The degree of normality in a database is inversely proportional to that of its DBA.

unknown

9 Database

9.1 Introduction

As already said: PHP is a stateless scripting language. A PHP script runs for a very short time and if it is finished, it clears its memory.

There are some possibilities to store data client-side (cookies) or server side (sessions), but these are meant for small pieces of data which your application doesn't rely on. Your application will be able to create these sessions and cookies.

So what about login-credentials, or content-items like posts of pages? You can store them in textfiles but this is very hard if you want to read a selection or edit them easily or even sort them.

Databases are a common used solution for the problem of storage of information. PHP comes loaded with a lot of drivers and functions to use when it comes to database connectivity.

9.2 Database abstraction

Instead of starting with a list of database functions, it is worthy to have a look at some common used design principle: make an abstraction.

The reason for this, is that your application shouldn't bother about what database management system (MySQL, postgreSQL, SQL-server, ...) is behind the server. By making use of an abstraction layer, your application keeps working, even if you port it to another server, with another DBMS.

Using a full database abstraction layer is beyond the scope of this course. But it is encouraged to use PDO, PHP 's own database-access abstraction layer. Have a look at `www.php.net/manual/en/book.pdo.php` for a complete overview. It doesn't rewrite SQL or emulate missing features, but still does a pretty good job in connection, executing query's and fetching results. So the only thing you need to take care of, is the SQL itself.

9.3 Connecting to the database

For this course, we will use a postgreSQL database.

For every student there should be an extra `.db_password.php` file in de root folder of your ftp dir. (It is in the folder above your webroot folder.) This file contains credentials to connect to your database.

Have a look at `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex9.1.phps`. The connection details to connect to the postgresql database are all documented in the PHP source code. `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex9.1.php` gives you a live example.

9.4 Selecting data from the database

Once you have your connection, you can reuse it all the time.

Listing 9.1 using $db to select some data

```
<?php

// query
$sql = "SELECT * FROM products";

// result is the object after executing the select query
$result = $db->query($sql);

// fetchAll will collect all rows from the object
// and stores them into an associative array (parameter PDO::FETCH_ASSOC)
$data = $result->fetchAll(PDO::FETCH_ASSOC);

// debug print of your two dimensional, associative array
print_r($data);

?>
```

Have a look at `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex9.1.phps` for more details.

9.5 Modifying data

All modifications happen through the `exec()` method.

Listing 9.2 using $db to select some data

```
<?php

// query
$sql = "
        INSERT INTO
                products_123456 (name, price, quantity)
        VALUES ('some rare things', 3.3, 10);
";

// result is the number of affected rows by the sql command
$result = $db->exec($sql);

// debug print of the number of affected rows
echo $result;

// query
$sql = "
        UPDATE  products_123456
        SET quantity = 13
        WHERE name = 'some rare things';
";

// result is the number of affected rows by the sql command
$result = $db->exec($sql);

// debug print of the number of affected rows
echo $result;

?>
```

If you want to have your own script, you need your own database.

Don't forget to give the correct grants to your local user! (The script should do this for you, but if you make your own tables ...)

Try to log everything: an error message is the first step in solving a problem.

Exercise 9.1 - connecting to a database and doing some modifications

- Change the port number to 51213.
- Use pgAdmin to connect to the database of your class (e.g. 2TX34)
- Copy the code from `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex9.1.phps`. Make sure you have got the include file as well. Change the database name and try to connect to your database. There should be a table products in every database.
- Now go to `https://dynweb.webontwerp.khleuven.be/dynweb/examples/ex9.2.1.php`, get the DDL code generated for your studentnr. Connect via pgAdmin to your class-database and execute the code to create your own schema and table.
- Change/create your script so that it connects to your own database schema and table.

9.6 Preventing SQL injection

Have a look at this example (or google for 'sql injection'):

Listing 9.3 database transactions vulnerable to sql injection

```
<?php

// query
$sql = "
        SELECT password
        FROM users
        WHERE username = '" . $_GET['username'] . "';
";

// result is the first column of the first row
$result = $db->query($sql);
$value = $result->fetchColumn();

if ($value == sha1($_GET['password'])) { // password hashes, remember?
        echo 'your are logged in';
}

?>
```

It looks fine, but what happens when a user enters `' OR 1=1-`? The executed query will be something like: `SELECT password FROM users WHERE username = '' OR 1=1--` which means: every user. The first row will perhaps be that of an administrator.

And this is just the beginning ;-)

There are sollutions, PDO offers you one: statement preparing. The principle is simple: you use placeholders for the values in your sql-command.

- Use a `:` followed by a choosen key for each variable.
- Use this in your sql-statement.
- Run the sql-statement through the `prepare()` method
- bind every sql-parameter to a PHP variable using the `bindParam()` method. In this method you can specify the expected datatype.
- Execute your statement
- Fetch whatever you want from the statement

Listing 9.4 pdo and prepared statements to prevent sql-injection

```php
<?php

// query with a sql-parameter
$sql = "
        SELECT password
        FROM users
        WHERE username = :user_name;
";

// a database statement with the prepared statement
$stmt = $db->prepare($sql);

// binding a php variable to the sql-parameter, it should be a string
// with maximum of 15 characters
$stmt->bindParam(':user_name', $_GET['username'], PDO::PARAM_STR, 15);

// execute the statement
$stmt->execute();

$value = $stmt->fetchColumn();

if ($value == sha1($_GET['password'])) {
        echo 'your are logged in';
}

?>
```

If you apply this to all parameters in any query (SELECT, INSERT, UPDATE, DELETE, ...), this should prevent most SQL-injection.

Exercise 9.2 - writing some queries

- Display how many items of the product Dana Winner CD there are in stock (off course by writing a query and getting this information from the DB). Have a look at a working version at `https://dynweb.webontwerp.khleuven.be/exercises/9.2.1.php`.
- Make a form where you can give a product name and which returns the price and the number of items still in stock of this product. If the product those not exist, you give an error to the user. Have a look at a working version at `https://dynweb.webontwerp.khleuven.be/exercises/9.2.2.php`.
  Make sure you use prepared statements.
- Make a form where you can update the stock for a particular product. If the product those not exist, you give an error to the user.
  Have a look at a working version at `https://dynweb.webontwerp.khleuven.be/exercises/9.2.3.php`.
  Make sure you use prepared statements.