PHP Sessions

CIS-1152 Adv Web Dev

Steve Ruegsegger

Modified with permission by Peter Chapin

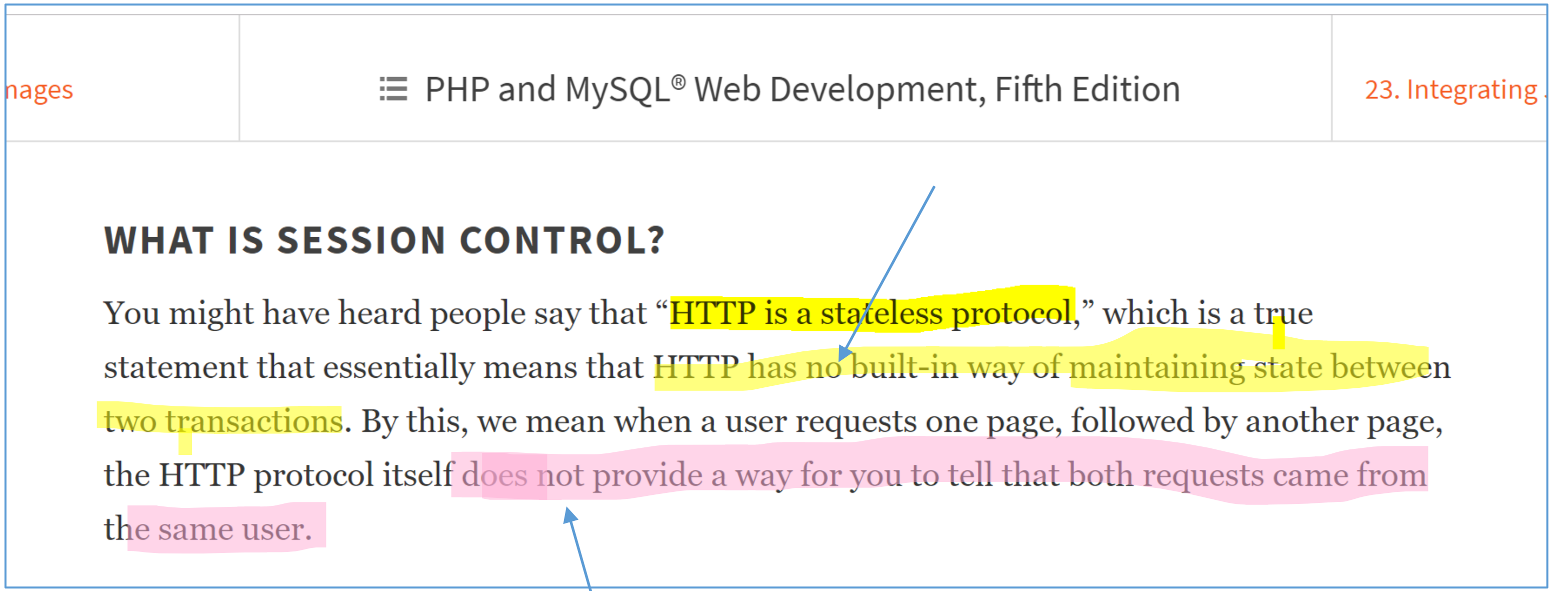
Outline

1. States
2. PHP Session
3. Cookie
4. Web Security

1. State

Stateless

Purple book, ch 22



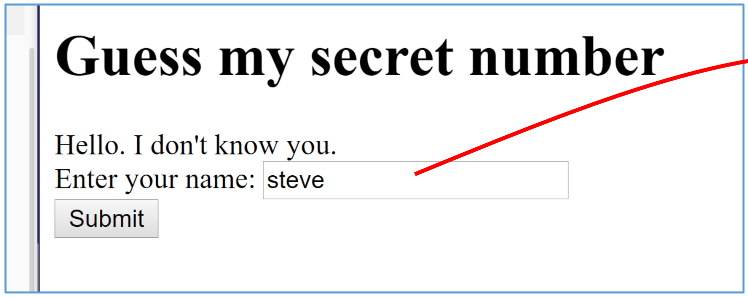
nages | ☰ PHP and MySQL® Web Development, Fifth Edition | 23. Integrating

WHAT IS SESSION CONTROL?

You might have heard people say that "HTTP is a stateless protocol," which is a true statement that essentially means that HTTP has no built-in way of maintaining state between two transactions. By this, we mean when a user requests one page, followed by another page, the HTTP protocol itself does not provide a way for you to tell that both requests came from the same user.

Demo of stateless-ness

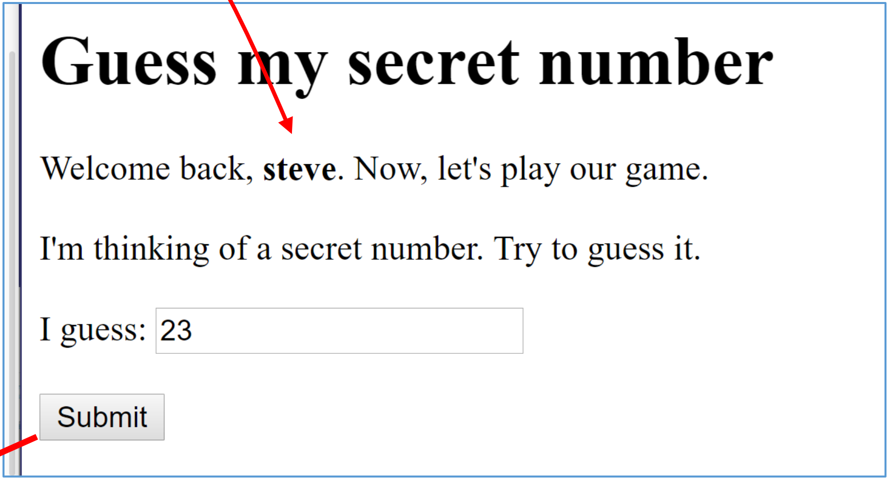
L12_guessing_game_0 – problem
L12_guessing_game_1 – sessions
L12_guessing_game_2 – cookies



Guess my secret number

Hello. I don't know you.
Enter your name: steve
Submit

How did this name get here?



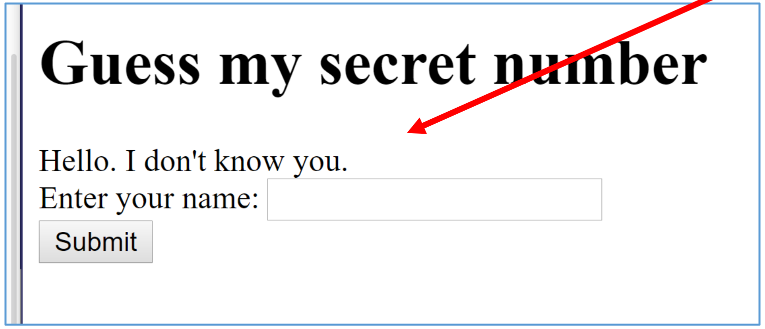
Guess my secret number

Welcome back, **steve**. Now, let's play our game.

I'm thinking of a secret number. Try to guess it.

