CSY2028
Web Programming
Topic 9

Tom Butler
thomas.butler@northampton.ac.uk

Topic 9

- Log in forms
- Tracking log-ins across pages
- Using a database for user accounts
- Secure password storage
  - Password hashing
  - Increasing security with salts
  - PHP's inbuilt password hashing

Password protecting pages

- A very basic log in page is a HTML form that asks for a username and a password
- If the username and password are entered correctly they are "logged in" and get to see the password protected content
- If they do not enter the username/password they are shown a log-in form

Password-protected content

- This can be implemented in PHP with the following code with the valid username "csy2028" and password "secret"
- At it's most basic level, the password check is just an if statement

```php
<?php
//Was the submit button pressed?
if (isset($_POST['submit'])) {
        //Check they entered the correct username/password
        if ($_POST['username'] === 'csy2028' && $_POST['password'] === 'secret') {
                echo '<p>Welcome back ' . $_POST['username'] .'</p>';
                echo '<p>This page is only visible if you entered the correct password!</p>';
        }
        //If they didn't, display an error message
        else {
                echo '<p>You did not enter the correct username and password</p>';
        }
}
else { //The submit button was not pressed, show the log-in form
?>
<form action="login.php" method="POST">
        <label>Username: </label> <input type="text" name="username" />
        <label>Password: </label> <input type="password" name="password" />
        <input type="submit" name="submit" value="Log In" />
</form>
<?php
}
?>
```

Password-protected areas

- This works by first checking to see if the submit button was pressed
- If the submit button is pressed the username/password can be checked
- If not, a log in form is displayed

```php
<?php
//Was the submit button pressed?
if (isset($_POST['submit'])) {

}
//The submit button was not pressed, show the log-in form
else {
?>
<form action="login.php" method="POST">
        <label>Username: </label> <input type="text" name="username" />
        <label>Password: </label> <input type="password" name="password" />
        <input type="submit" name="submit" value="Log In" />
</form>
<?php
}
```

Password-protected areas

- If the submit button is pressed, the username and password from the form can be checked against the correct username/password
- If the username and password are entered correctly the password protected content is shown
- Otherwise an error message is displayed

```php
//Was the submit button pressed?
if (isset($_POST['submit'])) {
        //Check they entered the correct username/password
        if ($_POST['username'] === 'csy2028' && $_POST['password'] === 'secret') {
                echo '<p>Welcome back ' . $_POST['username'] .'</p>';
                echo '<p>This page is only visible if you entered the correct password!</p>';
        }
        //If they didn't, display an error message
        else {
                echo '<p>You did not enter the correct username and password</p>';
        }
}
```

Password-protected areas

- This allows you to add some content which is only visible if the correct username and password is displayed
- However, when using this code you will need to ask for a username and password on every page you want to password protect
- As someone navigates between pages they will need to enter a username and password on each page
- This is inefficient and difficult for the user
- It's also a lot of code!

Sessions

- There is a special type of variable known as a session variable
- Unlike normal variables it is stored between page loads
- Normally, variables will be destroyed at the end of the page

Session variables

- There is a special variable called $_SESSION
- Like $_POST this is a *superglobal* which means it is available in every function, file and class
- To enable the $_SESSION variable you must start the session
- To start a session run the code:
-

```
session_start();
```

Sessions

- session_start() must be called before any HTML is printed
- Usually this means the top of a page
- To keep things simple, just call session_start() at the very top of each page

```
<h1>Hello</h1>

<?php
session_start();
```

This will cause an error because some HTML has been printed before the session is started

```
<?php
session_start();
?>
<h1>Hello</h1>
```

Instead, you should reverse the order so the session is Started before any HTML is printed

Sessions

- Once you have started a session you can use the $_SESSION variable
- The $_SESSION variable is an array which you can read from/write to like any other array by writing to keys
- You can write to a key with any name you like
- Once you have written to the session variable, you can read from it on other pages

```php
<?php
session_start();
$_SESSION['variableName'] = 'value';
```

Sessions

You can also check to see if an index has been set using isset()

```php
<?php
session_start();

if (!isset($_SESSION['counter'])) {
        $_SESSION['counter'] = 0;
}

$_SESSION['counter']++;

echo $_SESSION['counter'];
```

Sessions

- You can clear a session by using unset on the key you want to remove:

```php
<?php
session_start();

if (!isset($_SESSION['counter'])) {
        $_SESSION['counter'] = 0;
}

$_SESSION['counter']++;

echo $_SESSION['counter'];

if ($_SESSION['counter'] >= 10) {
        unset($_SESSION['counter']);
}
```

Sessions

- Once a session is created, it is available on any page which includes the line session_start()
- Each session is unique to a user, two people can view the website and have different information in their session variable
- Sessions can store (almost) any data type, numbers strings, arrays, objects*

