Оглавление

[**Простая авторизация через базу данных на PHP** 1](#_Toc156154444)

[**Авторизация через сессию на PHP** 2](#_Toc156154445)

[**Логаут** 4](#_Toc156154446)

[**Регистрация на PHP** 4](#_Toc156154447)

[**Авторизуем сразу при регистрации** 5](#_Toc156154448)

[**Добавляем id пользователя в сессию** 5](#_Toc156154449)

[**Скрываем пароль** 6](#_Toc156154450)

[**Проверка логина на занятость** 7](#_Toc156154451)

[**Валидация данных при регистрации на PHP** 7](#_Toc156154452)

[**Хеширование пароля на PHP** 8](#_Toc156154453)

[**Добавляем соль в регистрацию** 8](#_Toc156154454)

[**Добавляем соль в авторизацию** 10](#_Toc156154455)

[**Используем функцию password\_hash** 11](#_Toc156154456)

[**Реализация профиля на PHP** 13](#_Toc156154457)

[**Личный кабинет на PHP** 13](#_Toc156154458)

[**Смена пароля на PHP** 15](#_Toc156154459)

[**Подтверждение пароля** 15](#_Toc156154460)

[**Удаление аккаунта на PHP** 16](#_Toc156154461)

[**Права доступа на PHP** 17](#_Toc156154462)

[**Изменения в регистрации** 17](#_Toc156154463)

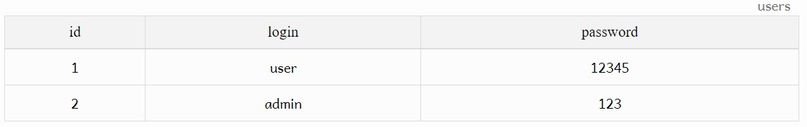
[**Нормализируем базу данных** 18](#_Toc156154464)

**Введение в аутентификацию на PHP**

Аутентификация - это процесс определения пользователя сайтом. Для этого пользователь обычно должен вбить свой логин и пароль. После этого наш сайт выполняет авторизацию этого пользователя, то есть наделяет его определенными правами на совершение операций.

## **Простая авторизация через базу данных на PHP**

Давайте реализуем самую простую авторизацию на базе данных, пока без регистрации. Вместо регистрации пользователей, мы просто вобьем их логины и пароли в таблицу в базе данных:



Сделаем теперь форму, в которую будут вбиваться логин и пароль:

<**form action="" method="POST"**>  
 <**input name="login"**>  
 <**input name="password" type="password"**>  
 <**input type="submit"**>  
</**form**>

Напишем теперь код, который будет проверять, отправлена ли форма и, если отправлена, то проверять, есть ли в базе данных пользователь с таким логином и паролем:

**<?php  
if** (!**empty**($\_POST[**'password'**]) **and** !**empty**($\_POST[**'login'**])) {  
 $login = $\_POST[**'login'**];  
 $password = $\_POST[**'password'**];

$query = **"SELECT \* FROM users WHERE login='**$login**' AND password='**$password**'"**;  
 $result = mysqli\_query($link, $query);  
 $user = mysqli\_fetch\_assoc($result);

**if** (!**empty**($user)) {  
 // прошел авторизацию  
 } **else** {  
 // неверно ввел логин или пароль  
 }  
}  
**?>**

## **Авторизация через сессию на PHP**

Наша авторизация должна работать так: пользователь, который хочет авторизоваться на сайте, заходит на страницу login.php, вбивает правильные логин и пароль и *далее ходит по страницам сайта уже будучи авторизованным*.

Чтобы другие страницы сайта знали о том, что наш пользователь авторизован, мы должны хранить в сессии пометку об этом.

Пока наша авторизация не совсем рабочая, так как сессию мы еще не подключили и другие страницы сайта не могут понять, авторизован пользователь или нет.

Будем хранить пометку об авторизации в переменной сессии $\_SESSION['auth'] - если там записано true, то пользователь авторизован, а если null - то не авторизован.