I guess: 23

Submit



Guess my secret number

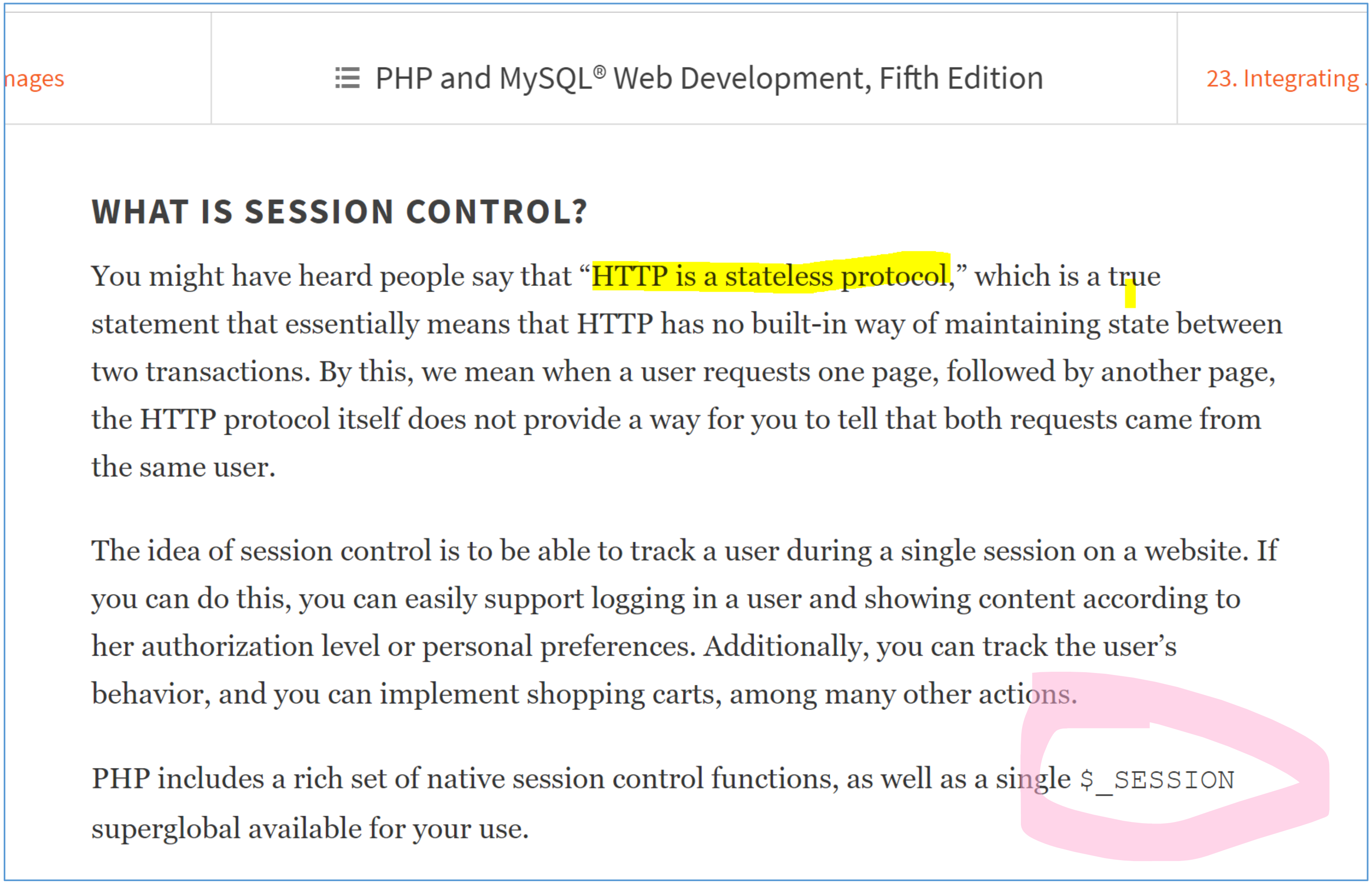
Hello. I don't know you.
Enter your name:
Submit

Why is it gone now? (step 3)

2. PHP Sessions

This is unique to PHP.

PHP $_SESSION


nages | PHP and MySQL® Web Development, Fifth Edition | 23. Integrating

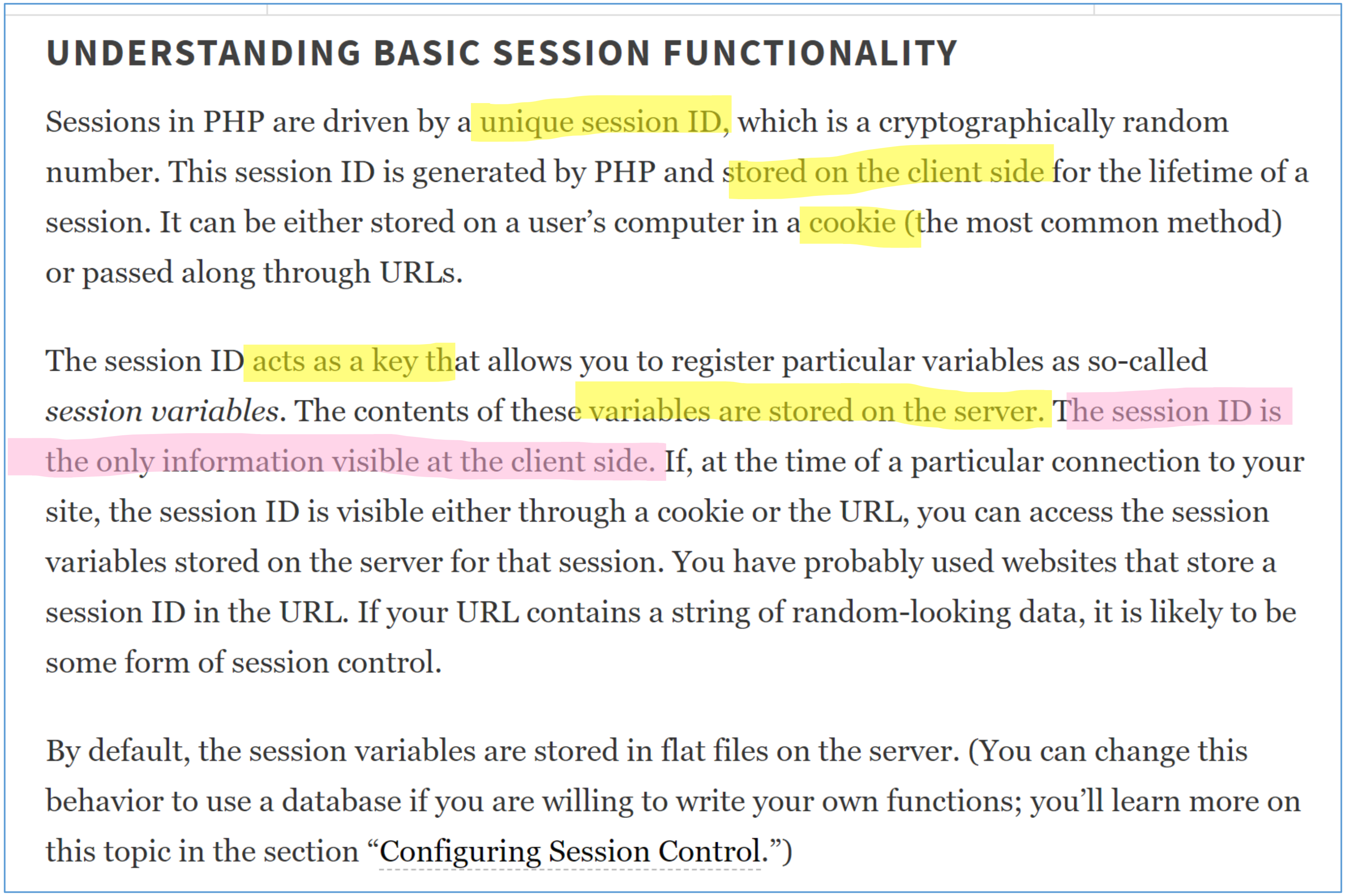
WHAT IS SESSION CONTROL?

You might have heard people say that "HTTP is a stateless protocol," which is a true statement that essentially means that HTTP has no built-in way of maintaining state between two transactions. By this, we mean when a user requests one page, followed by another page, the HTTP protocol itself does not provide a way for you to tell that both requests came from the same user.

The idea of session control is to be able to track a user during a single session on a website. If you can do this, you can easily support logging in a user and showing content according to her authorization level or personal preferences. Additionally, you can track the user's behavior, and you can implement shopping carts, among many other actions.

PHP includes a rich set of native session control functions, as well as a single `$_SESSION` superglobal available for your use.

What is PHP Session?



