Personalized WEB applications with PHP/databases

SQLite auto-increment field

- In SQLite, every row of every table has an 64-bit signed integer ROWID. The ROWID for each row is unique among all rows in the same table.
- You can access the ROWID of an SQLite table using one the special column names ROWID, _ROWID_, or OID. Except if you declare an ordinary table column to use one of those special names, then the use of that name will refer to the declared column not to the internal ROWID.
- If a table contains a column of type INTEGER PRIMARY KEY, then that column becomes an alias for the ROWID. You can then access the ROWID using any of four different names, the original three names described above or the name given to the INTEGER PRIMARY KEY column. All these names are aliases for one another and work equally well in any context.

Web services

web service: software functionality that can be invoked through the internet using common protocols

- like a remote function(s) you can call by contacting a program on a web server
- many web services accept parameters and produce results
- service's output can be text, HTML, XML, JSON or other content types

RESTful web services

- **Re**presentational **S**tate **T**ransfer :

  presented with a network of Web pages (a virtual state-machine), the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for his use.
- **REST** is initially described in the context of HTTP.
- RESTful applications maximize the use of the existing, well-defined HTTP interface, its built-in capabilities, and minimize the addition of new application-specific features on top of it.

Central principles of REST

- The existence of *resources* (sources of specific information), each of which is referenced with a global identifier (e.g., a URI in HTTP).
- In order to manipulate these resources, *components* of the network (user agents and origin servers) communicate via a standardized interface (e.g., HTTP) and exchange *representations* of these resources (the actual documents conveying the information).
- In addition to URIs, Internet media types, request and response codes HTTP has a rich vocabulary of operations:

  GET POST PUT DELETE etc.

  REST uses these operations and existing features of the well-defined HTTP protocol.

REST vs. SOAP

- SOAP (**Simple Object Access Protocol)** encourages each application designer to define new, application-specific operations that supplant HTTP operations:

  getUsers()

  getNewUsersSince(date SinceDate)

  savePurchaseOrder(string CustomerID, string PurchaseOrderID) etc.

- This additive, "re-invention of the wheel" vocabulary — defined on the spot and subject to individual judgment or preference — disregards many of HTTP's existing capabilities, such as authentication, caching, and content-type negotiation.
- The advantage of SOAP over REST comes from this same limitation: Since it does not take advantage of HTTP conventions, SOAP works equally well over raw TCP, named pipes, message queues etc.

Web services with PHP

- As we already know, RESTful web services can be written in PHP and contacted by the browser through Ajax or JSONP

- The content provided by the service can be of many different types:

Content ("MIME") types

Example: Exponent web service

- Write a web service that accepts a base and exponent and outputs base raised to the exponent power.
- For example, the following “query” should output 81 :

http://example.com/exponent.php?base=3&exponent=4

- Solution:

```
header("Content-type: text/plain");
$base = $_GET["base"];
$exp = $_GET["exponent"];
$result = pow($base, $exp);
print $result;
```

Displaying partial HTML page returned from PHP Web service

- Java script code in the consumer application:

```
var ajax=new XMLHttpRequest();
ajax.onreadystatechange=function()  {
   if (ajax.readyState==4 && ajax.status==200)    {
      document.getElementById("txtHint").innerHTML
                                                         =ajax.responseText;
   }
};
ajax.open("GET","getuser.php?q="+str,true);
ajax.send();

<div id="txtHint"><b>Person info will be listed here.</b></div>
```

getuser.php

```
<?php
$q=$_GET["q"];
…
$sql="SELECT * FROM user WHERE id = '".$q."'";
$result = query ($sql);
echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
</tr>";
…
?>
```

getuser.php

```
<?php
…
while($row = mysql_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "<td>" . $row['Age'] . "</td>";
echo "</tr>";
  }
echo "</table>";
?>
```

PHP graphics

- **PHP has graphics capabilities that can dynamically generate images you can then display using HTML code.**
- With the help of a graphics library called GD (Graphics Draw), we can dynamically generate images in popular formats such as GIF, JPEG, and PNG, and either return them to a web browser for display or write them to a file on the server.
- This capability of PHP is extremely important because there is no notion of being able to “draw” on a web page purely through HTML.
- PHP allows you to “draw” on a portion of a page by performing graphics operations on an image, and then displaying that image on the page using the familiar <img> tag.

PHP graphics

- **PHP has graphics capabilities that can dynamically generate images you can then display using HTML code.**
- With the help of a graphics library called GD (Graphics Draw), we can dynamically generate images in popular formats such as GIF, JPEG, and PNG, and either return them to a web browser for display or write them to a file on the server.
- This capability of PHP is extremely important because ~~there is no notion of being able to "draw" on a web page purely through HTML.~~
- PHP allows you to "draw" on a portion of a page by performing graphics operations on an image, and then displaying that image on the page using the familiar <img> tag.

