to it:

```
$('#user_' + $id)
```

And then we need a series of parent-child selectors – the class `.deletecell` and we want the div inside that:

```
$('#user_' + $id + ' .deletecell div')
```

From that we are going to remove the two classes ‘delete’ and ‘hidden’:

```
$('#user_' + $id + ' .deletecell div').removeClass('delete hidden');
```

and add the class ‘loader_small’:

```
$('#user_' + $id + ' .deletecell div').removeClass('delete
hidden').addClass('loader_small');
```

Then, in the `'success'` callback, replace the ‘loader_small’ class, wherever it appears, with the ‘success’ class:

```
'success': function() {
    $('.loader_small').removeClass('loader_small').addClass('success');
}
```

Save again and refresh. Try entering new firstnames in the input boxes.

That’s OK but if we update another record we get another success icon. Write the code to prevent that, so that a new database update removes any exsiting success icons.

In the `'beforesend'` function, when a new database update begins, this removes the success class wherever it exists:

```
'beforeSend': function() {
    $('.success').removeClass('success').addClass('delete hidden');
}
```

Save and refresh and insert a couple more records without refreshing the page - the multiple success icons will not appear.

But the success icon doesn't disapear when you mouseover it – it should be replaced by 'delete'.

Can you solve that?

Working inside the `'success'` callback, the event handler we want is document – on – mouseover, working on the 'deletecell' class:

```
$(document).on('mouseover', '.deletecell', function() {
});
```

We want to act on the 'success' class, but only when it refers to this particular selection. We can get the using `$('.success', this)`:

```
$(document).on('mouseover', '.deletecell', function() {
    $('.success', this)
});
```

and then we want to add the class 'delete' and remove the classes 'hidden' and 'success':

```
$(document).on('mouseover', '.deletecell', function() {
    $('.success', this).addClass('delete').removeClass('hidden success');
});
```

Save and refresh, and move the mouse over the 'success' icon, and that problem is solved.

We also want the user list in the navigation to update instantly with the altered record, and at the moment we haven't done the code for that.

We can copy the list item we need from 'add-user.js'.

Looking at it, it looks as though I missed off half the URL – it should of course be `index.php?user_id =` and then `$userID`. Make that change in 'add-user.js' as well, and I think move the link text to a separate line – it's neater:

add-user.js:

```
$output = "<li id='userlist_" + $userID + "'>";
$output += "<a href='index.php?user_id=" + $userID + "'>";
$output += $firstname + " " + $lastname;
$output += "</a></li>";
```

Select and copy all of that including `append()` and `tsort()` and paste it into 'update-user.js' between the end of the AJAX call and the end of the `if` condition:

update-user.js:

```
...
}); // End AJAX

$output = "<li id='userlist_" + $userID + "'>";
$output += "<a href='index.php?user_id=" + $userID + "'>";
$output += $firstname + " " + $lastname;
$output += "</a></li>";
```

```
$('li#userlist_' + $id).remove();
$('.users_menu').append($output);
$('.users_menu li').tsort();

} // End if names-not-same condition
...
```

In 'update-user.js', change `$firstname` to `$newName`: and change `$userID` to `$id` in two places:

```
$output = "<li id='userlist_" + $id + "'>";
$output += "<a href='index.php?user_id=" + $id + "'>";
$output += $newName + " " + $lastname;
$output += "</a></li>";
```

Save and refresh and the new list item should appear ... and it doesn't.

Let's look in Firebug to see what's wrong: `$lastname` is not defined, so we need to set that.

So up where we set `$firstname`, we can set `$lastname` in much the same way. Our identification comes from the table row with the ID user_ and the ID number we have set, inside which is the input box named 'lastname':

update-user.js:

```
$firstname = $this.val();
$lastname = $('tr#user_' + $id + ' .data[name="lastname"]').val();
```

We need to define this – which we can do in the same way as we did for firstname, by getting the value of the relevant input.

