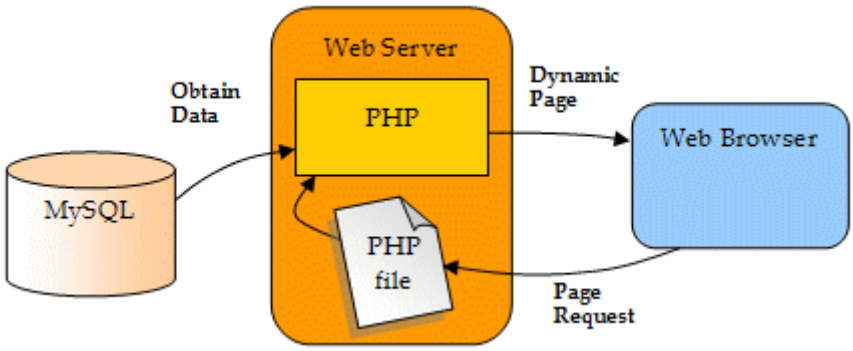
Simon Lyngbo Nielsen
Business Academy Aarhus – Front-Line

.PHP, .SQL & DATABASE EXAM



Github:

https://github.com/slyngbo/simon_lyngbo_superheroes

Username: Batman m.fl

Password: none

Characters with spaces:

12,544

Supervisor: jarne w. beutnagel

Business Academy Aarhus

Front-end

27/04-2018

INDHOLD

INTRODUCTION

On the internet there are several ways to store data. One of the is in a database, where we can structure and organize our data. The database covered in this paper will be a relational databased based on .SQL.

On the internet we use databases for storing everything from images to e-mails and to the money on your bank account. The purpose of a database is to store data until it is needed. A database can obtain data in several ways. We can create the data, it can collect it online or it can even create the data itself.

With this paper, I will go through the entire process of designing, implementing and deploying the database, and showcase some of the awesome stuff, that just a simple database can accomplish.

CASE (SUPERHERO DATING PORTAL)

For this project I am developing an online meeting platform for those busy worlds saving superheroes. Since this website is purely fictional, and does not already exist mean, that I must develop everything from scratch. That involves being able to sign up to the site, find, like and comment on other user profiles, and edit your own data. All relevant data must be public.

STEP: 1 DATABASE DESIGN

When starting to develop a database driven site from scratch, you always want to start with specifying what functions you want, what data they need, and what data should have access to what data. Since I don't have access to any real superheroes, I will base the website functionality on other dating sites like daing.dk and scor.dk, to see what functions they have.

Based on those functions I in this case, have to create a database that can showcase create and update a profile, see and like and or comment on other users profiles. All this data will be put into tables, that can store the data to be called upon. All these functions must be public, since being a superhero is basically a popularity contest.

With all of this in mind I will start to draw a map of the relations between the different tables of my database. To do this I will use the Entity-Relationship Model. The E.R model contains 3 different kinds of 3 parts entities, attributes and relations.