CAPTCHA code

```
define('CAPTCHA_NUMCHARS', 6);
// Generate the random pass-phrase
$pass_phrase = "";
for ($i = 0; $i < CAPTCHA_NUMCHARS; $i++) {
        $pass_phrase .= chr(rand(97, 122));
}
```

Generating distorted image

```
define('CAPTCHA_WIDTH', 100);
define('CAPTCHA_HEIGHT', 25);
$img = imagecreatetruecolor(CAPTCHA_WIDTH, CAPTCHA_HEIGHT);

// Set a white background with black text and gray graphics
$bg_color = imagecolorallocate($img, 255, 255, 255); // white
$text_color = imagecolorallocate($img, 0, 0, 0); // black
$graphic_color = imagecolorallocate($img, 64, 64, 64); // dark gray

// Fill the background
imagefilledrectangle($img, 0, 0, CAPTCHA_WIDTH, CAPTCHA_HEIGHT, $bg_color);
// Draw some random lines
for ($i = 0; $i < 5; $i++) {
        imageline($img, 0, rand() % CAPTCHA_HEIGHT, CAPTCHA_WIDTH,
                                  rand() % CAPTCHA_HEIGHT, $graphic_color);
}
```

Generating distorted image

```
…
// Sprinkle in some random dots
for ($i = 0; $i < 50; $i++) {
        imagesetpixel($img, rand() % CAPTCHA_WIDTH,
                        rand() % CAPTCHA_HEIGHT, $graphic_color);
}

// Draw the pass-phrase string
imagettftext($img, 18, 0, 5, CAPTCHA_HEIGHT - 5,
                        $text_color, 'Courier New Bold.ttf', $pass_phrase);

// Output the image as a PNG using a header
header("Content-type: image/png");
imagepng($img);
```

imagepng()

- When you’re all finished drawing to an image, you can output it directly to the client web browser or to a file on the server.
- Either way, the end result is an image that can be used with the HTML <img> tag for display on a web page.
- If you elect to generate a PNG image directly to memory (i.e., no filename), then you must also call the header() function to have it delivered to the browser via a header.

Generating random image on the server with PHP: example

- Code is in 02.04.games captcha.php

- Graphic packages are now installed in the lab: link

Defining constants (in file appvars.php)

```
<?php
// Define application constants
define('MM_UPLOADPATH', 'images/');
define('MM_MAXFILESIZE', 32768);      // 32 KB
define('MM_MAXIMGWIDTH', 120);        // 120 pixels
define('MM_MAXIMGHEIGHT', 120);       // 120 pixels
?>
```

To reuse PHP script

```
require_once('header.php');
require_once('appvars.php');
```

Submitting page to itself

- $_SERVER['PHP_SELF'] simply returns a pathname of the current page on the server being executed.
- So if you had a PHP script and needed to send a form to the same page you could use:

```
<form action="<?php echo($_SERVER['PHP_SELF']); ?>"
method="post">
```

Example: signup page

```
<p>Please enter your username and desired password to sign up.</p>
<form method="post" action ="<?php Echo($_SERVER['PHP_SELF']); ?>" >
  <fieldset>
    <label for="username">Username:</label>
    <input type="text" name="username"
                value="<?php if (!empty($username)) echo $username; ?>" /><br />
    <label for="password1">Password:</label>
    <input type="password" name="password1" /><br />
    <label for="password2">Password (retype):</label>
    <input type="password" name="password2" /><br />
  </fieldset>
  <input type="submit" value="Sign Up" name="submit" />
</form>
```

PHP code to produce different pages, depending on the status

```
if (isset($_POST['submit'])) {
 // Grab the profile data from the POST
 $username = trim($_POST['username']);
 $password1 =trim($_POST['password1']);
 $password2 = trim($_POST['password2']);

 // Make sure someone isn't already registered using this username
 …
 if ( user does not exist)  {
   add user to a database
 }
 else {
   echo '<p class="error">The user with this name already exists. ‘.
                'Please choose a different name.</p>‘';
 }
}
```

Uploading files (images) and storing them on server

```
<input type="file" id="new_picture" name="new_picture" />
<?php if (!empty($old_picture)) {
        echo '<img class="profile" src="' . MM_UPLOADPATH .
                        $old_picture . '" alt="Profile Picture" />';
 }
?>
```

- Full code in mismatch_source/editprofile.php

PHP image upload code

```
$target = MM_UPLOADPATH . basename($new_picture);
if (move_uploaded_file($_FILES['new_picture']['tmp_name'], $target)) {
   chmod($target, 0755);     //unix server
// The new picture file move was successful, now make sure any old picture is deleted

  if (!empty($old_picture) && ($old_picture != $new_picture)) {
    @unlink(MM_UPLOADPATH . $old_picture);
  }
}
else {
// The new picture file move failed, so delete the temporary file and set the error flag
 @unlink($_FILES['new_picture']['tmp_name']);
 $error = true;
 echo '<p class="error">Sorry, there was a problem uploading your picture.</p>';
}
```

HTTP authentication

- The idea behind HTTP authentication is that the server withholds a protected web page, and then asks the browser to prompt the user for a user name and password.
- If the user enters these correctly, the browser goes ahead and sends along the page.
- This dialog between browser and server takes place through headers, which are little text messages with specific instructions on what is being requested or delivered.
- Headers are actually used every time you visit a web page, not just when authentication is required.