Just to check that there is no problem, I'll put in a console log on `$lastname`:

update-user.js:

```
$firstname = $this.val();
$lastname = $('tr#user_' + $id + ' .data[name="lastname"]').val();
console.log($lastname);
```

Save and refresh and now that works, but it adds duplicate items.

To prevent this we need to remove the old corresponding list item just before we insert the new one, and then it will look as though it has just been altered:

```
$('li#userlist_' + $id).remove();
$('.users_menu').append($output);
```

Save and refresh. Make some changes and keep your eye on the navigation list items, and they are changing simultaneously just as we want.

That's all done and working fine for the firstname. Remove the console log line.

Now we need to do the same for the lastname.

One way to do the lastname would be to copy the whole lot and call the new scripts 'update-firstname.js', 'update-lastname.js', and the same for the PHP files. But we would be getting an awful lot of files and duplicated code by the time we've finished, so we'll avoid that.

A much better way would be to adapt this script and the PHP script it calls so they know which input box is being used and use more generic variable names, something like `$thisName` for the selected name and `$otherName` for the other one.

This is your final assignment – there are instuctions in the PDF file which follows, to make this single script and the single AJAX file which it calls work for both the firstname and the lastname.

Instructions follow.

Assignment 12: Full update user interface

Write the code which 'update-user.js' needs in order to identify which input box, firstname or lastname, has been given the focus.

Then, using more generic variable names than in the previous lesson, adapt the script so that it can pass the fieldname to be updated to the PHP file in the AJAX call.

You will have to make sure also that the updated user navigation list always displays the names in the right order.

Good luck!

Lesson 126: Full update user interface

To make a single script able to update both the firstname and the lastname, it has to react to either input and identify which input box has the focus.

This was the toughest assignment so far, so I'll go through the answer in some detail.

Open 'update-user.js' in Komodo Edit.

Remove [name="firstname"] from the document on focus selector so that focus on either input box will trigger the script:

update-user.js:

```
$(document).on('focus', '.data', function() {
});
```

We can identify which input box was clicked on by getting its 'name' attribute:

```
$this.attr('name');
```

This will be the affected fieldname in the database, so we'll assign this to the variable `$thisField`:

```
$thisField = $this.attr('name');
```

Put in a console log to check that it is working OK:

```
console.log($thisField);
```

Save and refresh and click on both sides of the table. We can see in Firebug that it is getting the fieldname whichever side we click on. Now delete the console log line.

Now we need to set the other fieldname in the database. We can do this using two `if` clauses

```
if ($thisField=="firstname") {
    $otherField = "lastname";
}

if ($thisField=="lastname") {
    $otherField = "firstname";
}
```

Change `$firstname` to `$thisName`:

```
$thisName = $this.val();
```

Below that, change `$lastname` to `$otherName` and get its value, not from the input named 'lastname', but the input now named `$otherField`:

```
$otherName = $('tr#user_' + $id + ' .data[name="' + $otherField +
'"]').val();
```

In the data section of the AJAX call, add `this_field: $thisField`:

```
data: {
    'this_field' : $thisField,
    'id' : $id,
    'new_name' : $newName
}, // End data
```

In the PHP file ‘update-user.ajax.php’, add `$thisField` as another posted variable:

```
$thisField = $_POST['this_field'];
```

And in the prepared statement we need to replace the hardcoded `firstname` with the variable `$thisField`:

```
$stmt = $db->prepare("UPDATE movie_goers
                      SET $thisField = ? WHERE user_id = ?");
```

Back in ‘update-user.js’, where we set the navigation list output, we need to reverse the order of the fieldnames according to which was selected, so if `$thisField` is ‘firstname’, we want `$firstname` to appear first in the list:

```
if ($thisField=="firstname") {
    $output += $newName + " " + $otherName + "</a>";
}
```

but if `$thisField` is ‘lastname’, we want it to appear last in the list:

```
if ($thisField=="lastname") {
    $output += $otherName + " " + $newName + "</a>";
}
```

Save and refresh and try making some changes ... nothing is happening. Firebug tells us that `$firstname` is not defined, so we’ve missed changing `$firstname` somewhere.