Exercise 1

- 1) Create a page called “name.php” which has a <form> asking the user to enter their name
- 2) Store the supplied name in a SESSION variable and display a link to welcome.php
- 3) Create another page called “weclome.php” that says “Welcome back [name]” where [name] is the name entered on the form.
  - **welcome.php should not read from $_POST or $_GET!**
- 4) Try the two pages in different browsers entering the a different name. Reloading the welcome.php page should display different information for each user. A different browser is essentially a different “user”.
- 5) Test the counter from slide 13

Sessions

- This can be used to store whether a user is logged in or not across pages

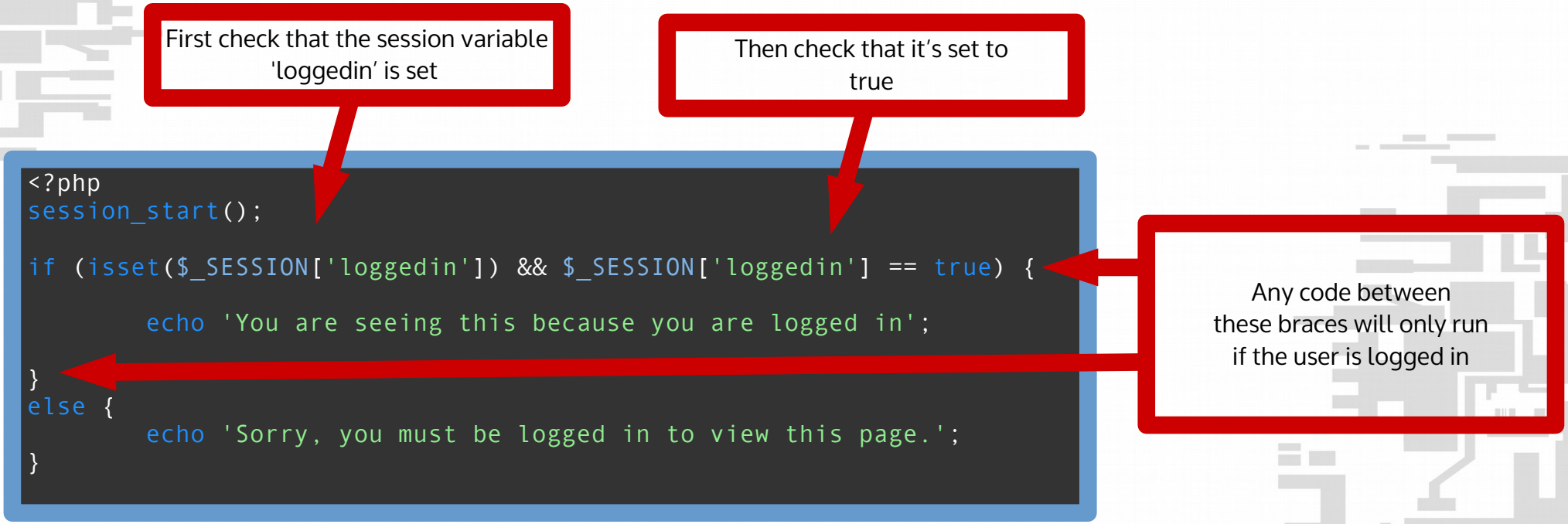
```php
<?php
session_start();

if (isset($_POST['submit'])) {
        //Check they entered the correct username/password
        if ($_POST['username'] === 'csy2028' && $_POST['password'] === 'secret') {
                $_SESSION['loggedin'] = true;
                echo '<p>Welcome back ' . $_POST['username'] .'</p>';
                echo '<p>This page is only visible if you entered the correct password!</p>';
        }
        //If they didn't, display an error message
        else {
                echo '<p>You did not enter the correct username and password</p>';
        }
}
else { //The submit button was not pressed, show the log-in form
?>
<form action="login.php" method="POST">
        <label>Username: </label>
        <input type="text" name="username" />
        <label>Password: </label>
        <input type="password" name="password" />

        <input type="submit" name="submit" value="Log In" />
</form>
<?php
}
```

Sessions

- Then, on another page you can check whether someone is logged in or not by using the code:



Sessions

- This log in check can be placed on any page you want to password protect

```php
<?php
session_start();

if (isset($_SESSION['loggedin']) && $_SESSION['loggedin'] == true) {

        echo 'You are seeing this because you are logged in';

}
else {
        echo 'Sorry, you must be logged in to view this page.';
}
```

Logging out

- To create a log out page, you need to *unset* the session:

```php
<?php
session_start();

unset($_SESSION['loggedin']);

echo 'You are now logged out';
```