Delivering a normal, unprotected web page

- The browser requests a page from the server by sending a couple of headers to identify the file being requested and the host name of the server.
- The server responds with a collection of headers, followed by the requested page.
- The browser receives the headers and the page, and renders the HTML code for the page.

Headers example

- POST /mail/ca/u/0/?ui=2&ik=107df84a1e&rid=mail%3Ai.680f.0.1&view=cv&th=13c04134dc1284ab&th=13c0255dc95e706
- host:mail.google.com
- scheme:https
- version:HTTP/1.1
- accept:*/*
- …
- ui:2
- ik:107df84a1e
- rid:mail:i.680f.0.1
- view:cv
- prf:1
- nsc:1

- <u>Response Header</u>
- cache-control:no-cache, no-store, max-age=0, must-revalidate
- content-encoding:gzip
- content-length:10445
- content-type:text/javascript; charset=UTF-8
- date:Fri, 04 Jan 2013 13:20:42 GMT
- expires:Fri, 01 Jan 1990 00:00:00 GMT
- pragma:no-cache
- server:GSE
- status:200 OK
- version:HTTP/1.1
- x-content-type-options:nosniff
- x-frame-options:SAMEORIGIN
- x-xss-protection:1; mode=block

Authentication code example

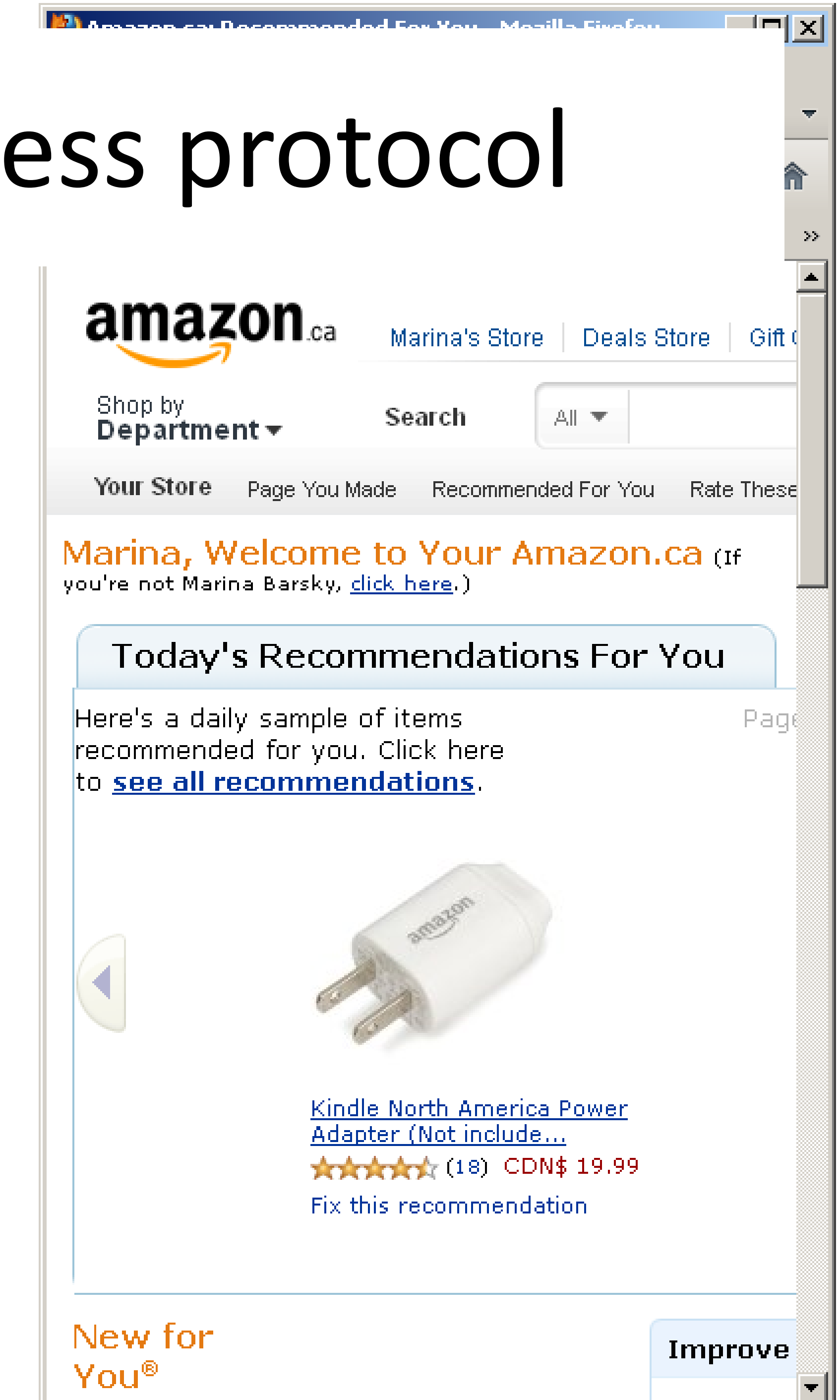
```
<?php
// User name and password for authentication
  $username = 'rock';
  $password = 'roll';
  if (!isset($_SERVER['PHP_AUTH_USER']) ||
          !isset($_SERVER['PHP_AUTH_PW']) ||
          ($_SERVER['PHP_AUTH_USER'] != $username) ||
          ($_SERVER['PHP_AUTH_PW'] != $password)) {
// The user name/password are incorrect so send the authentication headers
          header('HTTP/1.1 401 Unauthorized');
          header('WWW-Authenticate: Basic realm="Guitar Wars"');
          exit('Sorry, you must enter a valid user name and password to
access this page.');
  }
?>
```