In the focusout section where we make the condition surrounding the AJAX call, it should be if `($newName!=$thisName)`, not `($newName!=$firstname)`:

```
$this.on('focusout', function(){
    $newName = $this.val();
    if ($newName!=$thisName) {
    ...
```

Refresh it in the browser and check it by putting an ‘a’ or something similar after the firstname and then the lastname and double-check to make sure that the records are updating properly.

Try deleting a record, inserting a new one, and editing a new record before we’ve refreshed the page. Double-check that everything works with no problem.

Now the one script does both updates. It will be apparent that we could continue further this process of making our scripts more generic. Instead of copying ‘update-user.js’ and renaming it ‘update-movie.js’ for the movie version, we could use the script to find out whether it is working on the user page or the movie page and act accordingly.

Then instead of two update scripts and two PHP files we could have just one, called ‘update.php’.

We could do the same for the scripts we have already written, replacing ‘delete-movie.js’ and ‘delete-user.js’ with just one script, ‘delete.js’, and a single ‘add.js’ for the addition of records.

How far we want to go in this process is a matter of choice. There is a trade-off between efficient coding and reduction of duplication, which is very desirable, and readability.

The more efficient the code, the more abstract it becomes and the more generic the variables and processes in it.

I’m going to stop the process here, copying and renaming ‘update-user.js’ and its associated PHP file and adapting the copies to work with the movies, because I think that’s a good compromise between readability and efficiency.

After you have finished the course, you can if you want continue the process we began in this lesson, making the scripts as terse and efficient as possible.

For our purposes, I think this is enough, so in the next lesson I’ll go through the changes made to these scripts to do the movie version before winding up this chapter on jQuery by catching browsers which do not have Javascript enabled.

Lesson 127: Update movie interface

Adapting the update user interface to work with the movies is extremely simple, as we we already have a reasonable level of abstraction in our variable names.

Copy ‘update-user.js’, rename the copy ‘update-movie.js’, and open it up in Komodo Edit.

Add the link to it in ‘head.inc.php’:

```
if ($page=="movies") { ?>
    <script src = "scripts/delete-movie.js"></script>
    <script src = "scripts/add-movie.js"></script>
    <script src = "scripts/update-movie.js"></script>
<?php } ?>
```

In ‘update-movie.js’, change the comment at the top:

```
// Updates existing movies
```

In ‘update-movie.js’, you can use Edit > Replace > Replace all to replace every occurrence of ‘firstname’ with ‘title’, ‘lastname’ with ‘description’, and ‘user’ with ‘movie’. Leave off underscores and dollar signs to make sure we get them all.

Then copy ‘update-user.ajax.php’ and rename the copy ‘update-movie.ajax.php’.

In ‘update-movie.ajax.php’ replace ‘movie_goers’ with ‘movies’ and ‘user_id’ with ‘movie_id’.

Save and refresh and go to the movie admin page and check it, making additions, deletions, and alterations.

Finally re-initialise the database.

All that remains to do before putting the project online is to deal with browsers which don’t have Javascript enabled.

In the next lesson we display an alert message which tells users without Javascript that as the project is a demonstration of jQuery and AJAX, they must have Javascript enabled in order to use the site.

Lesson 128: Catch browsers with Javascript disabled

Our final job before putting the project online is to make sure the website behaves nicely if a browser has Javascript disabled.

I am not going to attempt to make a non-Javascript version – the project is a demonstration of jQuery and AJAX in action, so this is the basis of its whole interface. Once, say about 10 years ago, I would have felt guilty about this, and I'd have thought I should make an alternative version. Now in 2013, if you have Javascript disabled you must be so paranoid you shouldn't be on the web at all - half the world's websites won't work for you.