Sessions

- By default sessions last for 30 minutes (you can change this by adjusting PHP's settings, but 30 minutes should be fine for most cases!)
- Each time you go to a page the timer is reset and you get another 30 minutes
- This means if you're on a page for 30 minutes without changing (e.g. typing a very long message, or going to look for your credit card on an online shop!) and then click to the next page, your session will be lost
- Because of this, you can never rely on the session being set
- When you close the browser the session is also cleared

User accounts

- The following code will allow logging in using:
  - Username: csy2028
  - Password: secret

```php
<?php
session_start();

if (isset($_POST['submit'])) {
        //Check they entered the correct username/password
        if ($_POST['username'] === 'csy2028' && $_POST['password'] === 'secret') {
                $_SESSION['loggedin'] = true;
                echo '<p>Welcome back ' . $_POST['username'] .'</p>';
                echo '<p>This page is only visible if you entered the correct password!</p>';
        }
        //If they didn't, display an error message
        else {
                echo '<p>You did not enter the correct username and password</p>';
        }

}
```

User accounts

- This may be enough for very small websites, but most websites will need to have different user accounts
- Multiple people will be able to log in with different usernames and password
  - e.g. Amazon, Ebay
- To achieve this, you need more than a simple if statement that compares strings

```
if ($_POST['username'] === 'csy2028' && $_POST['password'] === 'secret') {
```

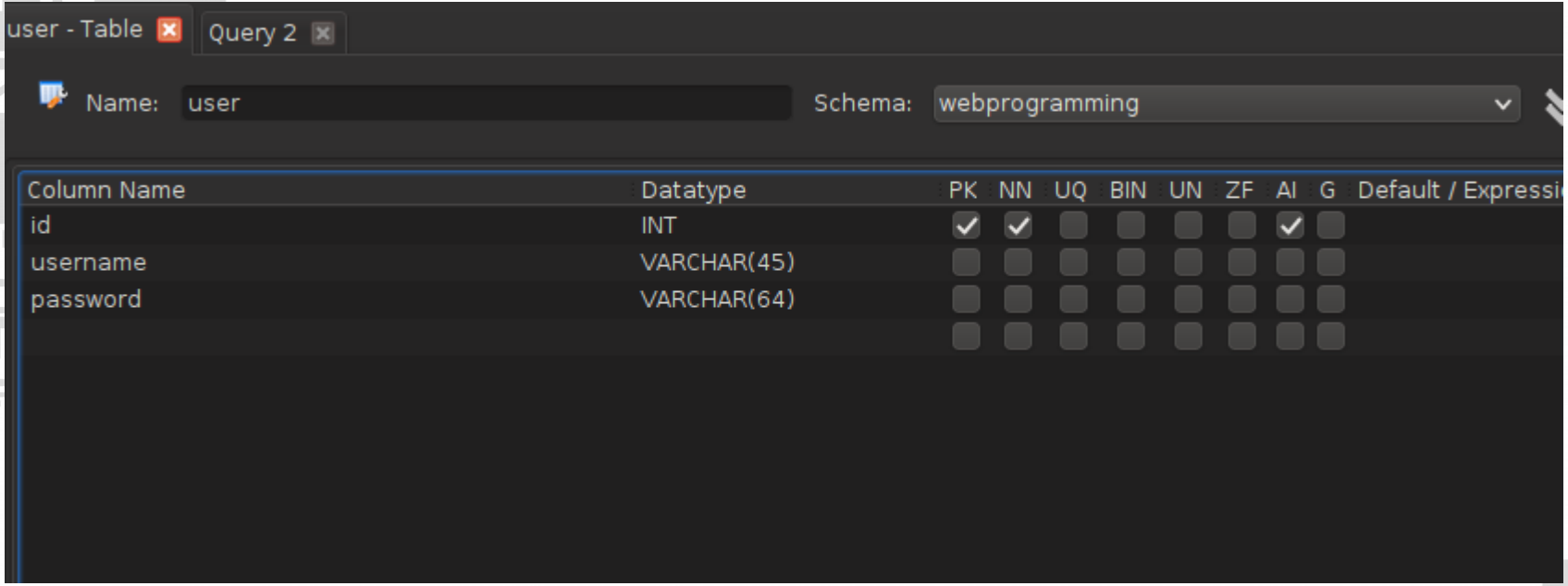
Exercise 2

- 1) Using exercise 1 as a basis, create a log-in system with *login.php* that contains a form asking for a username and a password and sets a session if the user is logged in and presents a link to "logincheck.php". Select a username and password that's used for logging in
- 2) Create "logincheck.php" which says "You are not logged in, click here to log in" and links to login.php or "You are logged in, click here to log out" and links to logout.php. This page should check the session to determine which message to show
- 3) Create logout.php which clears the session and provides a link back to logincheck.php