USER IDENTIFICATION

HTTP is a stateless protocol

- it simply allows a browser to request a single document from a web server

- How Amazon knows who I am, each time I am requesting a different page
- *How does a client uniquely identify itself to a server, and how does the server provide specific content to each client?*



Marina, Welcome to Your Amazon.ca (If you're not Marina Barsky, click here.)

Today's Recommendations For You

Here's a daily sample of items recommended for you. Click here to see all recommendations.

Kindle North America Power Adapter (Not include...
★★★★☆ (18) CDN$ 19.99
Fix this recommendation

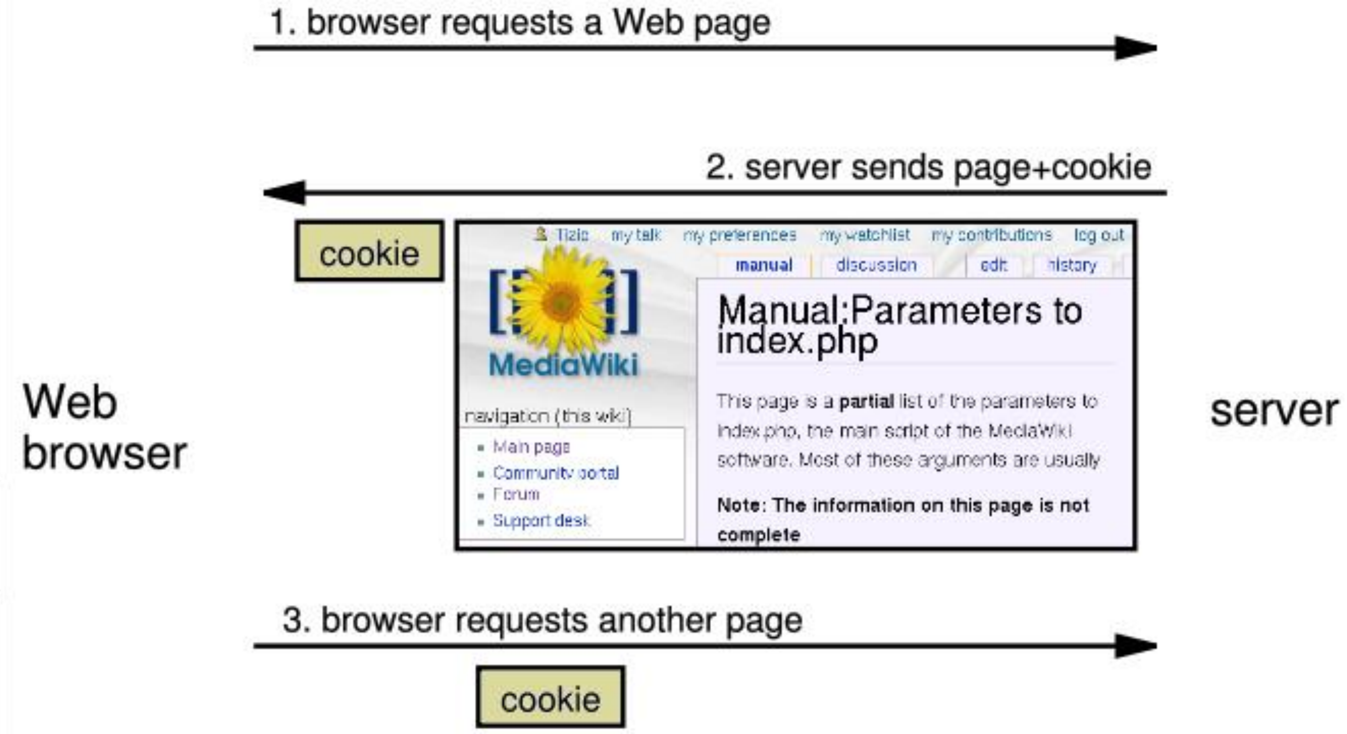
Stateful client/server interaction

- today we'll learn about pieces of data called cookies used to work around this problem, which are used as the basis of higher-level sessions between clients and servers

What is a cookie?

