In this lecture, we learn how to connect our PHP programs to a SQLite database. In addition, we learn about PHP session support – this allows PHP programs on different webpages to share data without accessing a database. We also discuss cookies, which are a method for storing information associated with a particular website on the user’s own computer.

The bulk of the material covered in this lecture is discussed in the “h11 PHP and SQLite” handout.

There are two sets of example files for this lecture. The first set are directly covered to the PHP and SQLite handout and are contained in the “h11 Examples.zip” file. The second set are discussed in this handout and are available in the “L16 Examples.zip” file.

**Message Board Example**
- This example provides a larger more realistic scenario using the techniques covered in the “h11 PHP and SQLite handout”.
- We implement a simple message board, where different users can leave messages that other users can read.

**Voting Example**
- This is a more complex example showing how to create a website where users can vote for different options.
- In addition to the techniques already shown in the h11 handout, it also shows how to:
  - Check to see if something exists in the database.
  - Perform an update of data in the database.
- You won’t need to use the fetch technique, show in this example, for your HW assignment, but if you do decide to explore PHP further, a couple of important warnings on its use:
  - **Warning #1**: I use the fetch method to determine if a vote already exists in the database. Fetch can be used to determine if a value is in the database, however, it cannot be used in conjunction with the traditional foreach method. There’s a different, alternative loop you can perform using just fetch, that I won’t have time to cover in this class.

  You’ll note my code does a second SELECT command after the first SELECT. This reset the results allowing me to use foreach (it also means that the newly counted vote is included in the results).

  o **Warning #2**: Some websites advocate using the rowCount method to determine if any rows have been returned. This is bad advice. According to the official documentation rowCount returns the number of rows affected by the SQL statement executed. This means if I call an UPDATE or DELETE SQL command it will return the number of rows updated or deleted. On some, but not all, PHP installations, it may also give a count of
the rows returned via a SELECT command. Stanford’s installation does not return the number of rows from a SELECT command.

- In addition to the regular files needed for the actual voting, I’ve included several extra files:
  - “results-debug.php” shows use of a variety of techniques to get information on what’s going on in the PHP program. Swap this with “results.php” to use.
  - “list-db.php” just goes through the database and lists the number of votes for each candidate.
  - “votes.txt” has the SQL commands used to create the “votes.db” SQLite database file.

Sessions
- When a user visits a series of webpages on our website, how do we get data from one webpage to the next?
  - Variables created in a PHP program on one webpage are not visible in PHP programs on other webpages. We need to find another mechanism to share data.
  - One approach is to store data into a database on each page, then retrieve the data from the database when the user goes to the next webpage.
    - This does work, but it’s a rather heavyweight, cumbersome process. In general, storing data into the database is only recommended if you want to remember the data long term.
  - Sessions are designed to provide a lighter-weight mechanism to store short-term information.
    - Sessions will only retain data while the user is visiting the website.
    - If the user exits the web browser, restarts, and returns to the website, the information will be lost.
  - If you want to use Sessions on a webpage, add the following to the very top of a PHP webpage, this should be inserted even before the <!DOCTYPE ...> tag:

```php
<?php
    session_start();
?>
```

You’ll need to do this at the top of every webpage that accesses session information.

  - Once you’ve setup your session, you can now access a special variable $_SESSION. This variable stores an Associative Array, which allows you to store and access data as key-value pairs. So if I store something like the user’s name on one page:

```php
$_SESSION["name"] = "Jane";
```

I can then retrieve that value on any other webpage on my website that also uses sessions (by using the session_start call at the top of the file):

```php
$name = $_SESSION["name"];```

Cookies
- Cookies provide a persistent data store on the client-side.
  - They differ from a database in several respects.
    - The data is stored on the client’s machine and is specific to a given web browser
    - As a result, if the user visits the website from a different machine or a different web browser, their previous cookie data will not be available.
They also provide much more limited storage. The original specification called for:

- 20 cookies (i.e., key-value pairs) per domain
- 300 cookies total in web browser storage for all websites.
- a maximum of 4kbytes per cookie

- in fact, all web浏览器 do provide more storage than this, but this give you a sense of what these cookies were intended for.

When a user visits a website, cookie data stored in their web browser, from previous visits to that website, is automatically sent to the web server.

- Note this does highlight the need to keep cookie data small.
- If you store a large cookie, every time the user visits a webpage on your website, all that data in the cookie will be sent to the server.

Cookies can be used for a variety of purposes.

- They can store user names or user ids. Since these are being stored in the web browser, when a user visits a website, their name or other identifier can be immediately sent to the webserver, along with the initial HTTP request, allowing the web server to immediately identify who is visiting the website.
- They can store user preferences, such as font size preferences or color preferences.

In PHP a cookie has its own function: setcookie

- This takes a cookie name and a cookie value
- There are a variety of other optional parameters
  - We can pass in an expiration date telling the browser how long to keep the cookie in storage (without this the cookie only lasts while the web browser is running).
  - We can pass in domain information on which webpages should be able to view the cookie data.
    - For example, if a webpage at www.stanford.edu sets a cookie, should math.stanford.edu or cs.stanford.edu websites be able to access that cookie.
  - We can specify that the cookie will only be sent to the server if the user has a secure connection using the HTTPS protocol instead of the regular HTTP protocol.

Here’s an example of a setcookie call:

```php
<?php
    setcookie("username","Jane",time() + 1800);
?>
```

This sets a cookie with the key-value pair “username” and “Jane”. The third parameter tells us when the cookie will expire. time() returns the current time in milliseconds\(^1\) and 1800 means that we add 1800 seconds or 30 minutes to the current time. This cookie will expire in 30 minutes.

To delete a cookie, call setcookie, but set an expiration date in the past – for example `time() - 1800`.

\(^1\) time() actually returns the number of seconds since midnight Greenwich Mean Time the 1st of January 1970. This is commonly how dates are stored internally in Unix systems.
Retrieving a cookie works exactly the same as retrieving Session data. There is a special variable setup called $_COOKIE. This acts as an associative array, allowing us to retrieve cookie key-value pairs.

- Here I'm retrieving the user's name and putting it on the webpage:

```php
<?php
    $name = $_COOKIE['username'];

    if ($name) {
        echo "<p>Welcome $name</p>";
    } else {
        echo "<p>Welcome Friend</p>";
    }
?>
```