Databases

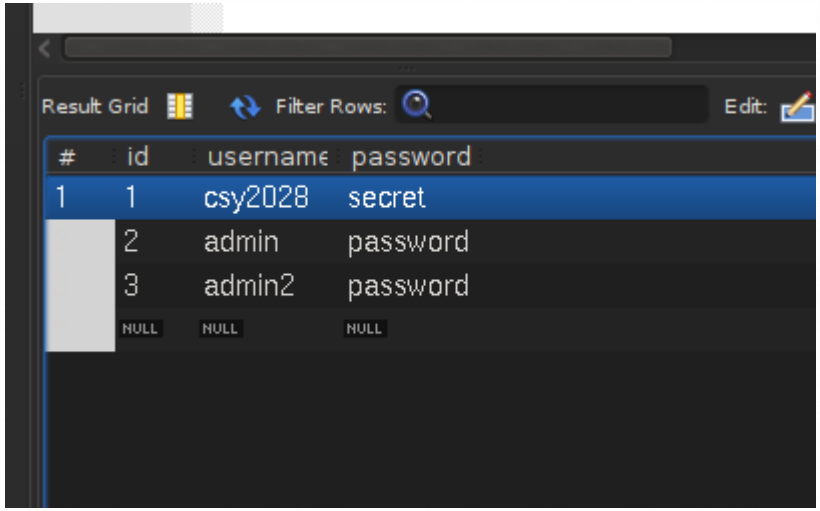
- It's possible to store user accounts in a database by performing a query instead of a simple if statement
- To do this you will need a table to store the usernames and passwords
- E.g. the following fields could be added to a database:
  - ID (INT, auto_increment)
  - Username (VARCHAR – 128 characters)
  - Password (VARCHAR – 64 characters)

Databases


user - Table
Query 2
Name: user
Schema: webprogramming
Column Name
Datatype
PK
NN
UQ
BIN
UN
ZF
AI
G
Default / Expressi
id
INT
username
VARCHAR(45)
password
VARCHAR(64)

Databases

- Users can then be added to the database by entering their usernames and passwords in the database



Databases

- Once the usernames and passwords are stored in the database the login PHP script can be used to log in
- Instead of using an if statement to compare strings, a database query can be used to check to see whether the account exists

```
if ($_POST['username'] === 'csy2028' && $_POST['password'] === 'secret') {
```

```
$stmt = $pdo->prepare('SELECT * FROM user WHERE username = :username AND password = :password');

$criteria = [
        'username' => $_POST['username'],
        'password' => $_POST['password']
];
$stmt->execute($criteria);
```

- Once you've executed the query you can read the number of returned rows using:
  - $stmt->rowCount();
- This can be used in an if statement to determine whether any rows matched the queried username and password:

```
if ($stmt->rowCount() > 0) {
        $_SESSION['loggedin'] = true;
        echo 'You are now logged in';
}
else {
        echo 'Sorry, your username and password could not be found';
}
```

Logging in

- Rather than tracking something as simple as whether or not someone is logged in or not, you may want to know *who* is logged in
- Using the code

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

- Only tracks that *someone* has logged in
- It might be useful to know who is logged in

Logging in

- Session variables can store anything
- You could have the session variable store the entire record that was matched by the login query:

```
if ($stmt->rowCount() > 0) {
        $_SESSION['loggedin'] = $stmt->fetch();
        echo 'You are now logged in';
}
else {
        echo 'Sorry, your username and password could not be found';
}
```

- This will store the user's username, password and ID in the session

Logging in

- However, this is generally a *bad idea* as you have a copy of the data in the session. If the database changes (e.g. their username or password changes) it will not be reflected inside the session
- Instead, it's better to store only the ID of the person who is logged in inside the session:

```
if ($stmt->rowCount() > 0) {
        $user = $stmt->fetch();
        $_SESSION['loggedin'] = $user['id'];
        echo 'You are now logged in';
}
else {
        echo 'Sorry, your username and password could not be found';
}
```

Logging In

- You can then fetch the current user's information from the database when you need it:

```
$stmt = $pdo->prepare('SELECT * FROM user WHERE id = :id');

$criteria = [
        'id' => $_SESSION['loggedin'];
];

$stmt->execute($criteria);

$user = $stmt->fetch();
```

- This will ensure the information you are using about this user is always up-to-date

Users

- You can add extra columns to the *user* table including their name and any other information you'd like to store about them
- For example, access levels or email addresses