- cookie: a small amount of information sent by a server to a browser, and then sent back by the browser on future page requests
- cookies have many uses:
  - authentication
  - user tracking
  - maintaining user preferences, shopping carts, etc.
- a cookie's data consists of a single name/value pair, sent in the header of the client's HTTP GET or POST request

How cookies are sent


1. browser requests a Web page
2. server sends page+cookie
cookie
Web browser
server
3. browser requests another page
cookie
Tizio my talk my preferences my watchlist my contributions log out
manual discussion edit history
MediaWiki
navigation (this wiki)
Main page
Community portal
Forum
Support desk

Manual:Parameters to index.php

This page is a **partial** list of the parameters to index.php, the main script of the MediaWiki software. Most of these arguments are usually

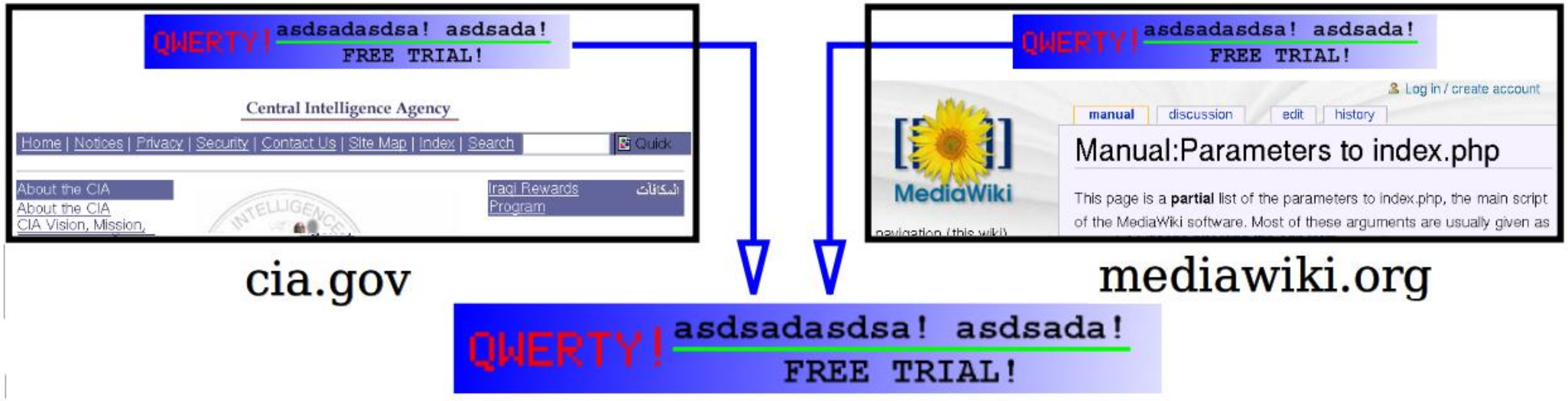
Note: The information on this page is not complete

- when the browser requests a page, the server may send back a cookie(s) with it
- if your server has previously sent any cookies to the browser, the browser will send them back on subsequent requests

Myths about cookies

- Myths:
  - Cookies are like worms/viruses and can erase data from the user's hard disk.
  - Cookies are a form of spyware and can steal your personal information.
  - Cookies generate popups and spam.
  - Cookies are only used for advertising.
- Facts:
  - Cookies are only data, not program code.
  - Cookies cannot erase or read information from the user's computer.
  - Cookies are usually anonymous (do not contain personal information).
  - Cookies CAN be used to track your viewing habits on a particular site.

A "tracking cookie"


QWERTY! asdsadasdsa! asdsada! FREE TRIAL!
Central Intelligence Agency
Home | Notices | Privacy | Security | Contact Us | Site Map | Index | Search
Quick
About the CIA
About the CIA
CIA Vision, Mission,
Iraqi Rewards Program
اكتشافات
QWERTY! asdsadasdsa! asdsada! FREE TRIAL!
Log in / create account
manual
discussion
edit
history
MediaWiki
Manual:Parameters to index.php
This page is a partial list of the parameters to index.php, the main script of the MediaWiki software. Most of these arguments are usually given as
navigation (this wiki)
QWERTY! asdsadasdsa! asdsada! FREE TRIAL!

cia.gov

mediawiki.org

lotofbanners.com

- an advertising company can put a cookie on your machine when you visit one site, and see it when you visit another site that also uses that advertising company
- therefore they can tell that the same person (you) visited both sites
- can be thwarted by telling your browser not to accept "third-party cookies"

Where are the cookies on my computer?

- each is stored as a .txt file similar to the site's domain name

- IE: *HomeDirectory*\Cookies e.g. C:\Documents and Settings\jsmith\Cookies
- Chrome: C:\Users\username\AppData\Local\Google\Chrome\User Data\Default
- Firefox: *HomeDirectory*\.mozilla\firefox\*???*.default\cookies.txt view cookies in Firefox preferences: Privacy, Show Cookies...

