Cookies, Sessions, and Authentication

COMP-3077 Web based Data Management

Dr. Saja Al-Mamoori

Placeholders

```
PREPARE statement FROM "INSERT INTO classics VALUES(?,?,?,?,?)";
SET @author   = "Emily Brontë",
    @title    = "Wuthering Heights",
    @category = "Classic Fiction",
    @year     = "1847",
    @isbn     = "9780553212587";
EXECUTE statement USING @author,@title,@category,@year,@isbn;
DEALLOCATE PREPARE statement;
```

Create Foreign Key

- Foreign key is used to create the **relationships** between tables.
- Foreign key refer to **a primary key** (aka **referenced key**) in different table.
- That means any value in the foreign key must **exist** in the referenced key.
- The foreign key must have an index (or being part of the primary key in it's own table).
- Employee_Education table refers to both employee and education tables to implement the Many to Many relationship.

Create primary key for Employee_Education

ALTER table employee_education ADD PRIMARY KEY (employee_id,education_id);

- Once employee_id is part of the **primary key, it has an index on it**, if it is not part of the primary key, it needs to have (or be part of) an index.
- Same with education_id

Create foreign keys

- To create the primary key relation between foreign key and referenced key:

```
ALTER TABLE employee_education
ADD FOREIGN KEY(employee_id)
REFERENCES employee(id);
```

- And:

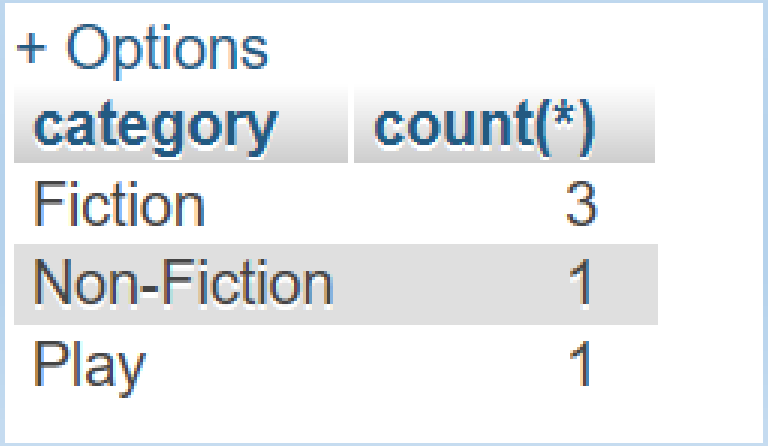
```
ALTER TABLE employee_education
ADD FOREIGN KEY(education_id)
REFERENCES education(id);
```

Reports

- Generate a report from database based on a query

```
SELECT category,count (*)
FROM `classics`
group by category;
```

- It returns the count (number of rows in classics) for each category



+ Options

category	count(*)
Fiction	3
Non-Fiction	1
Play	1

Objective

- As your web projects grow larger and more complicated, you will find an increasing need to keep track of your users.
- User preferences and settings.
- Even if you aren't offering logins and passwords, you will still often need to store details about a user's current session and possibly also recognize her when she returns to your site.

Using Cookies in PHP

- ***A cookie:*** is an item of data that a web server saves to your computer's hard disk via a web browser.
- It can contain almost any alphanumeric information (as long as it's under 4 KB)
- It can **be retrieved from your** computer **and returned to the server.**
- ***Usage:***
- session tracking
- maintaining data across multiple visits
- holding shopping cart contents
- Storing login detials