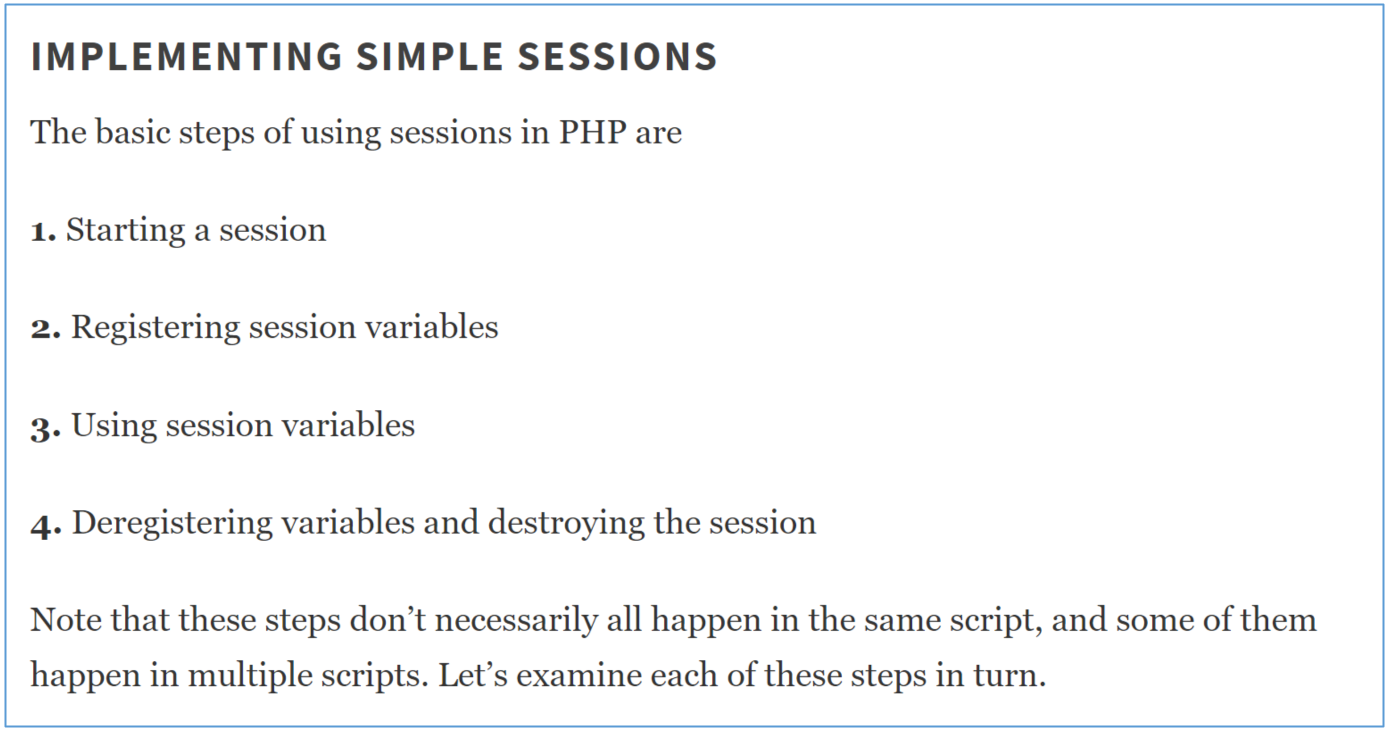
UNDERSTANDING BASIC SESSION FUNCTIONALITY

Sessions in PHP are driven by a unique session ID, which is a cryptographically random number. This session ID is generated by PHP and stored on the client side for the lifetime of a session. It can be either stored on a user's computer in a cookie (the most common method) or passed along through URLs.

The session ID acts as a key that allows you to register particular variables as so-called *session variables*. The contents of these variables are stored on the server. The session ID is the only information visible at the client side. If, at the time of a particular connection to your site, the session ID is visible either through a cookie or the URL, you can access the session variables stored on the server for that session. You have probably used websites that store a session ID in the URL. If your URL contains a string of random-looking data, it is likely to be some form of session control.

By default, the session variables are stored in flat files on the server. (You can change this behavior to use a database if you are willing to write your own functions; you'll learn more on this topic in the section "Configuring Session Control.")

PHP Session



IMPLEMENTING SIMPLE SESSIONS

The basic steps of using sessions in PHP are

1. Starting a session

2. Registering session variables

3. Using session variables

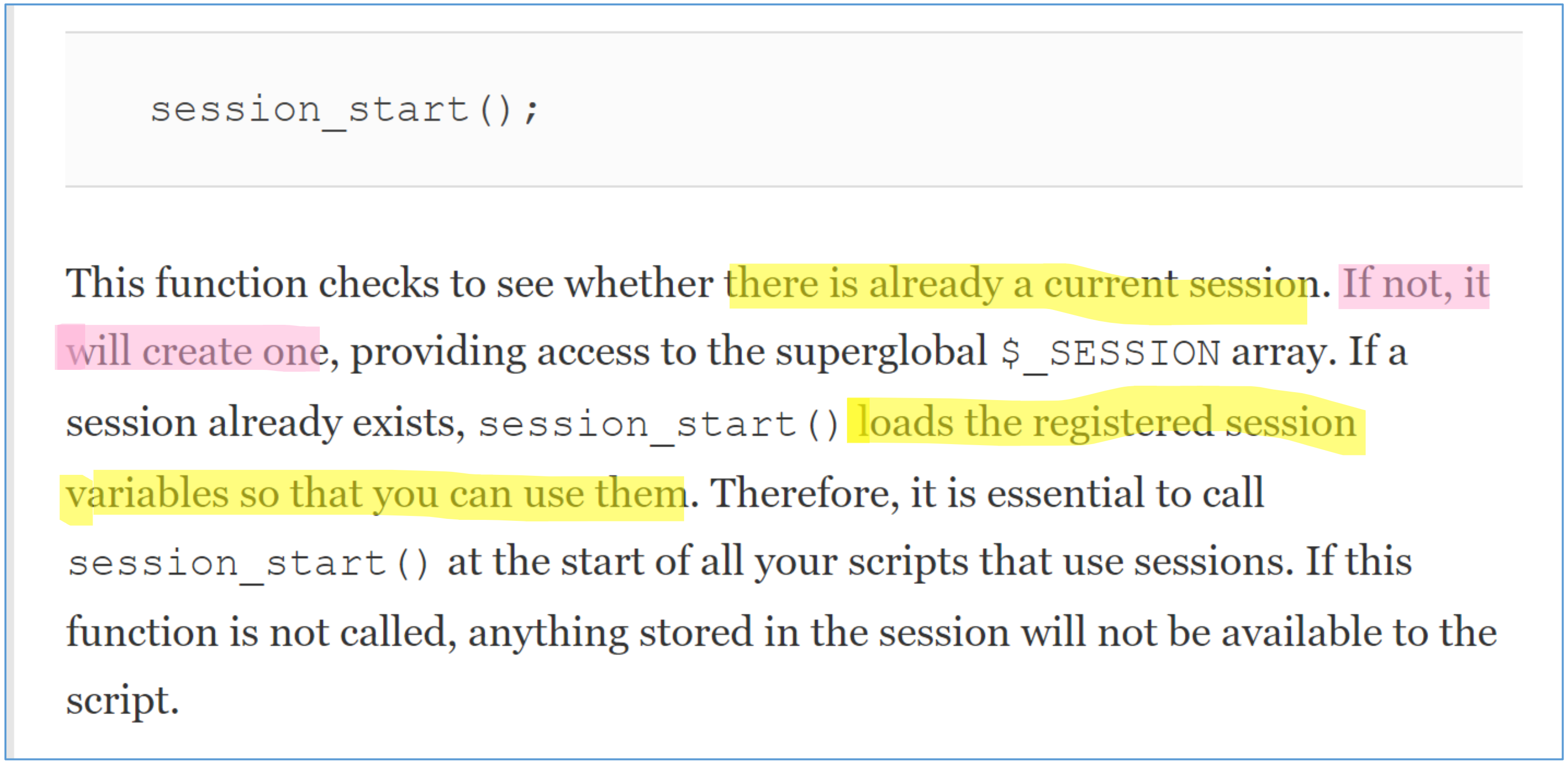
4. Deregistering variables and destroying the session

Note that these steps don't necessarily all happen in the same script, and some of them happen in multiple scripts. Let's examine each of these steps in turn.

Time for a demo...