Let me demonstrate by disabling Javascript in Firefox. To do this, enter `about:config` in the address bar and press Enter. This is where we can change Firefox's configuration settings. You'll probably get a silly message about your warranty being invalidated. Accept that, and if you want, check the checkbox so it doesn't appear again.

In the search box at the top of the page enter `javascript.enabled`. Click on it to disable Javascript.

Now go to Youtube and try to log into your account – you can't – and you're not even told why.

Go to trivago.com and try to book a hotel – you can't – but at least, unlike Youtube, you do get a message telling you what's wrong. This is the approach I'm going to take, and I feel I'm in good company if I no longer support non-Javascript browsers.

So I'm going to do just what Trivago do, and put a message telling people they need Javascript enabled to use the site.

We can use the HTML tags `<noscript></noscript>` to do this – what's inside them will only display if Javascript is turned off.

Open up header.inc.php, not head.inc.php. After the closing `</header>` tag, type opening and closing `<noscript></noscript>` tags.

And inside them put a div with the classname 'noscript':

```
<noscript>
    <div class="noscript"></div>
</noscript>
```

Inside that, put `<h1></h1>` tags enclosing the message "Javascript is not enabled in your browser" and below that paragraph tags enclosing "This website is a demonstration of jQuery and AJAX and requires Javascript in order to function":

```
...
    <div class="noscript">
        <h1> Javascript is not enabled in your browser</h1>
        <p> This website is a demonstration of jQuery and AJAX and
         requires Javascript in order to function</p>
    </div>
...
```

Now we'll style this div in 'style.css', giving it a margin of 20px on the top, auto on the right, 20px on the bottom, and auto on the left:

```
.noscript {
   margin: 20px auto 20px auto;
}
```

This will give some space above and below it and the auto margins left and right will centre it in the window.

Give it padding of 20px to give some space around the text in it, and a width of 50%:

```
    padding: 20px;
    width: 50%;
```

'Overflow-y: auto' means that it will expand vertically so that the text does not spill out of it:

```
    overflow-y: auto;
```

Align the text centrally:

```
    text-align: center;
```

Give it a yellow background colour:

```
    background-color: #FFEFBE;
}
```

Give the `<h1>` inside the `noscript` div the font size of 1.3em and a bottom margin of 1em, one line height:

```
.noscript h1 {
    font-size: 1.3em;
    margin-bottom: 1em;
}
```

Give the paragraph tag in it a font size of 0.9em:

```
.noscript p {
    font-size: 0.9em;
}
```

Give the 'content' class the `display` property 'inline':

```
.content {
    display: inline;
}
```

Make sure Javascript is disabled and refresh the page. The 'noscript' div appears. Enable Javascript and reload and it disappears, so 'noscript' is working alright.

But the content of the website is also displayed, even though nothing will work. To hide this when Javascript is disabled, put an opening `<div class="content">` as the last line of 'header.inc.php' and delete the closing `</div>` tag.

Open up 'footer.inc.php' and, at the top of the file, close the div.

This div now encloses all the HTML except the header and footer, so we can control the visibility of the main part of the page.

Inside the `<noscript></noscript>` tags, add a `<style></style>` section and to that add the display setting 'none' to the 'content' class:

```
<noscript>
```

```
    <style>
        .content {
            display: none;
        }
    </style>
...
```

Now when Javascript is disabled the 'content' div will be hidden. When Javascript is enabled, what's inside the noscript tags will not run and the 'content' div will be displayed as normal. For a slightly more emphatic-looking message box, we can add a box shadow to 'noscript' in 'style.css'. Unfortunately, we still need vendor prefixes for 'box-shadow'. I'll do the webkit version first:

style.css:

```
.noscript {
    ...
    -webkit-box-shadow:
}
```

We style this with a series of parameters, first the horizontal offset, which I'll set at 7px:

```
    .noscript {
    ...
    -webkit-box-shadow: 7px
}
```

Then the vertical offset. I'll set this at 7px as well:

```
    -webkit-box-shadow: 7px 7px
```

Then an optional blur setting. I'll set this at 5px:

```
    -webkit-box-shadow: 7px 7px 5px
```

Lastly I'll specify the colour using RGBA settings so that I can specify opacity as well:

```
    -webkit-box-shadow: 7px 7px 5px rgba(50, 50, 50, 0.75);
```

I have set all three numbers for the colour the same, 50, which will give a grey colour, and 0.75 opacity.