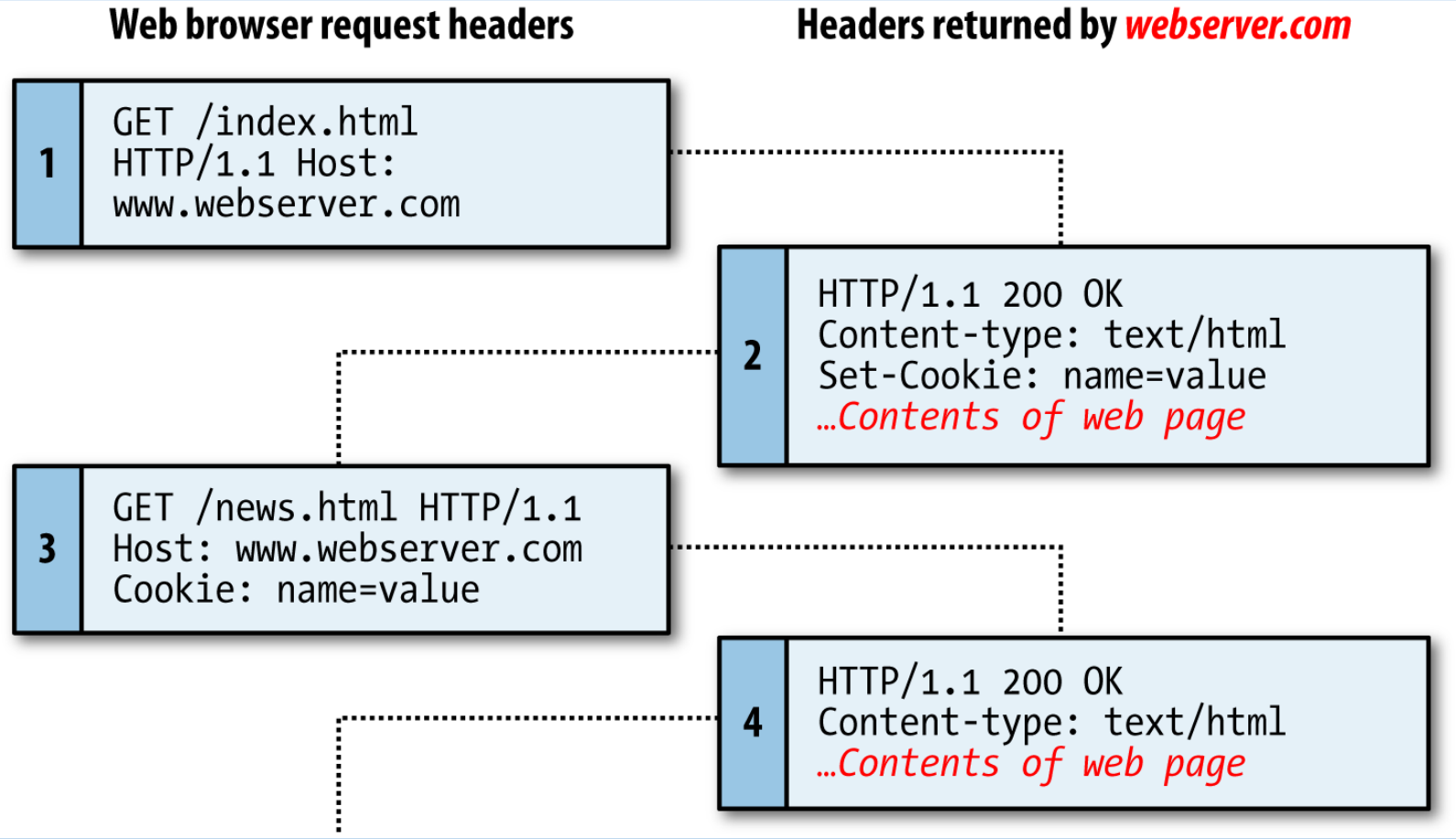
Privacy Implications

- Cookies can be read only from the **issuing domain.**
- In other words, if a cookie is issued by, for example, oreilly.com, it can be retrieved only by **a web server using that domain**.
- This prevents other websites from gaining access to details for which they are not authorized.
- multiple elements on a web page can be embedded from multiple domains, each of which can issue its own cookies. **Aka third-party cookies (TPC).**
- Most commonly, (TPC) are created by advertising companies in order to track users across multiple websites.

How Browsers treats Cookies

- browsers allow users to turn cookies off either for the current server's domain, TPC, or both.
- Fortunately, most people who disable cookies do so only for third-party websites.
- Cookies are exchanged during the transfer of headers, **before the actual HTML of a web page is sent**.
- It is impossible to send a cookie once any HTML has been transferred.
- Therefore, careful planning of cookie usage is important.

Exchanging Cookies between Browser and Server



A browser/server request/response dialog with cookies

1. The browser issues a request to retrieve the main page, index.html, at the website http://www.webserver.com. The first header specifies the file, and the second header specifies the server.

2. When the web server at webserver.com receives this pair of headers, it returns some of its own. The second header defines the type of content to be sent (text/html), and the third one sends a cookie of the name ***name*** and with the value ***value***. Only then are the contents of the web page transferred.

3. Once the browser has received the cookie, it will then return it with every future request made to the issuing server until the cookie expires or is deleted. So, when the browser requests the new page /news.html, it also returns the cookie name with the value ***value***.

4. Because the cookie has already been set, when the server receives the request to send /news.html, it does not have to resend the cookie, but just returns the requested page.