How long does a cookie exist?

- session cookie : the default type; a temporary cookie that is stored only in the browser's memory
  - when the browser is closed, temporary cookies will be erased
  - can not be used for tracking long-term information
  - safer, because no programs other than the browser can access them
- persistent cookie : one that is stored in a file on the browser's computer
  - can track long-term information
  - potentially less secure, because users (or programs they run) can open cookie files, see/change the cookie values, etc.

Reminder: cookies in JavaScript

- JS has a global document.cookie field (a string)
- you can manually set/get cookie data from this field (sep. by ;), and it will be changed

Reminder: Cookies in JavaScript

```
// setting two cookies
document.cookie = "username=smith";
document.cookie = "password=12345";
document.cookie = "age=29; expires=Thu, 01-Jan-1970 00:00:01 GMT";
// deleting a cookie ...
```

Reminder: Cookies in JavaScript

```
// (later)
var allCookies = document.cookie.split(";");
// ["username=smith", "password=12345"]
for (var i = 0; i < allCookies.length; i++)
{
        var eachCookie = allCookies[i].split("=");
        // ["username", "smith"]
        var cookieName = eachCookie[0]; // "username"
        var cookieValue = eachCookie[1]; // "smith" ... }
```

Setting a cookie in PHP

```
setcookie("name", "value");
setcookie("username", "marina");
setcookie("favoritecolor", "blue");
```

- setcookie causes your script to send a cookie to the user's browser
- setcookie must be called before any output statements (HTML blocks, print, or echo)
- you can set multiple cookies (20-50) per user, each up to 3-4K bytes

Retrieving information from a cookie in PHP

```
# retrieve value of the cookie
$variable = $_COOKIE["name"];
if (isset($_COOKIE["username"])) {
 $username = $_COOKIE["username"];
 print("Welcome back, $username.\n");
} else {
print("Never heard of you.\n"); }
```

- any cookies sent by client are stored in $_COOKIES associative array
- use isset function to see whether a given cookie name exists

Setting a persistent cookie in PHP

setcookie("*name*", "*value*", *timeout*);

```
$expireTime = time() + 60*60*24*7;   # 1 week from now
setcookie("CouponNumber", "389752", $expireTime);
setcookie("CouponValue", "100.00", $expireTime);
```

- to set a persistent cookie, pass a third parameter for its timeout in seconds
- time function returns the current time in seconds
  - date function can convert a time in seconds to a readable date

Removing a persistent cookie

- setcookie("name", "", time() - 1);

- setcookie("CouponNumber", "", time() - 1);

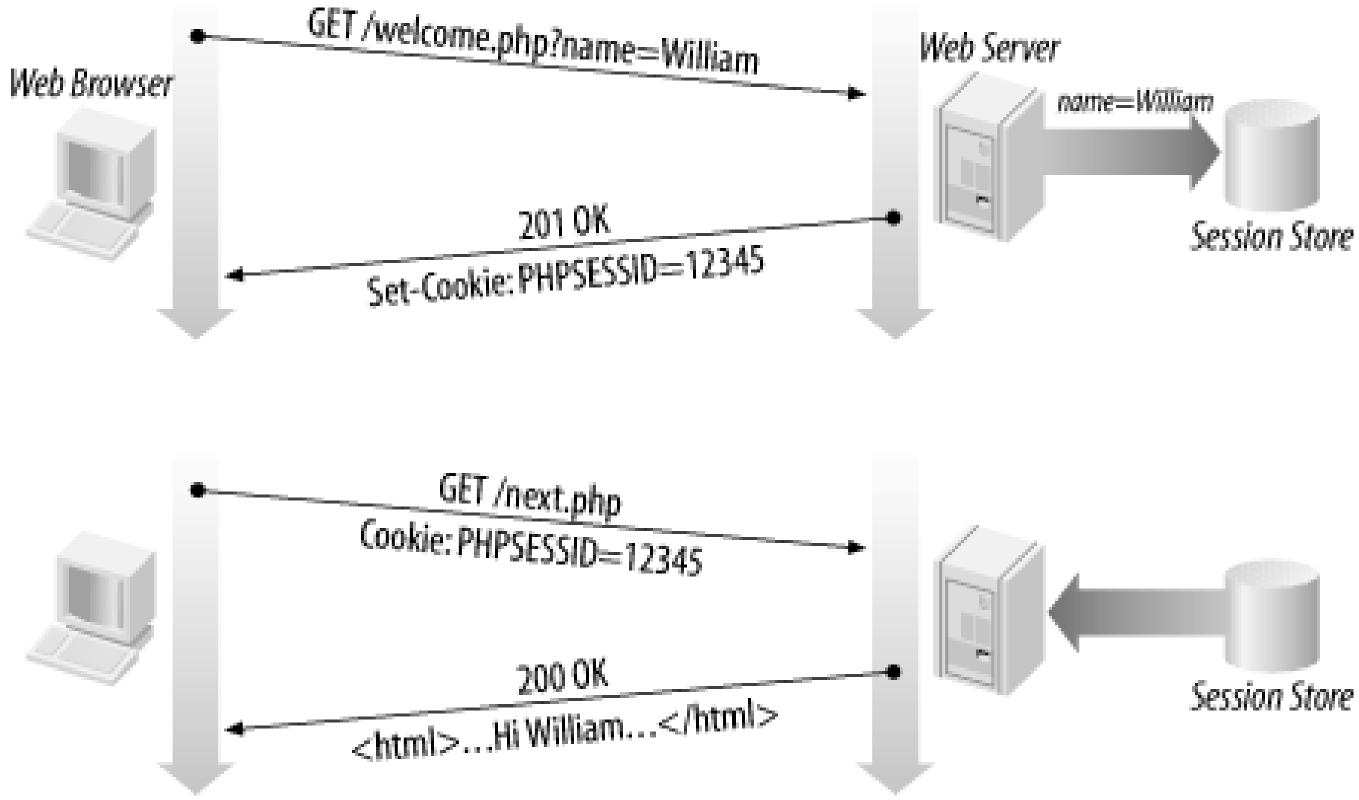
- if the server wants to remove a persistent cookie, it should set it again, passing a timeout that is prior to the present time