Then we need the vendor prefixes so I'll copy that and paste it back. We need Mozilla and lastly we take out the vendor prefix so that we've got plain 'box-shadow':

```
    -webkit-box-shadow: 7px 7px 5px rgba(50, 50, 50, 0.75);
    -moz-box-shadow: 7px 7px 5px rgba(50, 50, 50, 0.75);
    box-shadow: 7px 7px 5px rgba(50, 50, 50, 0.75);
```

Disable Javascript again, refresh the page, and that stands out better.

If you want, you can make the header and footer fade into the background by setting an opacity level in the 'noscript' style section for both the header and footer, perhaps of 0.75:

```
header, footer {
    opacity: 0.75;
}
```

Now we're ready to prepare the project to go online. Finish by re-enabling Javascript in your browser.

Chapter 12: Online version and security considerations

Lesson 129: Security considerations - directory browsing

Until now, everything we have done has been in the safe environment of our own localhost Apache server, where we haven't had to think of security risks. Before we put the project online, we need to take some steps against users, maliciously or otherwise, accessing the site in ways they shouldn't.

Let's just look at what could happen if we put the site online as it is.

Someone might guess we had a directory named 'php-includes', as that's a fairly common name, and enter that in the URL to see what was in there.

This would show the names of the files, which tells them a lot about the way the site is structured. Luckily, if they click on them they won't get very far – they will mostly display a fatal PHP error. Still, it gives away information about the site and about function names and things like that, so we should prevent access to the directory.

More serious is if someone got into the 'ajax' directory, again not a difficult name to guess. This would immediately give them the information about our key files, the ones that perform changes to the database, and the names of our most important variables.

Fortunately, they cannot simple enter a variable name in the URL and manipulate the data because we used POST, not GET as the method to transfer the data into the script.

If we had used GET instead of POST, that would have been a huge mistake, because potentially users could enter variable names and values straight into the URL and change the data without going through the website at all.

We want to avoid any security issues of this sort, so in the next lessons we will take steps to do this before uploading the site to a web host and putting it online.

Lesson 130: Prevent directory browsing with .htaccess

Once we have our project online, we'll be able to put a file named '.htaccess' in any of our directories to prevent anyone browsing directly to them and getting a listing of the files they contain.

We can try this out first on our localhost system.

Windows doesn't like filenames which begin with a dot, so we have to use something else for Windows and then rename it to '.htaccess' once it's online. We can call it 'htaccess.txt'.

Make this new text file in 'php-includes' and name it 'htaccess.txt'.

Open it up and type in:

```
Order allow, deny
Deny from all
```

And save it.

Now click on your XAMPP control panel and click on 'Stop' next to 'Apache'. This shuts down the Apache server. We need to do this when we make changes to Apache configuration files.

In Windows Explorer go to 'c:\xampp\apache\conf'. Copy 'httpd.conf' and save it as a backup.

Open up the original, and at the very end, add:

```
AccessFileName htaccess.txt
```

Save that and close it. Restart Apache.

That's set the access file name to 'htaccess.txt' on our local system. Now if you try to browse to http://localhost/favouritemovies/php-includes/ you'll find you no longer can - that's the first security hole blocked.

We can use that method for the directories which contain PHP include files, we can use it for the 'functions' directory as well. So copy 'htaccess.txt' into the 'functions' directory and that will prevent anyone seeing them too.

