PHP

Datenbanken, Cookies und Sessions

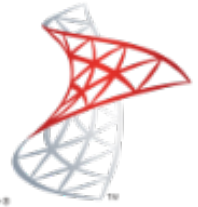
Datenbankzugriff

Datenhaltung mittels Dateien nicht zu empfehlen

- Datenbanken!

Die meisten gängigen Datenbanken lassen sich mit PHP nutzen

- (Oracle) MySQL
- MariaDB
- Oracle Database
- SQLite
- MS SQL
- PostgreSQL
- IBM Informix
- MongoDB (NoSQL)



Wir nutzen SQLite

Weitere Informationen (nicht wirklich…): http://howfuckedismydatabase.com/

mysql_* Funktionen

- Unübersichtliche API, da sehr alt
- Keine OOP
- Prepared Statements nicht möglich

mysqli_* Funktionen/Klassen (MySQL *improved*)

- OOP
- Unterstützung von Prepared Statements
- „Cutting edge"-MySQL-Support

Weitere Informationen: http://php.net/manual/en/mysqli.overview.php

PDO (PHP Data Objects)

- OOP
- Unterstützung von Prepared Statements
- Unterstützung vieler Datenbanktypen
  - > Migration der Datenbank mit wenig Aufwand möglich

Weitere Informationen: http://php.net/manual/de/book.pdo.php

Verbindung aufbauen

```php
$dsn = 'mysql:dbname=testdb;host=127.0.0.1';
$user = 'dbuser';
$password = 'dbpass';

try {
    $dbh = new PDO($dsn, $user, $password);
} catch (PDOException $e) {
    echo 'Connection failed: ' . $e->getMessage();
}
```

SQL-Abfrage mittels PDO::query

```
$sth = $dbh->query("SELECT * FROM `user`
                    WHERE id = '" . $_GET['id'] . "'");

$row = $sth->fetch();
echo $row['name'];
```

- Anfällig für SQL Injections
- Sollte für Abfragen nicht benutzt werden

Alternative: Prepared Statements

Prepared Statements

- Vorbereitete Anweisung wird an die Datenbank gesendet
  - > Platzhalter für Variablen

- Ausführung des Statements nach Einsetzen der Variablen
  - > Datenbanksystem kümmert sich um das Escapen
  - > Verhindert SQL-Injections und prüft die Gültigkeit von Parametern

- Geschwindigkeitsvorteil bei mehrfachen Ausführen
  - > Statement liegt dem Datenbanksystem bereits vor

Weitere Informationen: http://php.net/manual/de/pdo.prepared-statements.php

Zustände
Cookies & Sessions

Zustand des PHP Skriptes geht nach Ausführung verloren

- Variablen sind nur für einen Aufruf gültig

Wie können mehrstufige Operationen/Transaktionen vorgenommen werden?

- Der Server muss erkennen, dass der nächste Aufruf der Seite in einem Kontext geschieht
- ISO/OSI, Schicht 5? Session?
  - > Nein, im HTTP-Protokoll ist das Aufgabe der Applikation!

Persistente PHP Skripte

- Parameter, der den aktuellen „Schritt" speichert
- Cookies & Sessions bieten die Möglichkeit den Inhalt von Variablen länger zu speichern

Zustände
Cookies

Cookies bieten die Möglichkeit clientseitig Daten zu speichern

- Benutzer muss sich nicht mehrfach anmelden
- Aber auch Tracking des Nutzerverhaltens möglich

- Cookies werden im Header der Seite zurückgeliefert
- Das Setzen von Cookies muss daher erfolgen bevor Inhalt ausgegeben wird

- Beispiel:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
Set-Cookie: foo=bar; Expires=Wed, 09 Jun 2021 10:18:14 GMT

(Inhalt der Seite)
```

Zustände
Cookies

Speicherung des Cookies als „Datei“ auf dem Client

- Tatsächlich meist mit Hilfe von Datenbanken realisiert

Der Cookie wird für eine Domäne (DNS) gesetzt (typischerweise der Server, der diesen übermittelt)

Client sendet beim nächsten Aufruf der Webseite den Cookie mit

- Nur die Webseite, die den Cookie gesetzt hat, kann den Cookie lesen

- Beispiel:

```
GET / HTTP/1.1
Host: www.example.org
Cookie: name=value; foo=bar
…
```

Setzen eines Cookies in PHP

```
bool setcookie($name, $value [, $expire [, $path]])
```

> Speichert Cookie mit Namen $name und Wert $value
> $expire ist ein Unix Timestamp (Integer), der angibt wann der Cookie verfällt
> $path gibt an von welchen Verzeichnissen der Cookie gelesen werden kann
> Weitere Parameter möglich (siehe PHP-Doku)

- Beispiel:

```
setcookie('fontSize', '+1', time()+60*60*24*30, '/');
```

> Cookie „fontSize" mit Wert „+1" gesetzt, der in 30 Tagen abläuft und von allen Verzeichnissen in der Domain aus gelesen werden kann

Hinweis: Man kann über den Parameter Domain auch eine Sub-Domäne angeben, für den der Cookie gesetzt wird. Auf dem Rechner login.fh-aachen.de z.B. für fh-aachen.de. Niemals aber für de!

Weitere Informationen: http://php.net/manual/en/function.setcookie.php

Zustände
Cookies

Auslesen eines Cookies über das Cookie-Array:

- Beispiel:

```
$_COOKIE['fontSize'] // liefert '+1' zurück
```

- Kann nicht im selben Skriptdurchlauf gespeichert und wieder gelesen werden
  - > Cookie wird erst beim Senden des Clients im Array eingetragen
  - > Erst der nächste Skriptdurchlauf kann den Cookie lesen

Cookies
Same Origin Policy

Angenommen wir erhalten einen Cookie von http://store.company.com/dir/page.html

Letztlich wollen wir verhindern, dass der Cookie auch an fremde Server übertragen wird. Hierzu gibt es beim Thema Sicherheit die Same-Origin Policy!

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner/another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

Quelle: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

Cookies
Same Origin Policy

Cookies verfolgen ein etwas anderes Verständnis der Origin. Grundsätzlich bleibt die Idee der Same Origin Domain erhalten.

- **Domain:** Ein Rechner kann einen Cookie für seinen eigenen DNS-Namen setzen, aber auch einen, für eine übergeordnete Domäne (nur keine sogenannte Public Domain, z.B. .de)
- Der Cookie wird für alle Zieladressen ausgeliefert, die der gesetzten Domäne entsprechen oder übergeordnet sind (mit Ausnahme der Public Domains). Dies gilt zunächst unabhängig vom Port oder dem Protokoll (http vs https)
- **Path:** Ein Cookie kann an einen Pfad inkl. der Unterpfade gebunden werden. Wenn also ein Cookie auf einen Pfad /sander gebunden ist, dann wird er für den Pfad /roussel nicht ausgeliefert, aber für /sander/test
- **Secure:** Schalter, mit dem erzwungen wird, dass https zu verwenden ist
- **httponly:** Schalter, mit dem erzwungen wird, dass der Cookie keinen im Browser laufenden Scriptsprachen zur Verfügung gestellt wird

Quellen: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy, https://www.php.net/manual/en/function.setcookie.php

Cookies
Same Origin Policy

- Die klassische Same Origin Policy führt die Einschränkungen über
  (Schema, Domain, Port)
  Durch. Nur wenn dieses Triple Übereinstimmt ist der Zugriff erlaubt

- Cookies sind hier generöser und verfahren wir folgt
  (Schema*, Domain, Path)
  *: Schema nur dann, wenn Secure gesetzt ist

  Jedes (Domain, Path)-Tupel ist ein eigener Coolkie
  Es werden alle Cookies übertragen, bei dem die URL eine Subdomäne und ein Subpfad darstellen

Cookies
Geänderte Same Origin Policy

login.site.com setzt zwei verschiedene Cookies

cookie 1	cookie 2
name = **userid**	name = **userid**
value = u1	value = u2
domain = **login.site.com**	domain = **.site.com**
path = /	path = /
secure	non-secure

http://checkout.site.com/	cookie: userid=u2
http://login.site.com/	cookie: userid=u2
https://login.site.com/	**cookie: userid=u1; userid=u2**

Quelle: https://crypto.stanford.edu/cs142/lectures/10-cookie-security.pdf

Zuständige
Sessions

Cookies

- Speichern alle Informationen clientseitig ab
- Setzen der Information muss vor der ersten HTML-Ausgabe geschehen
- Daten können vom Benutzer manipuliert werden
- Nur die Speicherung eines Strings möglich

Sessions

- Speichern die Daten serverseitig ab
- Client erhält nur eine Session-ID (als Cookie)
  - > Um das Setzen des Cookies kümmert sich PHP
  - > Alternativ auch möglich die Session-ID per URL-Rewriting zu übergeben
    - PHP-Option: session.use_cookies
- Daten, die per Session gespeichert werden, werden auch nur auf dem Server gespeichert
  - > Keine Datenmanipulation möglich
  - > Komplexe Variablen (Arrays,Objekte) sind möglich

Weitere Informationen: http://php.net/manual/de/intro.session.php

Benutzung in PHP