Давайте внесем соответствующую правку в наш код:

**<?php**session\_start();

**if** (!**empty**($\_POST[**'password'**]) **and** !**empty**($\_POST[**'login'**])) {  
 $login = $\_POST[**'login'**];  
 $password = $\_POST[**'password'**];

$query = **"SELECT \* FROM users WHERE login='**$login**' AND password='**$password**'"**;  
 $result = mysqli\_query($link, $query);  
 $user = mysqli\_fetch\_assoc($result);

**if** (!**empty**($user)) {  
 $\_SESSION[**'auth'**] = true;  
 } **else** {  
 // неверно ввел логин или пароль  
 }  
}  
**?>**

Теперь на любой странице сайта мы можем проверить, авторизован пользователь или нет, вот таким образом:

**<?php  
if** (!**empty**($\_SESSION[**'auth'**])) {

}  
**?>**

Можно закрыть текст какой-нибудь страницы целиком для неавторизованного пользователя:

**<?php if** (!**empty**($\_SESSION[**'auth'**])): **?>** <!DOCTYPE **html**>  
 <**html**>  
 <**head**>

</**head**>  
 <**body**>  
 <**p**>текст только для авторизованного пользователя</**p**>  
 </**body**>  
 </**html**>  
**<?php else**: **?>** <**p**>пожалуйста, авторизуйтесь</**p**>  
**<?php endif**; **?>**

Можно закрыть только часть страницы:

<!DOCTYPE **html**>  
<**html**>  
<**head**>

</**head**>  
<**body**>  
<**p**>текст для любого пользователя</**p**>  
**<?php  
if** (!**empty**($\_SESSION[**'auth'**])) {  
 **echo 'текст только для авторизованного пользователя'**;  
}  
**?>**<**p**>текст для любого пользователя</**p**>  
</**body**>  
</**html**>

## **Логаут**

Авторизованный пользователь должен возможность перестать быть авторизованным, то есть совершить выход из своего аккаунта. Для этого нужно сделать отдельную страницу и удалять на ней пометку об авторизации, примерно вот так:

**<?php**session\_start();  
$\_SESSION[**'auth'**] = **null**;  
**?>**

## **Регистрация на PHP**

Давайте теперь реализуем регистрацию. Для этого сделаем отдельную страницу register.php. По заходу на эту страницу, пользователь должен увидеть форму, в которую он должен вбить желаемый логин и пароль:

<**form action="" method="POST"**>  
 <**input name="login"**>  
 <**input name="password" type="password"**>  
 <**input type="submit"**>  
</**form**>

По нажатию на кнопку отправки форму логин и пароль пользователя должны занестись в базу данных с помощью INSERT запроса, вот так:

**<?php  
if** (!**empty**($\_POST[**'login'**]) **and** !**empty**($\_POST[**'password'**])) {  
 $login = $\_POST[**'login'**];  
 $password = $\_POST[**'password'**];

$query = **"INSERT INTO users SET login='**$login**', password='**$password**'"**;  
 mysqli\_query($link, $query);  
}  
**?>**

Затем пользователь может зайти на страницу авторизации, ввести логин и пароль, под которым он зарегистрировался и авторизоваться.

## **Авторизуем сразу при регистрации**

Сейчас наша регистрация сделана таким образом, что пользователь первый раз вбивает логин-пароль регистрируясь, а потом сразу же вбивает их второй раз, желая зайти на сайт.

Это на самом деле не очень удобно и будет раздражать пользователей. Лучше сделать так, чтобы при успешной регистрации сразу же происходила автоматическая авторизация. Для этого сразу после успешной регистрации запишем в сессию пометку об успешной авторизации:

**<?php  
if** (!**empty**($\_POST[**'login'**]) **and** !**empty**($\_POST[**'password'**])) {  
 $login = $\_POST[**'login'**];  
 $password = $\_POST[**'password'**];

$query = **"INSERT INTO users SET login='**$login**', password='**$password**'"**;  
 mysqli\_query($link, $query);

$\_SESSION[**'auth'**] = **true**; // пометка об авторизации  
}  
**?>**

## **Добавляем id пользователя в сессию**

Пусть кроме пометки об авторизации мы бы хотели записать в сессию еще и его id.

В этом случае мы можем получить его с помощью функции mysqli\_insert\_id. Эта функция получает id последней вставленной в данном скрипте записи, то есть как раз то, что нам нужно.

Реализуем:

**<?php  
if** (!**empty**($\_POST[**'login'**]) **and** !**empty**($\_POST[**'password'**])) {  
 $login = $\_POST[**'login'**];  
 $password = $\_POST[**'password'**];

$query = **"INSERT INTO users SET login='**$login**', password='**$password**'"**;  
 mysqli\_query($link, $query);

$\_SESSION[**'auth'**] = **true**;

$id = mysqli\_insert\_id($link);  
 $\_SESSION[**'id'**] = $id; // пишем id в сессию  
}  
**?>**

## **Скрываем пароль**

Поле ввода пароля обычно представляет собой инпут с типом password, в котором введенные символы скрываются под звездочками. Это сделано для того, чтобы злоумышленник не мог подсмотреть пароль пользователя через плечо в момент регистрации.