Htaccess allows PHP include files to be used but blocks any sort of direct access. So if we used it in our images directory, we'd get no images, in our styles directory all the styling would be lost, and in our AJAX directory all the site functionality would be lost.

In the next lesson we'll look at how to prevent browsing in these other directories where we cannot use htaccess.

Lesson 131: PHP redirect out of directories

We cannot prevent access to any of our other directories as this would prevent the site displaying or working properly, but we can stop users getting a listing of the files they contain, using a PHP redirect.

Let's do this in the 'ajax' directory first.

In the this directory, make a new text file and name it 'index.php'. Open this up and enter:

```
<?php Header("Location: ../index.php"); ?>
```

Make sure there are no spaces or new lines anywhere.

This redirects the browser to the relative path for 'index.php' in the directory above the one we are in, that is, the site hopepage.

Now try to browse to 'http://localhost/favouritemovies/ajax/ directory and you will be redirected back to the homepage.

This is a simple way of preventing users getting information about the components of our website.

You can use this on all the other directories, so copy this 'index.php' and paste it into all the other subfolders.

In the next leson we will look at a more sophisticated way of ensuring that the AJAX files can be called through AJAX only and not directly in the browser.

Lesson 132: Accessing AJAX files through AJAX call only

We're almost there but although we've prevented users seeing a listing of the names of the AJAX files, we haven't prevented them from loading them in the URL if they can guess the names of the files.

We can't use '.htaccess' to protect the AJAX files because we need access to them, but this should only be allowed through an AJAX call, not through the browser address bar.

We can detect an AJAX call using the HTTP server variable `HTTP_X_REQUESTED_WITH`.

The value of this will be set to `'xmlhttprequest'` if the request comes via AJAX.

So with 'delete-user.ajax.php' loaded in your browser and displaying information we'd rather be kept private, add at the top of the file:

```
if ($_SERVER['HTTP_X_REQUESTED_WITH'] == 'XMLHttpRequest') {
    // Enclose the whole file here
}
```

enclosing the whole of the rest of the code in its braces.

Save and refresh.

Now we get an error message because `HTTP_X_REQUESTED_WITH` is not set at all.

We need to go one step further and check that it is actually set and that its value is 'XMLHttpRequest', so add in an `if isset()` clause as well:

```
if (isset($_SERVER['HTTP_X_REQUESTED_WITH']) &&
   ($_SERVER['HTTP_X_REQUESTED_WITH'] == 'XMLHttpRequest'))
```

Save and refresh. Try to access the file directly again – and now you get no information at all, just as we want.

Do the same with all the AJAX files and the project is now ready to go online.

Lesson 133: Uploading the project to a webhost

Now as a final lesson I will demonstrate the processes need to upload the project to your webhost and set up its database online. You can follow exactly the same steps with any other dynamic website you develop. For this stage, you need to have a web hosting which provides a Control Panel, has PHP installed, and allows you to create MySQL databases.

It's easiest to upload the project as a single zip file rather than uploading the files separately, so in 'htdocs', right click on the 'favouritemovies' folder name and zip it, using either Send to ... Compressed folders function in Windows or your own zip application.

Log into your web host's control panel with the username and password provided by your hosting company.

This is Cpanel, which is a very common interface but yours may look slightly different according to your web host. If you have problems because the panel is different, you should contact your web host's technical support and ask them for help.

Click on the file manager.

Click to get to your web root directory, which is either 'public_html' or 'www'.

Then if you have a directory name for your website, click to get into that.

Click on upload and upload the zip file. Wait a moment or two while the file uploads.

Select the zip file and click on 'Extract' to extract it.

Now you have the files online but you have to set up the MySQL database.

Go back to Cpanel home and click on 'MySQL databases'.

Under 'Create new database', enter a name for the database. A prefix will normally be automatically added to it by the system.

Click on 'Create database'.

We get a confirmation – click on 'Go back'.