Usernames

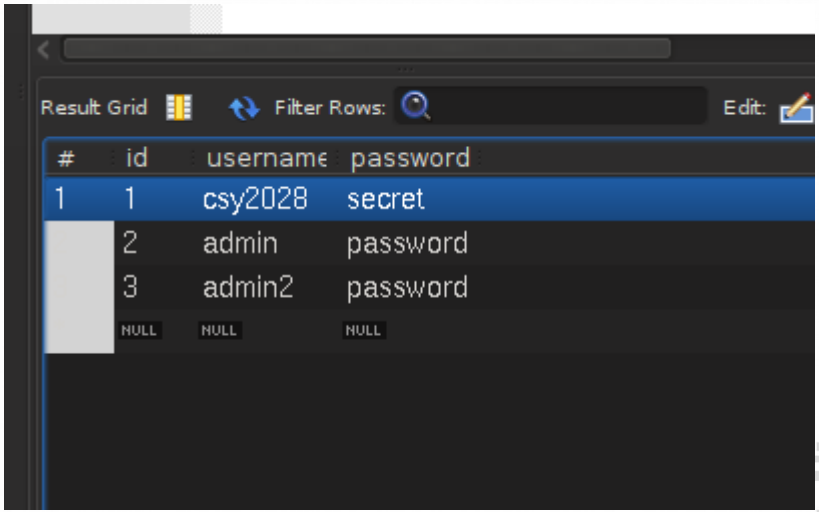
- 10-15 years ago every website required you to sign up with a *username* and a *password*.
- Each user would have to pick their unique usernames
- As more people sign up, simple, easy to remember usernames would get taken
- This resulted in people constantly forgetting their usernames
- To overcome this, it's better to ask for an *email address* in place of a username
- The user is not likely to forget their email address and there's no risk of two people trying to sign up with the same one!

Usernames

- A more recent trend is logging in using existing accounts:
  - "Log in with Facebook"
  - "Log in with Google Plus"
  - "Log in with Github"
- This makes it incredibly easy for users to sign up and log in, and gives them less to remember
- However, implementing it is beyond the scope of this lecture (and often not easy!)

Security Concerns

- The user table I created looks like this:



- The password column stores the password in *plain text*
- Anyone who has access (or gains access!) to the database can read everyone's password!

Security Concerns

- Rather than storing everyone's password in plain text so anyone else can read it, for security it's better to avoid storing the user's password
- But how can they log in if you don't store their password?
- Instead of storing the *password* you can store a *one-way encrypted* version of it

Security Concerns

- A one-way encryption is called a *hash*
- A hasting algorithm takes a string (such as a password) and converts it into an indecipherable string of seemingly random numbers and letters
- SHA1 is one such algorithm
- In PHP you can use the sha1() function to generate a SHA1 hash for a string
- Each string will generate a different *hash*

Security Concerns

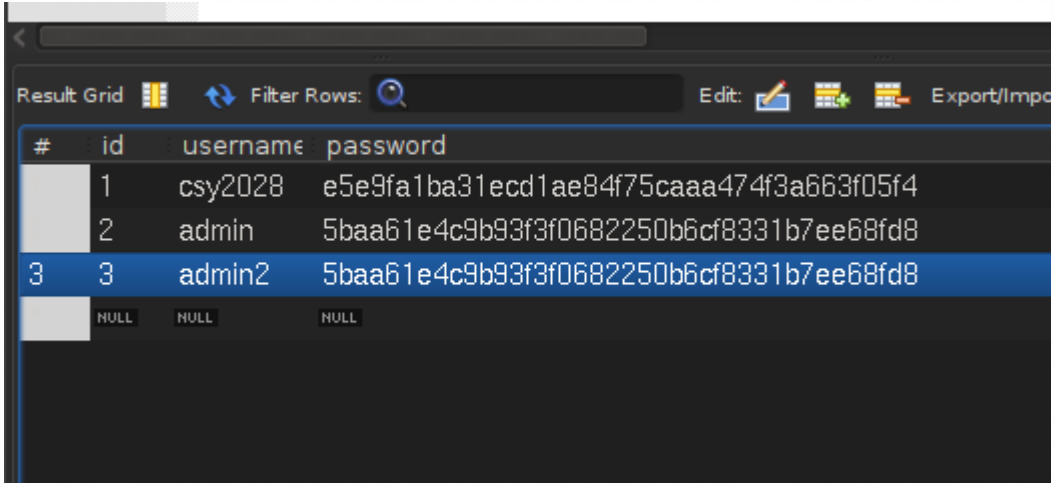
- Each time you run the hashing algorithm with the same string it will produce the same result but different strings will give different *hashes*

```
echo sha1('hello');
echo sha1('goodbye');
echo sha1('password');
echo sha1('hello');
```

```
Output:
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
3c8ec4874488f6090a157b014ce3397ca8e06d4f
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
```

Security Concerns

- This can be used in place of the *password* field in the database:



Security Concerns

- However, this will break the log in script!

```php
$stmt = $pdo->prepare('SELECT * FROM user WHERE username = :username AND password = :password');

$criteria = [
        'username' => $_POST['username'],
        'password' => $_POST['password']
];
$stmt->execute($criteria);
```