Сокрытие пароля таким образом конечно хорошо, но есть, однако, проблема - пользователь не видит, что вводит. Он может ошибиться при вводе какого-либо символа и зарегистрироваться не с тем паролем, с которым он хотел. Это будет печально:(, так как он затем не сможет авторизоваться на сайте.

Существует стандартное решение этой проблемы: пользователю показываются два инпута для ввода пароля - в первый инпут он вводит пароль, а во второй инпут - его подтверждение, то есть этот же пароль второй раз:

<**form action="" method="POST"**>  
 <**input name="login"**>  
 <**input type="password" name="password"**>  
 <**input type="password" name="confirm"**>  
 <**input type="submit"**>  
</**form**>

Задача нашего сайта - проверить, что пароль и его подтверждение совпадают, так как логично, что в этом случае пользователь ввел именно то, что хотел ввести:

**<?php  
if** (!**empty**($\_POST[**'login'**]) **and** !**empty**($\_POST[**'password'**]) **and** !**empty**($\_POST[**'confirm'**])) {  
 **if** ($\_POST[**'password'**] == $\_POST[**'confirm'**]) {  
 // регистрируем  
 } **else** {  
 // выведем сообщение о несовпадении  
 }  
}  
**?>**

## **Проверка логина на занятость**

Сейчас наша регистрация имеет одну проблему - новый пользователь нашего сайта может зарегистрироваться под уже существующим логином, что, конечно же, недопустимо.

Для решения проблемы необходимо перед запросом на добавление нового пользователя в базу данных, выполнить SELECT запрос, который проверит, занят желаемый логин или нет. Если не занят - регистрируем, если занят - не регистрируем, а выводим сообщение об этом.

Давайте напишем этот код:

**<?php  
if** (!**empty**($\_POST[**'login'**]) **and** !**empty**($\_POST[**'password'**])) {  
 $login = $\_POST[**'login'**];  
 $password = $\_POST[**'password'**];

$query = **"SELECT \* FROM users WHERE login='**$login**'"**;  
 $user = mysqli\_fetch\_assoc(mysqli\_query($link, $query));

**if** (**empty**($user)) {  
 $query = **"INSERT INTO users SET login='**$login**', password='**$password**'"**;  
 mysqli\_query($link, $query);

$\_SESSION[**'auth'**] = **true**;

} **else** {  
 // логин занят, выведем сообщение об этом  
 }  
}  
**?>**

## **Валидация данных при регистрации на PHP**

Сейчас мы не накладываем никаких ограничений на пару логин-пароль, однако, это неправильно. К примеру, сейчас у нас пользователи случайно или намеренно могут зарегистрироваться с пустым логином или паролем, или с паролем, состоящим из одного символа. Такой пароль будет слишком простым и не безопасным.

Учтите, что если какое-то поле вбито некорректно, форма не должна очищаться, так как это будет доставлять ему неудобство пользователю: он вводил-вводил данные, нажал - и все пропало, хотя ошибка возможно была в одном символе.

## **Хеширование пароля на PHP**

Хранить пароль в открытом виде - неправильно. Хакер-злоумышленник может получить доступ к вашей базе данных и украсть пароли.

Поэтому обычно логин хранится в открытом виде, а пароль хешируется специальной функцией md5, которая параметром принимает пароль, а возвращает его хеш, по которому нельзя восстановить этот самый пароль.

Давайте, например, найдем хеш какой-нибудь строки:

<?php echo md5('12345'); //выведет '827ccb0eea8a706c4c34a16891f84e7b' ?>

Сейчас нам необходимо переделать нашу регистрацию и нашу авторизацию. Для начала я бы советовал очистить таблицу с юзерами, так как там сейчас хранятся пароли в открытом виде, а должны хранится их хеши. Затем при тестировании регистрации таблица заполнится данными в новом формате.

Давайте теперь поправим нашу регистрацию так, чтобы при сохранении нового пользователя в базу добавлялся не пароль, а его хеш.