Session start

- Start with session_start()

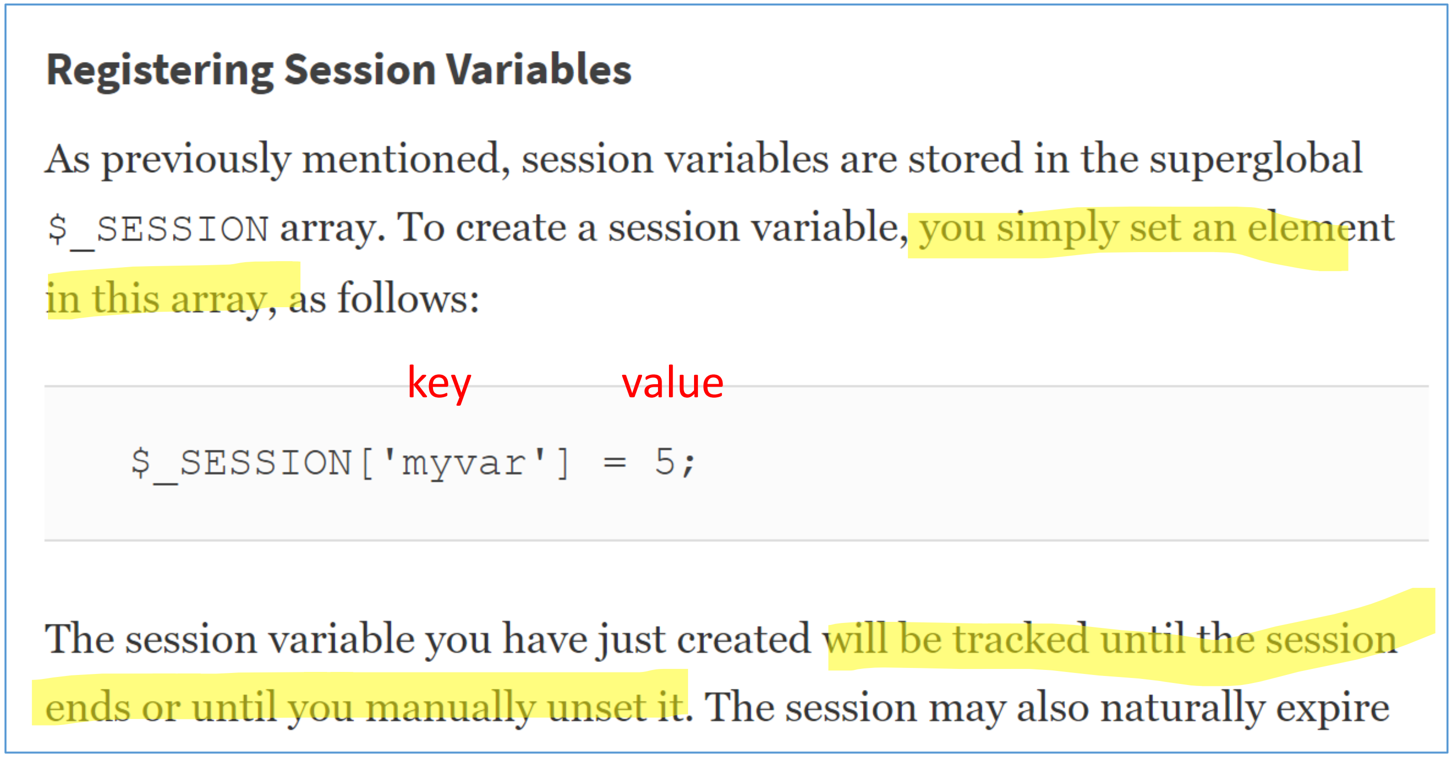


```
session_start();
```

This function checks to see whether there is already a current session. If not, it will create one, providing access to the superglobal `$_SESSION` array. If a session already exists, `session_start()` loads the registered session variables so that you can use them. Therefore, it is essential to call `session_start()` at the start of all your scripts that use sessions. If this function is not called, anything stored in the session will not be available to the script.

Creating new session vars

- Super easy! $_SESSION is just a *hash array*.
- Where does the $_SESSION global array come from????



Registering Session Variables

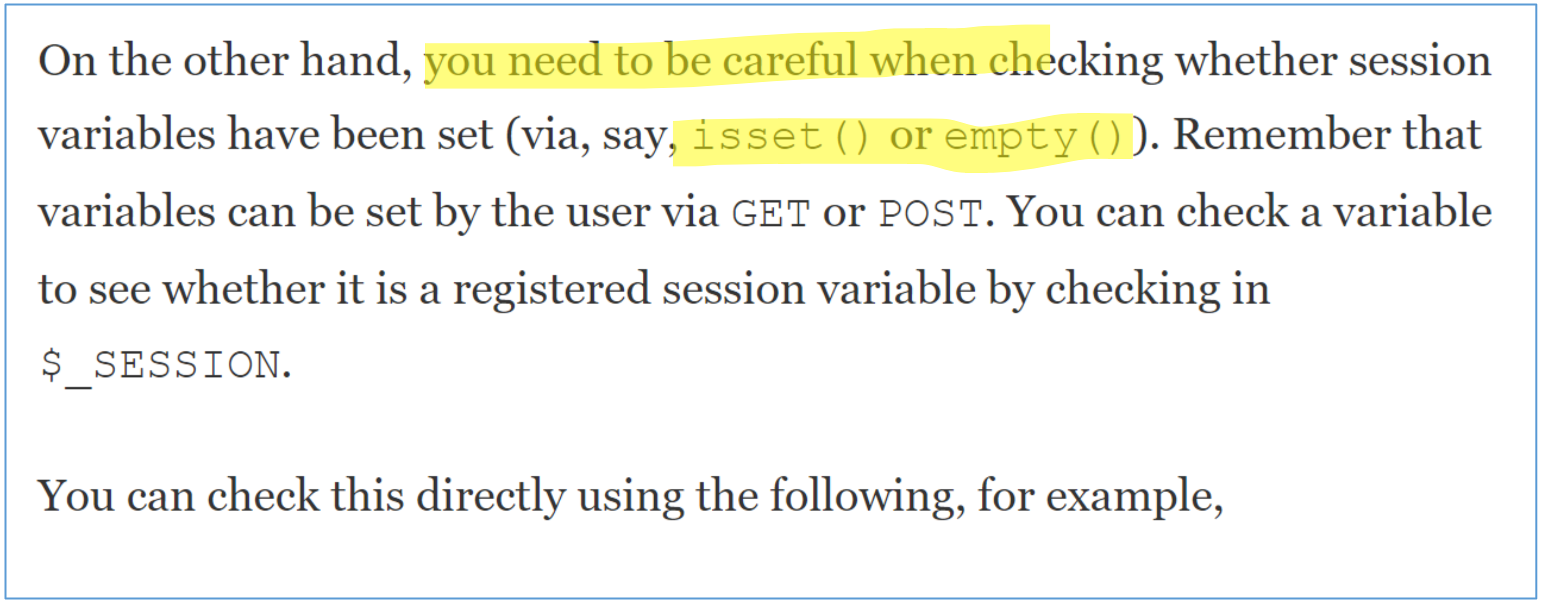
As previously mentioned, session variables are stored in the superglobal `$_SESSION` array. To create a session variable, you simply set an element in this array, as follows:

```
$_SESSION['myvar'] = 5;
```

The session variable you have just created will be tracked until the session ends or until you manually unset it. The session may also naturally expire

Using Session variables

- Super easy! $_SESSION is just a *hash array*.
- **But**, *remember*, the key must exist in the *hash* or PHP throws an error.


On the other hand, you need to be careful when checking whether session variables have been set (via, say, `isset()` or `empty()`). Remember that variables can be set by the user via `GET` or `POST`. You can check a variable to see whether it is a registered session variable by checking in `$_SESSION`.

You can check this directly using the following, for example,

```
if (isset($_SESSION['name'])) { $name = $_SESSION['name']; }
$name = $_SESSION['name'] ?? "";
```

Removing stored session items

- Remove 1 var:

```
unset($_SESSION['name']);
```

- Remove entire session:

```
session_destroy();
```

Tricks

- I find tricky that I can have 3 variables for the same item:
  - $_POST – how the user defines the value OR $stmt->fetch() - retrieval from MySQL **init**
  - $_SESSION – what I want to store and retrieve over & over **restore**
  - $variable – what I want to use in the code **use**
- Why not use just 1?
  - Because there are 2 ways to get a value initially and a 3rd way to retrieve that value over and over. I want 1 variable to use in the code.
- So the logic goes like this:
  - Is the program "state" at the place to initially get the value from POST or MySQL? *i.e.* something is missing in $_SESSION
  - If so, get it (from POST or MySQL) and then *put* in SESSION array.
  - If not, then assign the PHP $var from SESSION key/value.
  - Throw an error if something unexpected happens.