Setting a Cookie

- setcookie(name, value, expire, path, domain, secure, httponly);

Parameter	Description	Example
name	The name of the cookie. This is the name that your server will use to access the cookie on subsequent browser requests.	username
value	The value of the cookie, or the cookie's contents. This can contain up to 4 KB of alphanumeric text.	Hannah
expire	(Optional.) Unix timestamp of the expiration date. Generally, you will probably use time() plus a number of seconds. If not set, the cookie expires when the browser closes.	time() + 2592000
path	(Optional.) The path of the cookie on the server. If this **is a / (forward slash), the cookie is available over the entire domain,** such as www.webserver.com. If it is a subdirectory, the cookie is available only within that subdirectory. The default is the current directory that the cookie is being set in, and this is the setting you will normally use.	/
domain	(Optional.) The Internet domain of the cookie. If this is .webserver.com, the cookie is available to all of webserver.com and its subdomains, such as www.webserver.com and images.webserver.com. If it is images.webserver.com, the cookie is available only to images.webserver.com and its subdomains such as sub.images.webserver.com, but not, say, to www.webserver.com.	.webserver.com
secure	(Optional.) Whether the cookie must use a secure connection (https://). If this value is TRUE, the cookie can be transferred only across a secure connection. The default is FALSE.	FALSE
httponly	(Optional; implemented since PHP version 5.2.0.) Whether the cookie must use the HTTP protocol. If this value is TRUE, scripting languages such as JavaScript cannot access the cookie. (Not supported in all browsers.) The default is FALSE.	FALSE

Setting a Cookie

- Example:

```
setcookie('username', 'Hannah', time() + 60 * 60 * 24 * 7, '/');
```

- This will create a cookie with the name username and the value Hannah that is accessible across the entire web server on the current domain, and will be removed from the browser's cache in seven days

Accessing a Cookie

- Reading the value of a cookie is as simple as accessing the $_COOKIE system array.

```
if (isset($_COOKIE['username']))
    $username = $_COOKIE['username'];
```

- Note that you can read a cookie back only after it has been sent to a web browser.
- When you issue a cookie**, you cannot read it in again until the browser reloads the page** (or another with access to the cookie) from your website and passes the cookie back to the server in the process.

Destroying a Cookie

- To delete a cookie, you must **issue it again and set a date in the past**.
- It is important for all parameters in your new setcookie call except the timestamp to be identical to the parameters when the cookie was first issued;
- otherwise, the deletion will fail.

```
setcookie('username', 'Hannah', time() - 2592000, '/');
```

Storing Usernames and Passwords

- Obviously, MySQL is the natural way to store usernames and passwords.
- But again, we don’t want to store the passwords as clear text, because our website could be compromised if the database were accessed by a hacker.
- Instead, we’ll use a neat trick called a **one-way function**.
- using the **hash function**, you can keep up with future developments in security and simply pass the hashing algorithm to it that you wish to implement, resulting in less code maintenance

Storing Usernames and Passwords

- PHP hash function, passing it a version of the **ripemd** algorithm, which was designed by the open academic community and which returns a 32-character hexadecimal number

```
$token = hash('ripemd128', 'mypassword');
```

- That example happens to give $token this value:

```
7b694600c8a2a2b0897c719958713619
```

Salting

- Unfortunately, hash on its own is not enough to protect a database of passwords, because it could still be susceptible to a brute force attack that uses another database of known 32-character hexadecimal tokens.
- Salting is simply a matter of adding some text that only we know about to each parameter to be encrypted (hashed).
- it can be before (and/or) after the password

```
$token = hash('ripemd128', 'saltstringmypassword');
```

- The text saltstring has been prepended to the password. Of course, the more obscure you can make the salt, the better. I like to use salts such as this:

```
$token = hash('ripemd128', 'hqb%$tmypasswordcg*l');
```

Using Sessions

- **HTTP address doesn't maintain state**.
- Scenario: the user surfs a website pages, move from page to another, close the bowser. The server does not know who is this user.
- Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.
- So; Session variables hold information about one single user, and are available to all pages in one application.
- Unlike cookies, sessions are stored in server-side.

Starting a Session

- $_SESSION['variable'] = $value;
- They can then be read back just as easily in later program runs, like this:
- $variable = $_SESSION['variable'];

- https://www.w3schools.com/php/showphp.asp?filename=demo_session1

- https://www.w3schools.com/php/showphp.asp?filename=demo_session3

Ending a Session

```
// remove all session variables
session_unset();

// destroy the session
session_destroy();
```

https://www.w3schools.com/php/showphp.asp?filename=demo_session5

Session Time-Outs

- If a user logs in to your site in an Internet café and then leaves the computer and café without logging out.
- The next user sit on that computer can see the private transactions of the first user.
- You may set a timer for the session to expire after certain time (10 mins).

Session Time-Outs

```
<?php  //page.php
session_start();
$inactive = 600; // set time-out period (in seconds)
if (isset($_SESSION["timeout"])) {// check to see if $_SESSION["timeout"] is set
// calculate the session's "time to live"
$sessionTTL = time() - $_SESSION["timeout"];
    if ($sessionTTL > $inactive) {
        session_destroy();
        header("Location: /logout.php");    } }
$_SESSION["timeout"] = time();
?>
```

Header(“Location:/somepage.php”) will redirect the page to somepage.php

References

- Learning PHP, MySQL & JavaScript, 4th Edition. With jQuery, CSS & HTML5, by Robin Nixon, O'Reilly Media, 2014, 978-1-4919-1866-1, chapter 12