- If someone enters the correct information:
  - Username: csy2028
  - Password: secret
- Which will run the query:
- SELECT * FROM user WHERE username= 'csy2028' AND PASSWORD = 'secret'

Security Concerns

- SELECT * FROM user WHERE username= 'csy2028' AND PASSWORD = 'secret'
- Because 'secret' isn't stored in the database this will return zero results!
- Instead, the password entered by the user needs to be *hashed* before sending the query:

```
$stmt = $pdo->prepare('SELECT * FROM user WHERE username = :username AND password = :password');

$criteria = [
        'username' => $_POST['username'],
        'password' => sha1($_POST['password'])
];
$stmt->execute($criteria);
```

Security Concerns

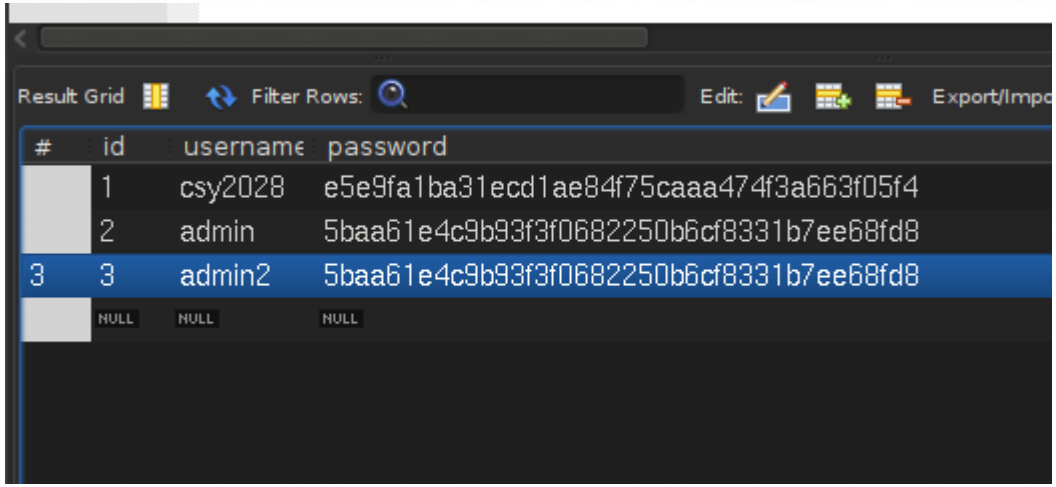
- This will generate the query:
- SELECT * FROM user WHERE username = 'csy2028' AND password = 'e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4'
- Which will now match a record in the database

```
$stmt = $pdo->prepare('SELECT * FROM user WHERE username = :username AND password = :password');

$criteria = [
        'username' => $_POST['username'],
        'password' => sha1($_POST['password'])
];
$stmt->execute($criteria);
```

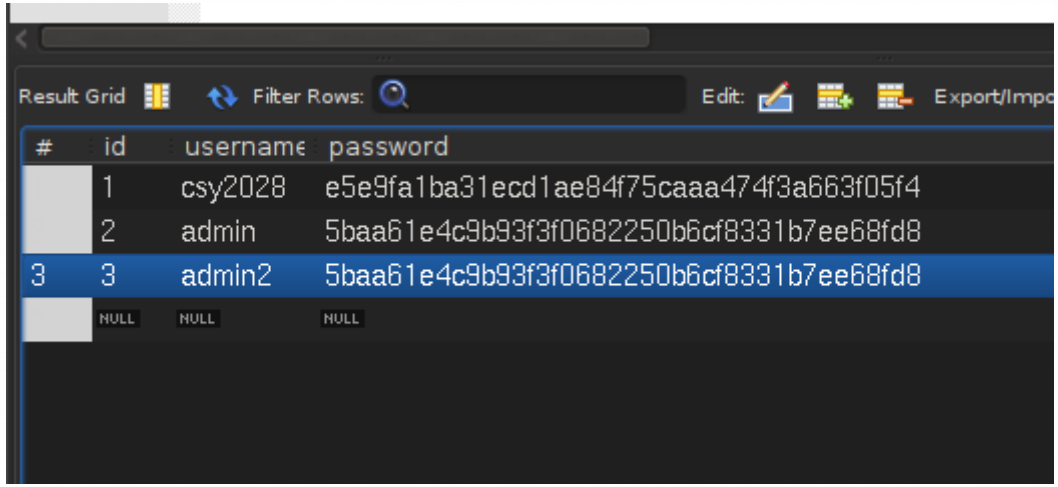
Security Concerns

- This is significantly more secure as the user's real password is never stored anywhere on the web server, only the hashed version
- If someone steals the database they will only see the hash:



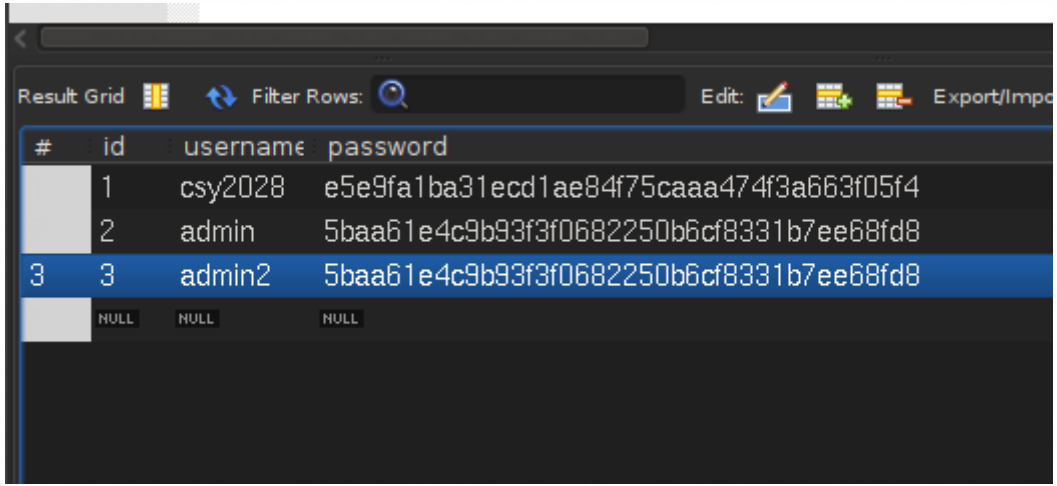
Security Concerns

- However, there is still a problem here
- What can an attacker deduce from the following information?



Security Concerns

- They can see that both admin and admin2 have the same password
- If they can work out admin2's password, they know admin's password



Security Concerns

- In fact, anyone on *any website* that is using the same hashing algorithm will have the same password stored!
- If they can crack a password on one website they will know the password of anyone else using the same password
-

Security Concerns

- Some people choose bad passwords!
- A lot of people use the same badly chosen passwords For example, the most common 5 passwords are:
  - 123456
  - password
  - 12345
  - 12345678
  - qwerty

Security Concerns

- An attacker could generate the hashes of all the most common passwords (and dictionary words) and compare it to your database
- This would let them work out if any of your users are using any passwords they have generated the hashes for!
- For example, the attacker could generate the hash for the string 'password':
  - 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
- And use a SELECT query to find anyone who is using this password

Security Concerns

- To avoid this, each user's password should be *unique* in the database!
- sha1('password'); will always generate the hash:
  - `5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8`
- However, you can add something to the password to make it unique for each user
- This is referred to as a *salt*

Salting hashes

- The salt is an extra piece of information added to the hashed string that makes the password unique to each user in the database
- E.g. if user1 and user2 both chose 'password' as their password, the has could be:
- sha1('1password');
  - 8ad742ee5d26c1b43701e598e1ed767b4352377a
- And
- sha1('2password');
  - 0c757723c6b5bd1f130082fc869f50660ed6edf4
- By doing this, it's impossible to see if two users have the same password

Salting passwords

- To use a salt you need to choose a value that will be known at the time a user logs in
- The simplest way of doing this is using the username (or email address)
- For example. When they log in with the username and password:
  - admin
  - Password
- Instead of storing **sha1('password');** in the database, we can store:
- **sha1('adminpassword');** by concatenating the username and password
- For admin2, the database will store the result of **sha1('admin2password');**
- Or **sha1($_POST['username'] . $_POST['password']);**

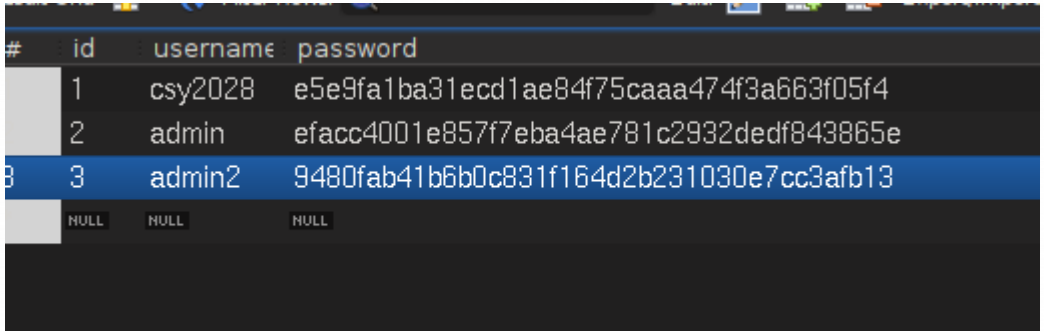
Security Concerns

- In the code this will work like this:

```
$stmt = $pdo->prepare('SELECT * FROM user WHERE username = :username AND password = :password');

$criteria = [
        'username' => $_POST['username'],
        'password' => sha1($_POST['username'] . $_POST['password'])
];
$stmt->execute($criteria);
```