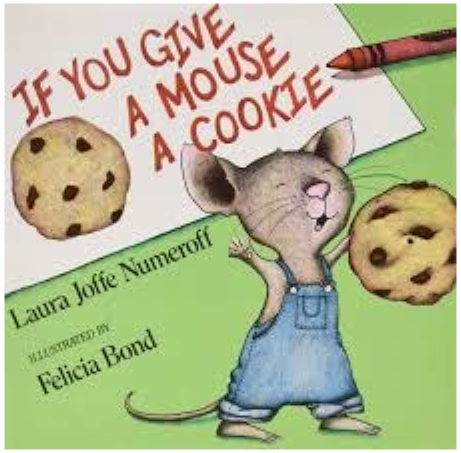
In class Demo

Number guessing

L12_guessing_game_1.php

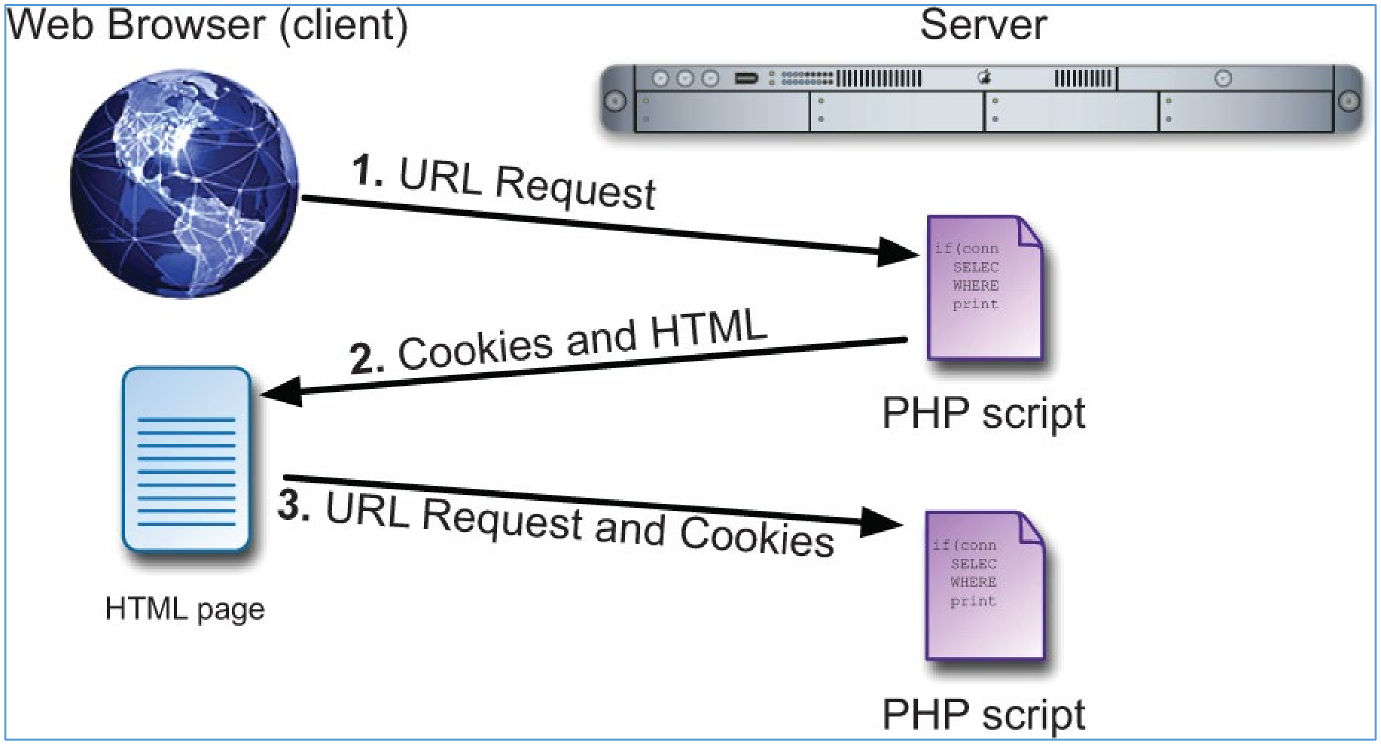
3. Cookies

If you give a mouse a ...

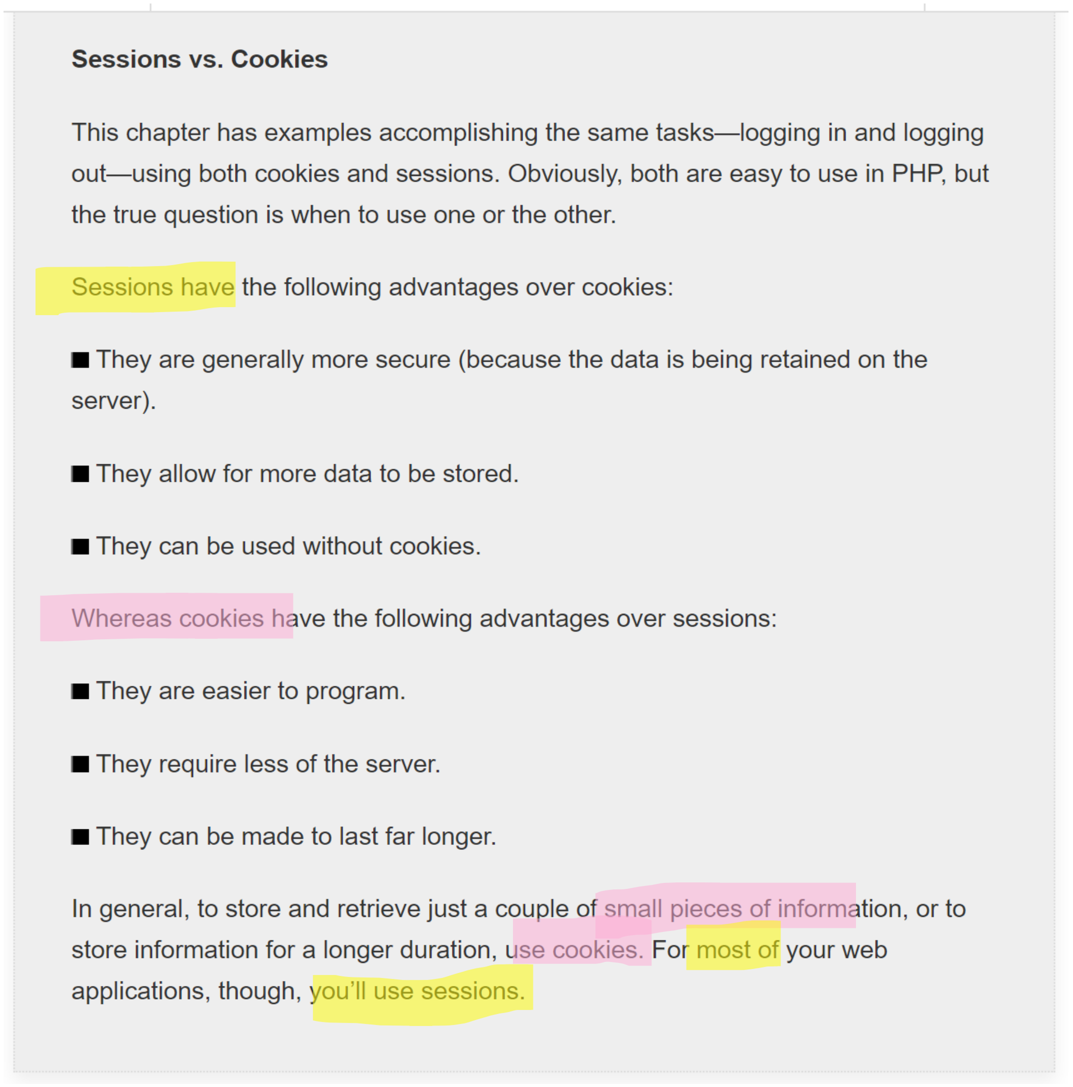


Cookies

- Persistent variables per client browser
- Key/values stored on the *client*



PHP Session vs Cookies



Sessions vs. Cookies

This chapter has examples accomplishing the same tasks—logging in and logging out—using both cookies and sessions. Obviously, both are easy to use in PHP, but the true question is when to use one or the other.

Sessions have the following advantages over cookies:

■ They are generally more secure (because the data is being retained on the server).

■ They allow for more data to be stored.

■ They can be used without cookies.

Whereas cookies have the following advantages over sessions:

■ They are easier to program.

■ They require less of the server.

■ They can be made to last far longer.

In general, to store and retrieve just a couple of small pieces of information, or to store information for a longer duration, use cookies. For most of your web applications, though, you'll use sessions.

Using cookies

- Define: `setcookie(key,value);`
- Use: `$value=$_COOKIE['key'];`

- Of course, the *key* must be in the *hash* $_COOKIE, so use isset() and empty() and ?? as needed.