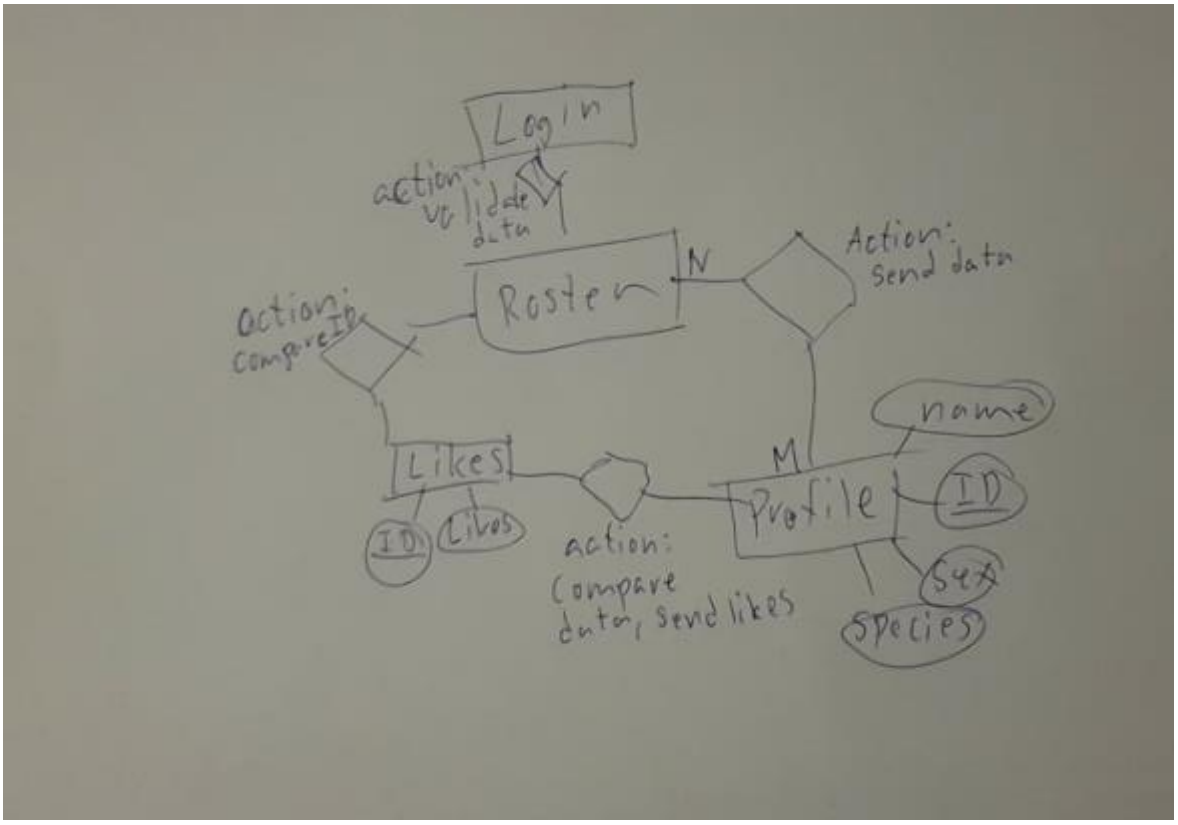
Entities: An entity is your real-world interest. An example of an entity, could be a hero profile. Entities are usually the base table where you enter data. Entities usually consist of 1 or more attributes. Most of the time they include an ID to define the primary key.

Attributes: Attributes are our main set of data. They are attributes of our entities hence the name. And they go in the table of content, that the entity creates in our database. It could be an ID, an e-mail, a link to an url-address, or anything else you can think of. The first attribute should always, for ease of use, be the primary key.

Relations: Relations are the status between two or more entities. Most of the time they include a function. Be that receive/ sending data, etc.

My E.R. Model:

The E.R. model I made for my superheroes dating site contains of the following:



Where I keep roster as my main database to insert heroes in to using different functions.

Profile will display roster and likes to the user if ID is correct. Where roster is also used to validate what is entered during login. I know now, that all the attributes on profile should have been under roster.

I later realized that I needed a specific table for comments. It is used to display comments on the profile. And would be between roster and Profile in a similar way to how likes interacts with them.

Primary Keys: A primary key is a column in a table, that in a unique way, identifies all the records of said table. Meaning that for each column there is a unique identifier. In my Case I will implement an ID that Auto_increment, meaning that for each row added to the table, it will increase the number in the ID column by exactly 1.

Foreign keys: Foreign key, are used to link two tables together, so they can share data. This enables us to join them together. This is very useful if you have several tables containing the same data, and you want to join them, to see what data is different or the same, and get an output out of this.

STEP: 2 DATABASE IMPLEMENTATION

With the base design of my database in place, I can now focus on creating the tables themselves.
When creating the tables we I can use 3 different kinds of .SQL code, DDL, DML and DCL.