```
bool session_start()
```

> Erzeugt eine Session oder nimmt die aktuelle wieder auf
  - Generiert falls nötig Session-ID und setzt den entsprechenden Cookie
> Muss aufgerufen werden, bevor irgend etwas an den Browser geschickt wird

```
bool session_destroy()
```

> Beendet Sitzung und löscht Variablen

Zustände
Sessions

Setzen und Lesen von Session-Werten

- Im `$_SESSION`-Array können Werte gespeichert werden, die seitenübergreifen verfügbar sind

- Beispiel:

```
$_SESSION['logged_in'] = true;
```

- Zugriff analog über das Array

Zustände
Sessions

Beispiel

```php
<?php
    session_start();

    if (!isset($_SESSION['zaehler'])) {
        $_SESSION['zaehler'] = 1;
    } else {
        $_SESSION['zaehler']++;
    }

    echo 'Sie haben diese Seite ' . $_SESSION['zaehler'] .
         ' mal aufgerufen';
?>
```

PHP Output Buffering

Warum sehe ich die Ausgabe „Hello World!“ erst nach 5 Sekunden?

```php
<?php
    echo 'Hello ';
    sleep(5);
    echo 'World!';
?>
```

Und wieso sehe ich die Ausgabe schon früher?

```php
<?php
    $size = 1024 * 8;
    for($i = 1; $i <= $size; $i++) {
        echo '.';
    }
    sleep(5);
    echo 'Hello World';
?>
```

PHP Output Buffering

Output Buffering

- Mechanismus bei dem die Ausgaben nicht direkt zum Browser verschickt werden, sondern vielmehr in „Chunks“ eingeteilt werden
- Die Ausgaben werden also gepuffert und (standardmäßig) in Einheiten von 4KB an den Browser geschickt

- Wenn dieser übertragen wurde (also bei 4K an Daten), dann kann nachher auch kein HTTP-Header mehr gesetzt werden
  > Deshalb immer sicherstellen, dass Funktionen, die header verändern (header, session_start, setcookie) aufgerufen werden, bevor die ersten Daten im HTTP body übermittelt werden

Weitere Informationen: http://php.net/manual/de/book.outcontrol.php

PHP-Frameworks

Der Composer ermöglicht das Wiederverwenden und Teilen einzelner Komponenten

- Evtl. wollen wir ein „Gesamtpaket“ zum Starten
- Ein Framework enthält bereits „Best Practices“ beim Entwickeln und löst viele bereits gelöste Probleme

Die meisten Frameworks sind komponentenbasiert

- Typische Komponenten:
  - > Benutzerverwaltung
  - > Datenbankzugriff
    - – Object Relational Mapping
  - > MVC-Unterstützung
  - > Caching-Systeme
  - > Vereinfachung des Zusammenspiels von JavaScript und PHP

PHP-Frameworks

Zend Framework

- Das wohl bekannteste PHP-Framework
  - > Strikt objektorientiert

Symfony

- Beeinflusst durch andere Web Frameworks (Ruby on Rails, Django)
- Benutzt viele andere Opensource-Frameworks

Laravel

- Recht „neu“, aber bereits weit verbreitet