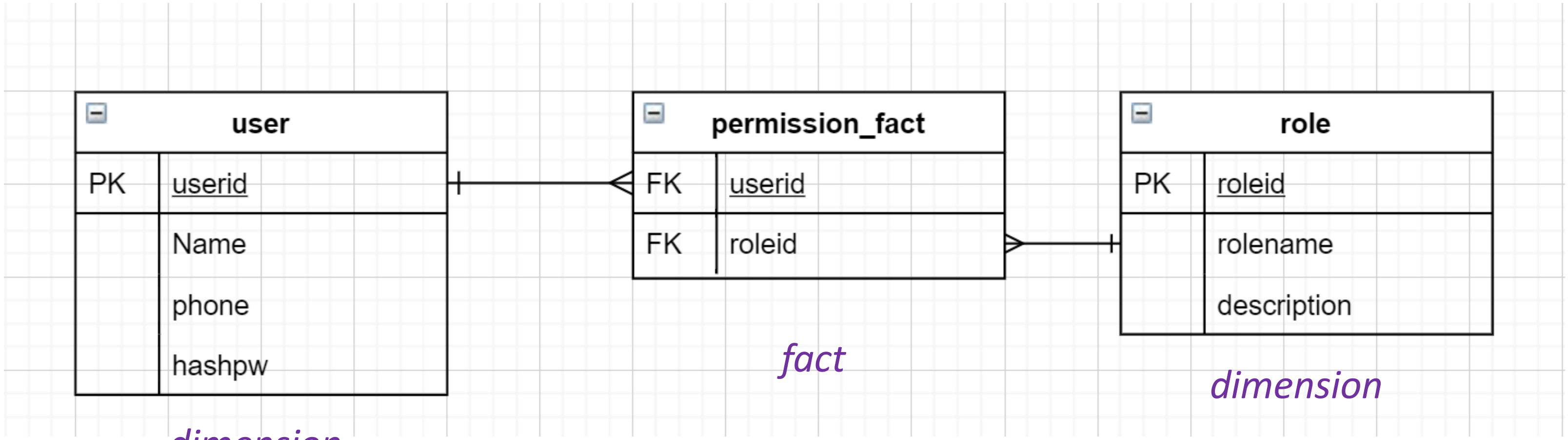
- Time for a demo...
  L10_guessing_game_2cookie.php

4. Security Measures

See Ullman (red) Ch 13.
Password hashing

Application: authentication and roles

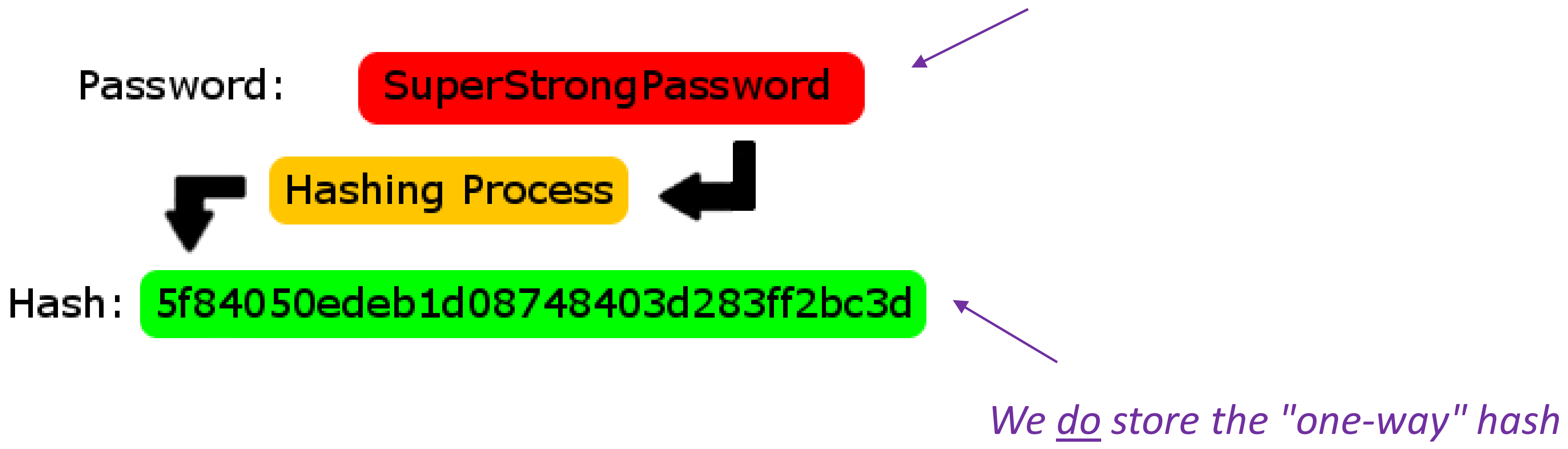
- Apps need protections, users, authentication, & roles.
- Enterprise apps must have different *users* with different *roles*: admin, IT, power users, regular users, just view reports
- How does a user get a role?
  - User login with username & pw
  - That userid is assigned the appropriate "role"
  - The "role" can "do things" in the app



l12 simple username role schema.sql

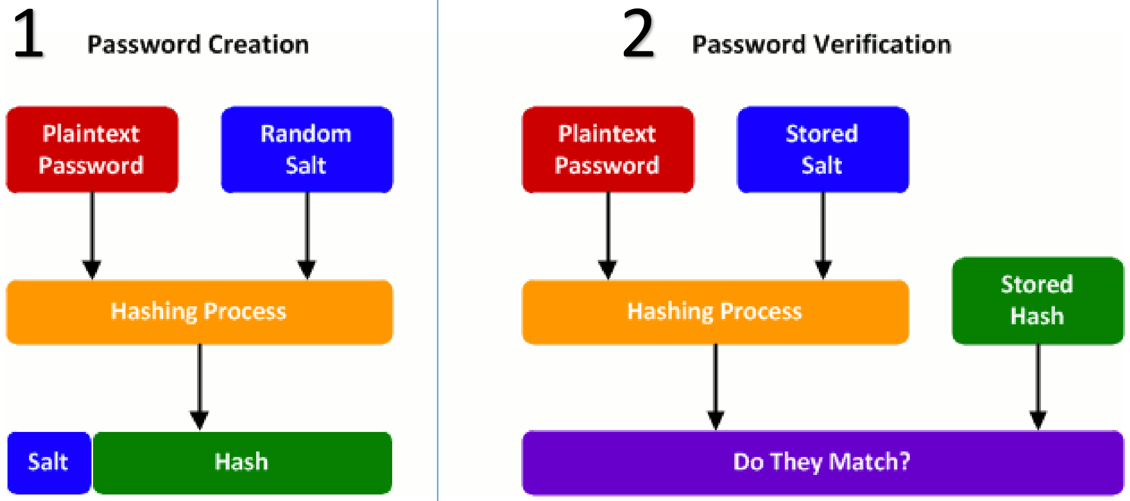
No Plaintext pws

- We do NOT want to store plaintext passwords in our database.
- If an external or inside-employee bad guy *steals* the database, they do not get the plaintext pws.
- We only store the HASHED password.


Do not store the plaintext pw
Password:
SuperStrongPassword
Hashing Process
Hash:
5f84050edeb1d08748403d283ff2bc3d
We do store the "one-way" hash

Use salt

- The salt is a random text string "added" to the pw.
- This makes the plaintext pw harder to "crack" via *brute-force* methods.
- It makes a single plaintext pw have thousands (millions) of alternative HASHES.
    - Even a powerful computer *can't* check millions of alternatives for each possible plaintext pw.



So, how do we do this?