Lower down, in 'MySQL users - add new user', enter a user name.

Then you need to enter a password. You can enter your own or click to generate one.

Copy the database name, user name and password into a text file so you have a record of them.

Finally in 'Add user to database', select the user and the database you just made and click to add.

On the next screen, select 'All privileges' and click 'Make changes'. We get a summary and again click on 'Go back' to go back to Cpanel home.

Now click on 'phpMyAdmin' and you should see the name of the database you've just made on the left.

We have to change the initialise database SQL file a bit because we will not be allowed to delete the database and reinitialise it so easily online – we have to get inside the database and then make the

tables and insert the data.

Open up the 'initialise database.sql' file in Komodo Edit and delete or comment out, using `/*` in SQL, the three lines:

```
/* DROP DATABASE IF EXISTS movies;
CREATE DATABASE IF NOT EXISTS movies;
USE movies; */
```

The first line should be `CREATE TABLE movies`.

In phpMyAdmin, click on the name of the database to get inside it... and click on 'Import', then 'Browse', and 'Go'. You should get a confirmation screen saying that the data has been successfully imported.

Now we need to edit the connection details in 'connect.inc.php' to match the new database name, username, and password. Click on File Manager. Click to get into 'php-includes' and then select 'connect.inc.php' and choose the code editor.

Very carefully, change the username, password and name of the database to the ones you have just set up. Double-check, making sure there are no spaces or any other mistakes, and click 'Save changes'.

It's a good idea also to save this online version of 'connect.inc.php' on your local system so you have the online access details to hand. Copy all the code and save it in 'php-includes' on your hard disk as 'connect.inc.ONLINE-VERSION.php'.

Click in the File Manager to get into the 'php-includes' directory and rename 'access.txt' to '.htaccess'. Do the same in the 'functions' directory.

Load up the website in your browser. If you get an error, double-check your password, database name and the user name.

Now deliberately generate an error by changing one of the conection parameters, the database name, username or password, so it can't connect.

Save and refresh. The error message gives information about our system, which we don't want. In 'connect.inc.php', add `error_reporting(0)` at the top:

```
error_reporting(0); // Disable error reporting for production version
                    // Remove this line for development
```

In the `if` condition which catches the conection error, remove the `connect_error` line and substitute `"<h1>Something went wrong!</h1>"` and `exit`:

```
if ($db->connect_errno>0) {
    echo "<h1>Something went wrong!</h1>";
    exit;
}
```

Save and refresh and now no meaningful information is displayed. Go back and reinstate the correct connection parameters.

Save and refresh and try the project and make sure everything works. We've got just one more job to do as far as security is concerned, and we'll do that in the next lesson.

Lesson 134: Moving database connection file out of web directory

Now the project is online there is one final step we should take to hide a particularly sensitive file from prying eyes. This is 'connect.inc.php', and it is highly sensitive because it contains the login credentials for the database, potentially giving anyone who manages to see its contents complete control over it.

Because we have used .php as the file extension, it shouldn't be possible for anyone to see its text contents even if they did get direct access to it through a browser – the PHP should be interpreted before it ever gets to the browser.

This is why we should always use a .php extension for PHP files, not something else like .inc, because if the extension is not .php, the PHP code will not run if the file is accessed directly and the browser will display the raw text contents.

But for this particular file, and for any file containing database connection credentials, we should go one stage further and make it impossible for anyone to access it directly, by moving it out of the web accessible part of the system.

In the file manager in Cpanel, select 'connect.inc.php' and click 'Move'. Move it to yur home directory, /. Go to your home directory and you should find 'connect.inc.php' there, alongside 'public_html'.

Now it is no longer accessible over the web.

Now we have to change 'index.php' to point to its new location. Using the code editor, change the relative path so that it goes up one directory – in my case I have to move up three directory levels to get to my home directory, first out of 'favouritemovies' with `../` , then out of 'webinaction', with another `../`, and then out of 'public_html' with a third `../`:

```
require_once '../../../connect.inc.php';
```

Save and refresh and try it.