DDL is used to create and mody data in the table using CREATE, DROP and similar commands.
DML is used to manipulate the data in our table. It can be done by using Update, Delete, SELECT etc.

commands.
DCL is used to control the privileges between the tables. Examples of commands are: Grant and revoke. I will however be doing this manually using phpMyAdmin to control the MYSQL database, and .sql code to utilize my actions with .PHP code.

To make sure I didn't corrupt the data in my tables, made sure, that everything only containing numbers, were set as an integer, so the users cannot enter characters, giving it a value high enough, not to be reached for a substantial amount of time. Everything else I made in to a varchar, also of appropriate length, meaning that people can enter any key that are legal to the utf-8 standard.

CODE

```php
// First, prepare the SQL
$sql = "INSERT INTO comments (
            Reciever,
            Message,
            Sender
          )
    VALUES (?,?,?)";
// Secondly, add values
$values = array(
  $_POST['Reciever'],
  $_POST['Message'],
  $_SESSION['ID']
);
```

IN this code snippet we see, how I start .SQL, and uses the DML command INSERT INTO, to modify the data in the table. In this case I didn't have any data before the user fills out, text in the related form and submits.

```
38      <label for="Sender">Your name</label>
39      <input type="text" name="Sender" >
40
41      <input type="submit" name="submit" value="Send message">
```

STEP: 3 FRONT-END DEVELOPMENT

I am using the by our teacher provided stylesheet to arrange everything in to articles, that he already design, since design is not part of our evaluation, I see no need to waste time on pretty shapes and colors.

This meant that I could focus on making my .html ready for the data I wanted to import from my database. I tried to use semantic tags wherever possible, to give the site a good ranking on search engines, so our heroes can find the site.

.HTML

```
  <h1>Who do you want to meet today?</h1>
<a href="profile.php?id=<?php echo $_SESSION['ID'];?>">
  <a href="logout.php" >Logout</a>
  <p class=" notice" style="color: red;">Logged in as <?php echo $_SESSION['Name']; ?></p>
</a>
```

This is a good example on, how I tried to stick to semantic .html tags, showcasing the use om h1, a and p, to print on the users screen, that they are logged in as 'Name' printed in red, and also gives them the ability to click their name and go to their profile, while the logout link ends their session and forwards them to the login page.

```
<form method="post">
<input name="Name" placeholder="User name" >
<input name="Password" placeholder="Password" type="password" >
<input type="submit"  name="submit" value="Login">
</form>
```

Here I show how I created a form with 3 input areas, so the user can enter data and the click submit to get forwarded to the main site.

.PHP AND .SQL

.PHP is a server side language, I will not make further comments on the language, but only showcase how I utilized it to make my site come to life.

In the .html form for adding and updating heroes, we find this nifty use of .php code.

```
  // Select all types
  $sql = "SELECT * FROM roster WHERE ID = " . $_SESSION['ID'];
  $roster = $database->query($sql);
?>
```

What it does is damn awesome. It SELECTS *(all) FROM the table "roster" WHERE the "ID" matches the current session "ID". The second line of code starts an SQL query and makes a variable($) named roster, so it is easier for me to know, where it is applied.

Next I put my new variable to good use.
What the code does, is that it pushes data to query names $roster.

```
 echo $roster[0]['Name']; ?>">


php echo $roster[0]['Species'];?>">


"<?php echo $roster[0]['Hair_color'];?>">


<?php echo $roster[0]['Eye_color'];?>">


"<?php echo $roster[0]['Body_shape'];?>">

abel>
ue="<?php echo $roster[0]['Special_power'];?>">

r</label>
alue="<?php echo $roster[0]['Signature_color'];?>">
```

All of this happens when we push the submit button in the following picture.

```
  <label for="Image">Image</label>
  <input type="text" name="Image" value="<?php echo $roster[0]['Image'];?>">

  <input type="submit" name="submit" value="Add">
</form>
```