- **password_hash()** – used to hash the password; includes the salt, so "we" don't have to manage it. Nice!
- **password_verify()** – used to verify a password against its hash (also manages the salt so we don't have to).
- **password_needs_rehash()** – used when a password needs to be rehashed.
- **password_get_info() –** returns the name of the hashing algorithm and various options used while hashing.

from user trying to authenticate

```
<?php
if (password_verify($password, $hash)) {
    // Success! -> store in Session
}
else {
    // Invalid credentials
}
```

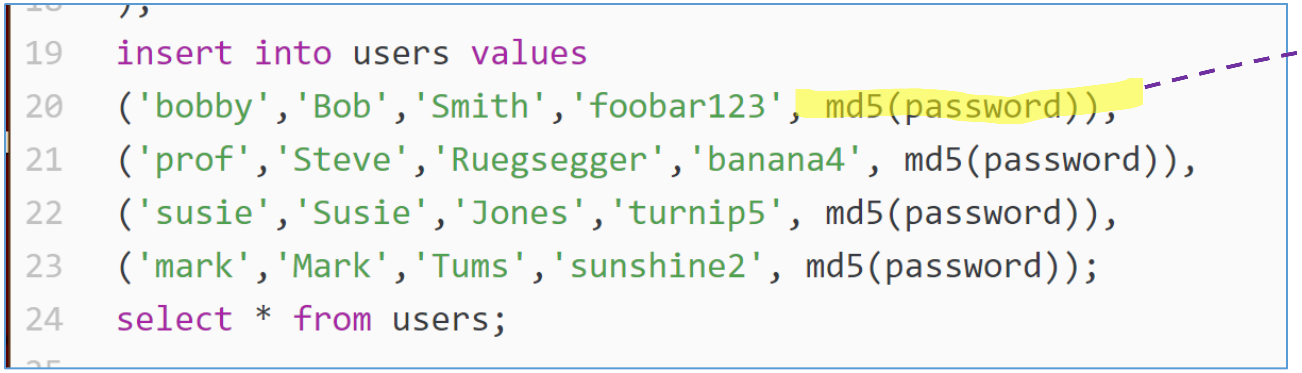
from db

https://bit.ly/3ceS4V3
https://bit.ly/2xpBSBy

Code example 1: sql setup

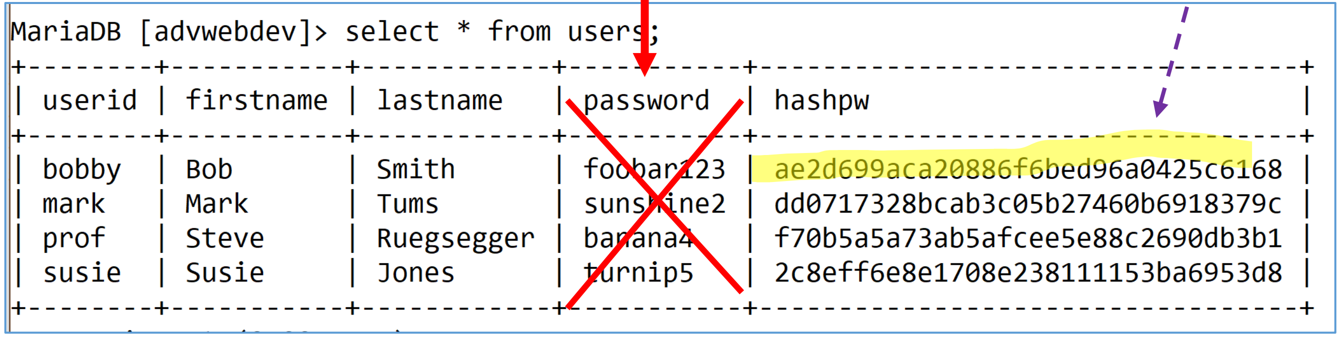
l12 simple username role schema.sql

- Setup mysql table schema (IT) from the ERD
- Use md5() function to create the hashed pw



```sql
);
insert into users values
('bobby','Bob','Smith','foobar123', md5(password)),
('prof','Steve','Ruegsegger','banana4', md5(password)),
('susie','Susie','Jones','turnip5', md5(password)),
('mark','Mark','Tums','sunshine2', md5(password));
select * from users;
```

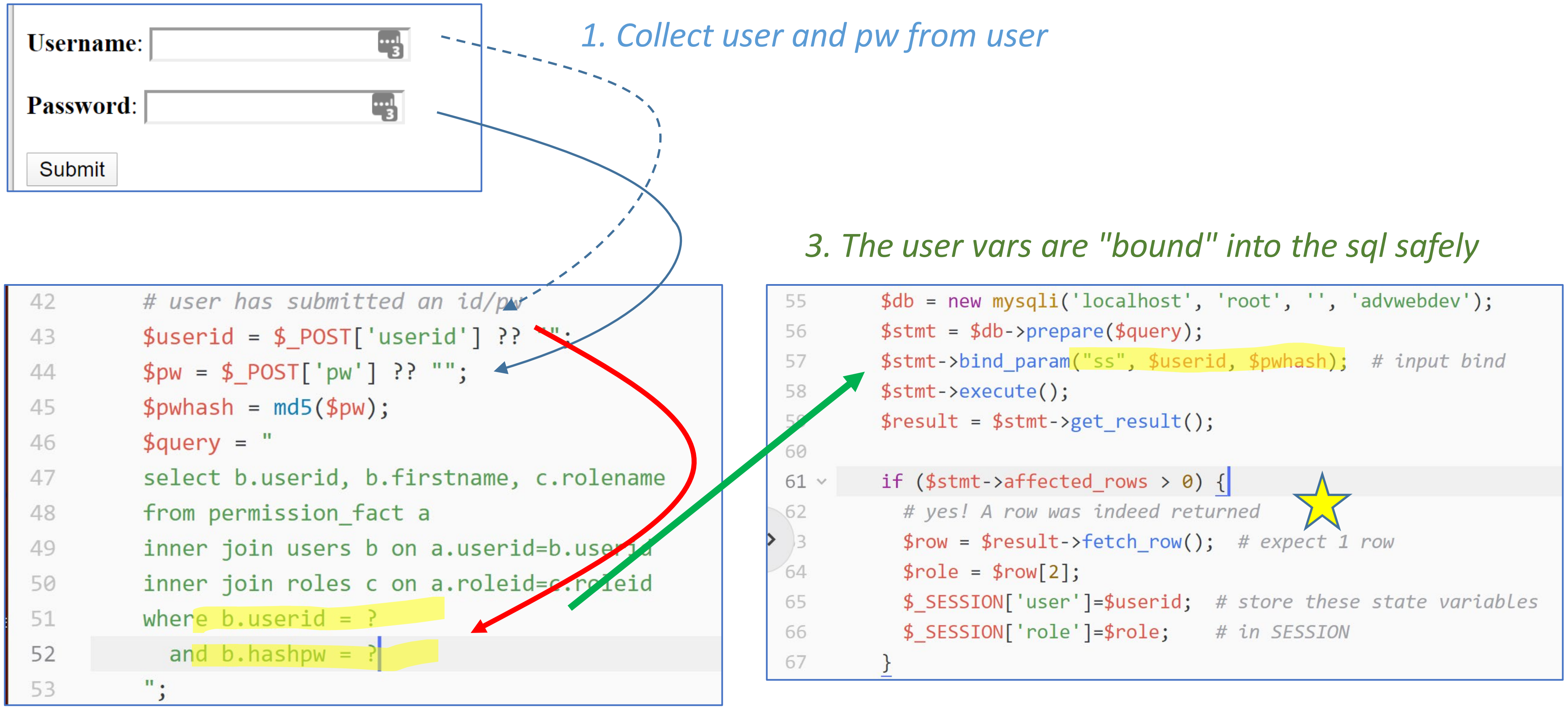
We don't want plaintext pws in a db!!!



```
MariaDB [advwebdev]> select * from users;
+--------+-----------+------------+-----------+----------------------------------+
| userid | firstname | lastname   | password  | hashpw                           |
+--------+-----------+------------+-----------+----------------------------------+
| bobby  | Bob       | Smith      | foobar123 | ae2d699aca20886f6bed96a0425c6168 |
| mark   | Mark      | Tums       | sunshine2 | dd0717328bcab3c05b27460b6918379c |
| prof   | Steve     | Ruegsegger | banana4   | f70b5a5a73ab5afcee5e88c2690db3b1 |
| susie  | Susie     | Jones      | turnip5   | 2c8eff6e8e1708e238111153ba6953d8 |
+--------+-----------+------------+-----------+----------------------------------+
```

Code example 2: simple authentication

- Here is simple sql authentication, without salt, using md5 only.