It appears to work but when we refresh the page we find that the changes have not stuck – the database has not been updated.

Before I give you the answer, what has gone wrong? What haven't we done?

If you look in Firebug you will see what's wrong – we haven't updated the AJAX files to point to the new location of 'connect.inc.php'.

Work out the relative path you need from the AJAX directory to the home directory and make the change in all the AJAX files.

Save and refresh and try the project and make sure everything works.

Now all the security considerations are dealt with. Now go through clicking and checking all the various functions and making sure that everything works.

I should say at this point that the project is not really designed for production in this way. It's really meant as a teaching demo for you to run locally, but I'm putting it online just so everyone can see what we are building.

The problem is that as it stands, multiple users over the web could be altering the various movie-goers' favourites at the same time, or even deleting or changing movie-goers names or movies.

In a real-world application we'd make sure this couldn't happen – we would have a login system so you would log in as one of the movie-goers and then you could only alter that particular movie-goers' favourites. Changing names and movie titles would allowed only for an administrator.

For the purposes of these lessons, I didn't go into that because it would have made the process of clicking to add and remove records cumbersome.

So the result of this is that if several people are all accessing the project at the same time and someone else deletes a movie-goer you are looking at, you may suddenly get the invalid ID message, so don't be surprised if this happens.

You can access the project online at www.webinaction.co.uk/favouritemovies. Any changes you make to the data won't matter as it is set to reinitialise the database every hour.

Lesson 135: Course conclusion and what comes next

That's the end of the course. I hope you enjoyed it and learned a lot from it and that you'll go on to build fast-updating, AJAX-powered websites of your own.

We've covered a great deal of material, going all the way from basic HTML structure, through many aspects of CSS styling, using PHP to output HTML, variables, and records from a database, SQL embedded in PHP to communicate with and manipulate the data in the database, jQuery to manipulate the HTML on the fly so that it matches the latest changes in the database, and finally AJAX to update the database records in the background. These are all the fundamentals of AJAX web development which underpin this type of website.

Just as importantly, I hope that through the lessons and assignments you've got a good grasp of thinking through the logical processes, thinking as a programmer, solving problems for yourself rather than my giving you the answers.

So where should you go from here in your professional development?

Well, the code we have written here is 'procedural' or 'inline' PHP, meaning that we set out from bare HTML and gradually convert this to dynamic PHP, working out each step as we go along.

Because we've taken care and used include files with clear naming conventions, the result is a fairly tidy set of files, which we shouldn't have too much trouble maintaining and changing if necessary if we wanted to alter the website later.

But you can see already that when we make changes we usually have to perform edits in several files and we do have to know which bit of code does what.

With a small project like this, that's no problem, and it's the way it would normally be written because anything more would be overkill.

But you can imagine that if we were to develop a large project, say a full-scale webiste, like trivago.com, or a blogging platform with an admin panel and all the bells and whistles that go with it, instead of our project files numbering in the tens, they would number in the high hundreds or even thousands. If we went on in this style of coding would involve variables here and functions there, and bits of HTML coming in from all directions. If we needed to add a new section or make changes, we could end up having to perform edits in a huge number of files, which could become quite unmanageable.

The next stage, when we go on to a project of that size, is Object Oriented Programming (OOP).

OOP is conceptually more difficult at the outset than procedural coding, because it deals with much more abstract concepts than strings of HTML to send to the browser, which is why I didn't tackle it here. At the end of the project I did begin the process of making our code more abstract, and in the process more multi-purpose and flexible.

OOP takes this abstractness to a higher level. It is much harder to get your head around in the early stages but, once the initially steep learning curve is overcome, the code can become much more elegant and simpler.

If you have finished this course, you will be ready to go on to this next level, and this is my project for another course, tentatively entitled *Step into Object Oriented PHP*. I hope very much to see you on it in a few months time!