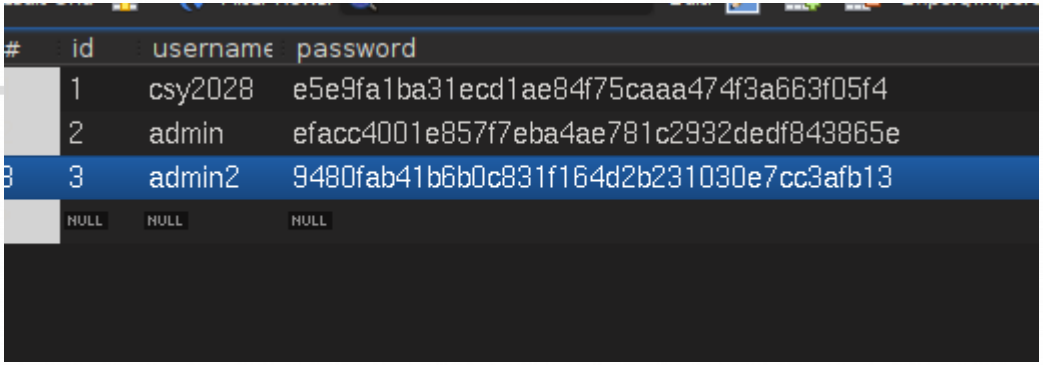
- And when you look in the database:



#	id	username	password
	1	csy2028	e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4
	2	admin	efacc4001e857f7eba4ae781c2932dedf843865e
	3	admin2	9480fab41b6b0c831f164d2b231030e7cc3afb13
	NULL	NULL	NULL

Security Concerns

- For anyone looking in the database, they can no longer tell if two users have the same password:



#	id	username	password
	1	csy2028	e5e9fa1ba31ecd1ae84f75caaa474f3a663f05f4
	2	admin	efacc4001e857f7eba4ae781c2932dedf843865e
	3	admin2	9480fab41b6b0c831f164d2b231030e7cc3afb13
	NULL	NULL	NULL

- They also can't compute hashes for common passwords to work out whether any of the users are using common passwords

PHP's inbuilt password salts

- Because this is a very common problem that needs to be solved in web applications, PHP provides a mechanism for doing this
- PHP has two inbuilt functions that allow for generating secure passwords with unique salts
- You should use this instead of trying to implement it yourself
  - It's easier to implement
  - And security concerns have all been considered for you
  - (But it’s still a good idea to understand the general concept of hashes/salts)

PHP password_hash

- To generate a password hash in PHP you can use the very simple code:

```
$hash = password_hash($password, PASSWORD_DEFAULT);
echo $hash;
```

- This generates a securely hashed password with a salt
- You should store the contents of the $hash variable in your database

PHP password_verify

- Once the secure password has been stored in the database, the log in code needs to be changed to check the password
- PHP's inbuilt password_verify() function can be used to verify a password
- This takes two arguments:
  - The plain text password
  - The hash you're comparing it against
- It returns true/false depending on whether the plain text password matches the stored hash

password_verify()

```php
if (password_verify($password, $hash)) {
    // Success!
}
else {
    // Invalid credentials
}
```

- This can be tied into the log in code with the database:

```php
$stmt = $pdo->prepare('SELECT * FROM user WHERE username = :username');

$criteria = [
        'username' => $_POST['username'],
];
$stmt->execute($criteria);

$user = $stmt->fetch();

if (password_verify($_POST['password'], $user['password'])) {
        $_SESSION['loggedin'] = $user['id'];
}
else {
        echo 'Sorry, your account could not be found';
}
```

Search for the user in the database
Note: This does not also query the Password column!

Fetch the user's information into the $user variable

Use password_verify to compare the password they typed in to the hashed password stored in the database

Exercise 3

- 1) Create a registration form that allows users to register with a username, password and name and have the information stored in the database
  - Grade D: Passwords are stored in plain text in the database
  - Grade C: Passwords are hashed using sha1, md5, sha256 or similar
  - Grade B: Passwords are hashed using a one way hash (sha1, md5, etc) but are also salted
  - Grade A: Passwords are hashed using PHP's password_hash() function
- 2) Register at least 3 different users and verify they are being stored in the database
- 3) Update your login.php to use a the database of user accounts for logging in
- 4) Amend logincheck.php to display "Welcome back [name]" where [name] is the name of the user who is logged in and check you can log in as each of the different user