Username:
Password:
Submit
1. Collect user and pw from user
3. The user vars are "bound" into the sql safely

```
        # user has submitted an id/pw
        $userid = $_POST['userid'] ?? "";
        $pw = $_POST['pw'] ?? "";
        $pwhash = md5($pw);
        $query = "
        select b.userid, b.firstname, c.rolename
        from permission_fact a
        inner join users b on a.userid=b.userid
        inner join roles c on a.roleid=c.roleid
        where b.userid = ?
          and b.hashpw = ?
        ";
```

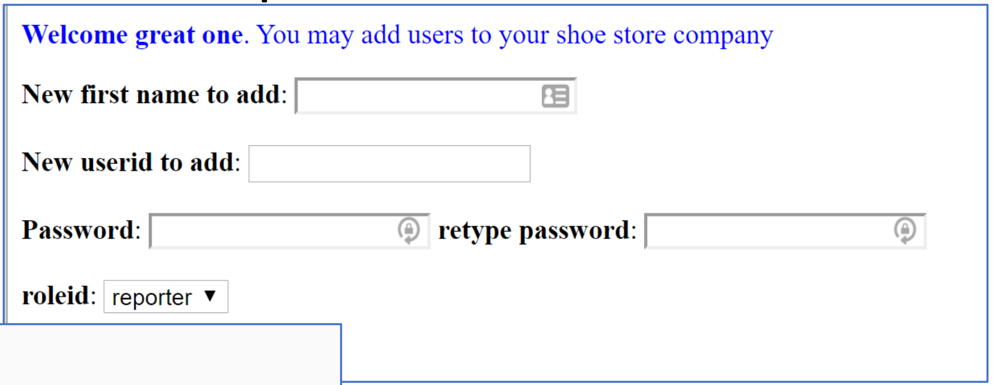
```
        $db = new mysqli('localhost', 'root', '', 'advwebdev');
        $stmt = $db->prepare($query);
        $stmt->bind_param("ss", $userid, $pwhash);  # input bind
        $stmt->execute();
        $result = $stmt->get_result();

        if ($stmt->affected_rows > 0) {
          # yes! A row was indeed returned
          $row = $result->fetch_row();  # expect 1 row
          $role = $row[2];
          $_SESSION['user']=$userid;  # store these state variables
          $_SESSION['role']=$role;    # in SESSION
        }
```

2. The sql is prepped for them

l12_shoe_shop_4_authentication_setup

Code example 3: PHP authentication

- Here is the PHP-way to add users & pw



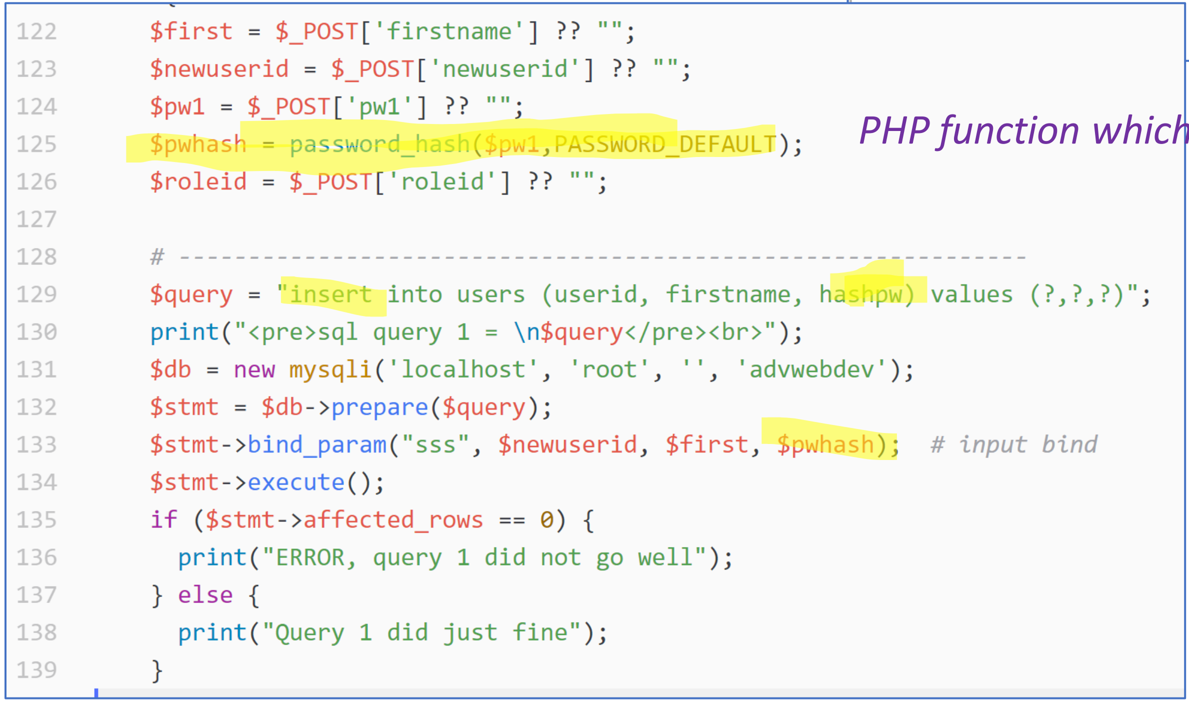
Welcome great one. You may add users to your shoe store company

New first name to add:

New userid to add:

Password: **retype password**:

roleid: reporter ▼



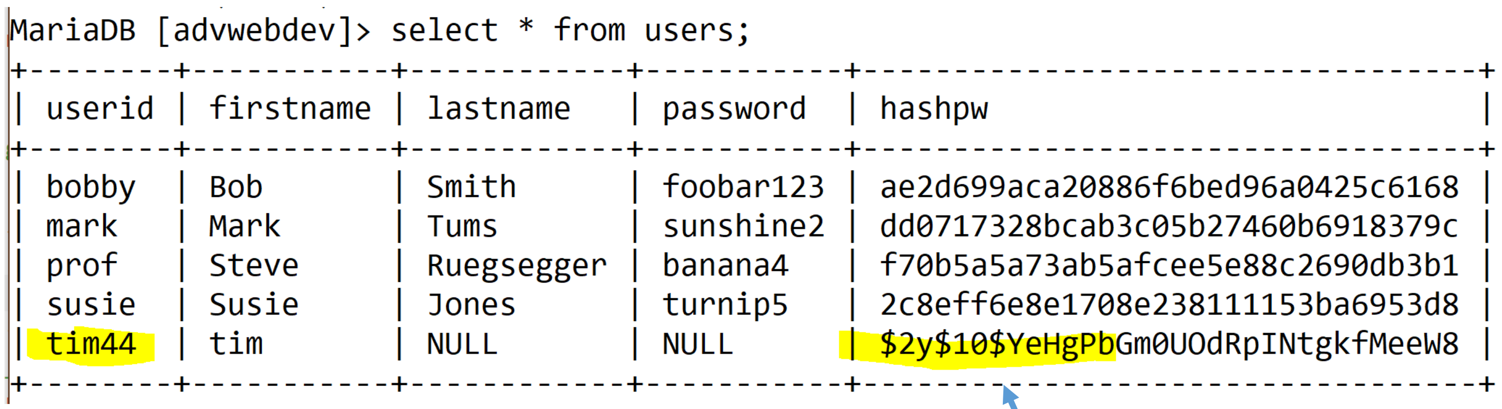
```php
    $first = $_POST['firstname'] ?? "";
    $newuserid = $_POST['newuserid'] ?? "";
    $pw1 = $_POST['pw1'] ?? "";
    $pwhash = password_hash($pw1,PASSWORD_DEFAULT);
    $roleid = $_POST['roleid'] ?? "";

    # ------------------------------------------------------------
    $query = "insert into users (userid, firstname, hashpw) values (?,?,?)";
    print("<pre>sql query 1 = \n$query</pre><br>");
    $db = new mysqli('localhost', 'root', '', 'advwebdev');
    $stmt = $db->prepare($query);
    $stmt->bind_param("sss", $newuserid, $first, $pwhash);  # input bind
    $stmt->execute();
    if ($stmt->affected_rows == 0) {
      print("ERROR, query 1 did not go well");
    } else {
      print("Query 1 did just fine");
    }
```

PHP function which manages the salt for us

No plaintext pw

Code example 3: PHP authentication

- Check the sql…



```
MariaDB [advwebdev]> select * from users;
+--------+-----------+------------+-----------+----------------------------------+
| userid | firstname | lastname   | password  | hashpw                           |
+--------+-----------+------------+-----------+----------------------------------+
| bobby  | Bob       | Smith      | foobar123 | ae2d699aca20886f6bed96a0425c6168 |
| mark   | Mark      | Tums       | sunshine2 | dd0717328bcab3c05b27460b6918379c |
| prof   | Steve     | Ruegsegger | banana4   | f70b5a5a73ab5afcee5e88c2690db3b1 |
| susie  | Susie     | Jones      | turnip5   | 2c8eff6e8e1708e238111153ba6953d8 |
| tim44  | tim       | NULL       | NULL      | $2y$10$YeHgPbGm0UOdRpINtgkfMeeW8 |
+--------+-----------+------------+-----------+----------------------------------+
```

Salt is part of the hash