What happens next is, that the $roster is reverted to the $sql variable, which then runs the following code.

```php
// First, prepare the SQL
$sql = "INSERT INTO roster (
            Name,
            Sex,
            Species,
            Hair_color,
            Eye_color,
            Body_shape,
            Special_power,
            Signature_color,
            Image
          )
    VALUES (?,?,?,?,?,?,?,?,?)";
// Secondly, add values
$values = array(
  $_POST['Name'],
  $_POST['Sex'],
  $_POST['Species'],
  $_POST['Hair_color'],
  $_POST['Eye_color'],
  $_POST['Body_shape'],
  $_POST['Special_power'],
  $_POST['Signature_color'],
  $_POST['Image']
);

// Call prepared function to execute the above
$database->prepared($sql,$values);
```

In this picture we see how the $sql variable is executed, and uses -SQL code to INSERT INTO our table "roster", while it checks, that is as many VALUES as columns, so we don't suddenly end of with data being submitted to the wrong column and creating corrupt data in our database.
Lastly it uses the $_POST command to post everything the $values = array() to activate our execute command, which is seen at the bottom of the snip.

The next cool feature I want to showcase is the .PHP function "foreach ()".
It allows the browser too loop through the data we get from out table. I used this on my index page, to give an overview of all the current profiles.

But first we need to do some .SQL magic and join to tables, so our heroes can show off how many people likes them.

```php
// Get all from tables roster and likes
$superheroes = $database->query('SELECT *
  FROM roster JOIN likes
  ON roster.ID = likes.ID');
```

We see here how, it is SELECT'ing all FROM roster, and JOINs it ON roster.ID = likes.ID, and then creates the variable $superheroes for later use.
What it does is, that it compares the two tables, and since I keep ID as the primary key on all my tables, I can keep on what data I push to the database, and make sure, that it ends up in the right place, so if a user gives a like to ID=1 on the likes table, then ID=1 on the likes table, is always equal to the same ID=1 on the roster table.

```php
// Loop through all titles
foreach ($superheroes as $superheroes) {
  ?>

  <article>
    <a href="profile.php?id=<?php echo $superheroes['ID'];?>">
    <h3><?php echo $superheroes['Name'];?>  <img src="<?php echo $superheroes['Sex'];?>"  class="gender"> </h3>
    <p>
      <img src="<?php echo $superheroes['Image'];?>">
    </p>
    <p>Species: <?php echo $superheroes['Species'];?></p>
    <p>Special power: <?php echo $superheroes['Special_power'];?></p>
    <p>Hair color: <?php echo $superheroes['Hair_color'];?></p>
    <p>Eye color: <?php echo $superheroes['Eye_color'];?></p>
    <p>Body shape: <?php echo $superheroes['Body_shape'];?></p>
    <p>Special power: <?php echo $superheroes['Special_power'];?></p>
    <p>Signature color: <?php echo $superheroes['Signature_color'];?></p>
    <p>Likes: <?php echo $superheroes['Likes'];?></p>
    </a>
```

After the two tables have been joined together, we just the aforementioned function foreach(), to loop through the two tables we JOINed. The reason we want something to loop is, that since our tables are based on a unique “ID”, then we know that the value changes every time it runs. We can then use that knowledge to be sure, that we find all profiles in the table. We then use the .PHP command echo to print them to the user.

Worth noticing is the first line after the <article> tag, it allows us click on the whole article, and show us everything that is stored in our whole database on a single page. This mean, that we don’t have to create a complicated system of files, which would be necessary if we created the site using only .html and .css. And this is where the power of mysql databases really come in to play.

```php
  //Like button
  // Select all types
  $sql = "SELECT * FROM likes";
  $likes = $database->query($sql);
?>

  <form action="likes.php" method="post">
    <input type="submit" name="Likes" value="Like">
    <input type="hidden" name="ID" value="<?php echo $superheroes['ID']; ?>" />
    <input type="hidden" name="redirect" value="index.php" >
  </form>
```

Here we how pushing the likes button sends its data back to the likes table. What is different here form the previous example is, that here we the action committing is hidden, and so is the redirect value.