Sessions

- session: an abstract concept to represent a series of HTTP requests and responses between a specific Web browser and server
- HTTP doesn't support the notion of a session, but PHP does

- sessions vs. cookies: a cookie is data stored on the client
- a session's data is stored on the server (only 1 session per client)
- sessions are often built on top of cookies: the only data the client stores is a cookie holding a unique session ID
- on each page request, the client sends its session ID cookie, and the server uses this to find and retrieve the client's session data

How sessions are established



- client's browser makes an initial request to the server
- server notes client's IP address/browser, stores some local session data, and sends a session ID back to client
- client sends that same session ID back to server on future requests
- server uses session ID to retrieve the data for the client's session later, like a ticket given at a coat-check room

Sessions in PHP: session_start

session_start();

- session_start signifies your script wants a session with the user
  - must be called at the top of your script, before any HTML output is produced
- when you call session_start:
  - if the server hasn't seen this user before, a new session is created
  - otherwise, existing session data is loaded into $_SESSION associative array
  - you can store data in $_SESSION and retrieve it on future pages
- complete list of PHP session functions

Accessing session data

```
$_SESSION["name"] = value;              # store session data
$variable = $_SESSION["name"];          # read session data
if (isset($_SESSION["name"])) {         # check for session data
        if (isset($_SESSION["points"])) {
                $points = $_SESSION["points"];
                print("You've earned $points points.\n");
        } else {
                $_SESSION["points"] = 0;  # default
        }
}
```

- the $_SESSION associative array reads/stores all session data
- use isset function to see whether a given value is in the session

Where is session data stored?

- on the client, the session ID is stored as a cookie with the name PHPSESSID
- on the server, session data are stored as temporary files such as /tmp/sess_fcc17f071...
- you can find out (or change) the folder where session data is saved using the session_save_path function
- for very large applications, session data can be stored into a SQL database (or other destination) instead using the session_set_save_handler function

Session timeout

- because HTTP is stateless, it is hard for the server to know when a user has finished a session
- ideally, user explicitly logs out, but many users don't
- client deletes session cookies when browser closes
- server automatically cleans up old sessions after a period of time
  - old session data consumes resources and may present a security risk
  - adjustable in PHP server settings or with session_cache_expire function
  - you can explicitly delete a session by calling session_destroy

Browsers that don't support cookies

- session_start(); # same as usual
- # Generate a URL to link to one of our site's pages
  $orderUrl = "/order.php*?PHPSESSID=*" . *session_id()*;
- if a client's browser doesn't support cookies, it can still send a session ID as a query string parameter named PHPSESSID
  - this is done automatically; session_start detects whether the browser supports cookies and chooses the right method
- if necessary (such as to build a URL for a link on the page), the server can find out the client's session ID by calling the session_id function

Ending a session

```
session_destroy();
```

- session_destroy ends your current session
- potential problem: if you call session_start again later, it sometimes reuses the same session ID/data you used before
- if you may want to start a completely new empty session later, it is best to flush out the old one:

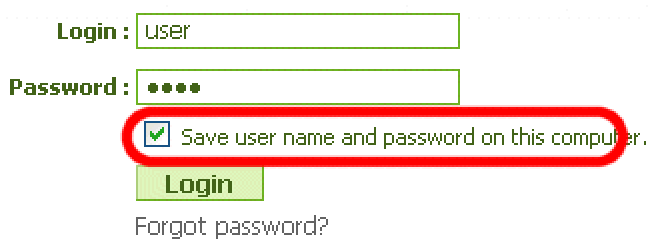
```
session_destroy();
session_regenerate_id(TRUE);
# flushes out session ID number session_start();
```

Implementing user logins

- many sites have the ability to create accounts and log in users
- most apps have a database of user accounts
- when you try to log in, your name/pw are compared to those in the database

"Remember Me" feature

- How might an app implement a "Remember Me" feature, where the user's login info is remembered and reused when the user comes back later?
- Is this stored as session data? Why or why not?
- What concerns come up when trying to remember data about the user who has logged in?