What it executes when you click the button is the following:

```
14    include('classes/database_likes.php');
15    $database = new Database;
16    $database->connect();
17
18
19    // First, prepare the SQL
20
21  //var_dump($_POST);
22    $SQL = "UPDATE likes SET Likes = Likes+1 WHERE ID=" . $_POST['ID'];
23  $database->query($SQL);
24  header("Location: " .$_POST['redirect']);
```

We see above, how it uses .SQL to UPDATE the table “likes” and SET a new value Likes = Likes+1(The value in the column is set to be an integer, so it takes the current value and sets it = current value + 1) and the it pushes it using WHERE to “ID” and crosschecks if that corresponds to the profile the like button should be positioned on. The final line redirects the user back to the same location after the data has been pushed to the server. The only drawback to this is, that you can like as many times as you wish.

STEP: 4 LOGIN SYSTEMS (FOR BONUS NERD POINTS)

To make everything really cool, I wanted to create a simple login system. And after creating it, and asking a friend about what else it could be implemented for, I started utilizing it elsewhere on the site, and found, that this way of development can cause both immense joy, and great depression.
Because it does really make the code way more complex, but also creates some shortcuts like I will now demonstrate.

```
<form method="post">
<input name="Name" placeholder="Na Na Na Batman" >
<input name="Password" placeholder="Guess if you need me" type="password" >
<input type="submit"  name="submit" value="Login">
</form>
```

To create the login, I used the code above to set whats displayed to the user.
Since I´m not using a Password, all you need to know Is a valid username(hint in the placeholder).

First we need to start a new session using the follow ing tag.

```
session_start();
```

Next we have the login code itself. For this I started by once again calling the "ID" from the roster table. And matching what is typed in, to see if the values match. If they don't, they will be meat with the error message in the next picture. Also the session is stopped, if they do not match, so the user can try again.

```php
if(isset($_POST['submit'])) {

  include('classes/database.php');
  $database = new Database;
  $database->connect();

  $result = $database->query("SELECT * FROM roster WHERE Name LIKE '" . $_POST['Name'] . "'");

  if(empty($result))
  {
    echo "<p>Brugernavn/password forkert!";
    // remove all session variables
    session_unset();

    // destroy the session
    session_destroy();
   } else {
```

If everything checks out, the user will be redirected to the index page. It looks like the following:

```php
      } else {
        //Allow login
        $_SESSION['ID'] =  $result[0]['ID'];
        $_SESSION['Name'] =  $result[0]['Name'];

     header('Location: index.php');
     exit;
     }
   }

?>
```

STEP: 6 SECURITY

My sites many security risk involves my .html <input>, since I didn't really set a low limit, on the amount of characters the user can send to the database. Meaning that any user can submit harmful code, that the server will execute next time, it is asked to execute the table the data is stored in.
This being anything from altering the table, in an unintended way, to using a DROP .SQL tag to completely wipe the table of all data.

This could have been prevented, if I only allowed users to choose from a predefined set of options where possible, and on more special columns like name, have limited the username to a few characters, so there wouldn't be space to run too much malicious code.

Another problem could be, that the site is not password protected, meaning that anyone can access anyone's profile, as long, as they have the username.

CONCLUSION

I always thought that databases and using them where immensely complicated, but after working on this project, I am very enthusiastic about how I can implement it in future projects. And thinking back on previous websites, I would have wished, that I learned to utilize this way sooner.

From a technical perspective I can conclude, that My product more that exceeds my own expectations, even though some of the functions, almost had me quit and hand in half a project. But all in all, I am convinced that I take a lot, both .PHP and .SQL from here.