Login : user

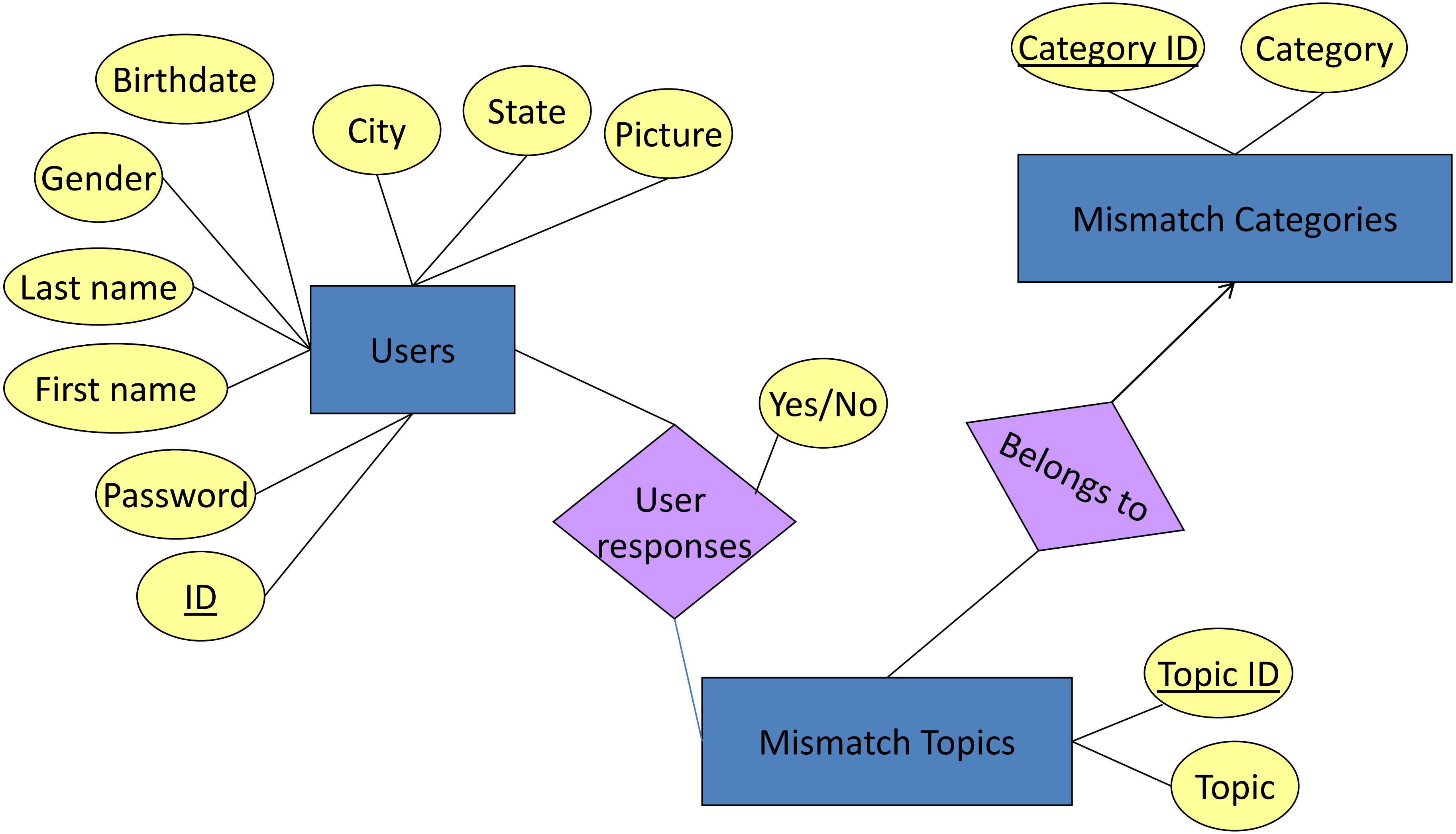
Password : ●●●●

☑ Save user name and password on this computer.

link

PHP WEB APP EXAMPLE

Designing PHP-DB application: ER diagram of Perfect Mismatch



Converting into tables

- Each entity is converted into a table

Users (ID, Password, FirstName, LastName, Gender, Birthdate, City, State, Picture)

Categories (ID, Category)

Topics (ID, Topic)

- Each relationship is converted into a table:

Resposes (UserID, TopicID, response)

TopicCategories (TopicID, CategoryID)

Pages flow

- Home page (index.php):
  - General info
  - Login
  - Signup
- Signup
  - New user name
  - New password
  - Password confirmation
- Log-in
  - Username
  - Password

- View profile
- Edit profile
- Questionnaire
- My mismatch

Full source code: link