Описанная правка будет представлять собой что-то такое:

**<?php**$login = $\_POST[**'login'**];  
$password = md5($\_POST[**'password'**]); // преобразуем пароль в его хеш

$query = **"INSERT INTO users SET login='**$login**', password='**$password**'"**;  
**?>**

Внесем аналогичные правки в авторизацию:

**<?php**$login = $\_POST[**'login'**];  
$password = md5($\_POST[**'password'**]); // преобразуем пароль в его хеш

$query = **"SELECT \* FROM users WHERE login='**$login**' AND password='**$password**'"**;  
**?>**

## **Добавляем соль в регистрацию**

Итак, вы уже знаете, что хеширование через md5 - необратимый процесс и хакер, получивший доступ к хешу, не сможет получить по этому хешу пароль.

На самом деле это утверждение не совсем верное - в настоящее время злые хакеры составили библиотеки хешей популярных и не очень паролей и любой дурак может разгадать пароль, просто загуглив его хеш.

Речь идет о достаточно простых, популярных паролях.

Загуглите, например, хеш 827ccb0eea8a706c4c34a16891f84e7b и сразу в поиске гугла вы увидите, что это пароль '12345'.

Хеши достаточно сложных паролей таким образом не разгадать (попробуйте).

Вы можете спросить, в чем тогда проблема - давайте все мы будем регистрироваться со сложными паролями. Есть, однако, проблема - большинство пользователей не задумываются о безопасности своих данных и могут вводить достаточно простые пароли.

Мы можем при регистрации заставлять придумывать более длинные пароли, ограничивая, к примеру, минимальное количество символов 6-ю или 8-ю, однако, все равно будут появляться пароли вида '123456' или '12345678'.

Можно, конечно, придумать более умный алгоритм проверки пароля на сложность, но есть другое решение.

Суть этого решения такая: пароли надо посолить. Соль - это специальная случайная строка, которая будет добавляться к паролю при регистрации и хеш уже будет вычисляться не от простого пароля типа, а от строки соль+пароль, то есть от соленого пароля.

То есть при регистрации вы будете делать что-то типа такого:

**<?php**$salt = **'1sJg3hfdf'**; // соль - сложная случайная строка  
$password = md5($salt . $\_POST[**'password'**]); // преобразуем пароль в соленый хеш  
**?>**

При этом соль будет разная для каждого пользователя, ее нужно будет генерировать случайным образом в момент регистрации.

Вот готовая функция, которая сделает это:

**<?php  
function** generateSalt()  
{  
 $salt = **''**;  
 $saltLength = 8; // длина соли

**for**($i = 0; $i < $saltLength; $i++) {  
 $salt .= chr(mt\_rand(33, 126)); // символ из ASCII-table  
 }

**return** $salt;  
}  
**?>**

С помощью этой функции можно переписать наш код вот так:

**<?php**$salt = generateSalt(); // соль  
$password = md5($salt . $\_POST[**'password'**]); // соленый пароль  
**?>**

Еще раз повторю, что это были изменения при регистрации - в БД сохраняем не просто хеш пароля, а хеш соленого пароля.

Это еще не все: в таблице с юзерами кроме поля login и password нужно сделать еще и поле salt, в котором мы будем хранить соль каждого пользователя.

## **Добавляем соль в авторизацию**

Теперь нам необходимо поменять авторизацию. Здесь уже изменения будут более существенными.

Уже не получится проверить правильность пары логин-пароль сразу же, одним запросом. Почему: потому что, чтобы проверить пароль, надо получить его соленый хеш, а соль хранится в базе данных и уникальная для каждого логина.

Придется вначале получить запись только по логину, прочитать соль, посолить введенный пароль и сравнить с соленым паролем из базы и только, если они совпадают, - авторизовывать пользователя.

Учтите, что может такое быть, что логин вбит неправильно, в этом случае проверку пароля можно не осуществлять, а сразу вывести, что авторизация не возможна - данные не верны:

**<?php**$login = $\_POST[**'login'**];

$query = **"SELECT \* FROM users WHERE login='**$login**'"**;  
$result = mysqli\_query($link, $query);  
$user = mysqli\_fetch\_assoc($result);

**if** (!**empty**($user)) {  
 // пользователь с таким логином есть, теперь надо проверять пароль...  
} **else** {  
 // пользователя с таким логином нет, выведем сообщение  
}  
**?>**

Давайте добавим проверку пароля:

**<?php**$login = $\_POST[**'login'**];

$query = **"SELECT \* FROM users WHERE login='**$login**'"**;  
$result = mysqli\_query($link, $query);  
$user = mysqli\_fetch\_assoc($result);

**if** (!**empty**($user)) {  
 $salt = $user[**'salt'**]; // соль из БД  
 $hash = $user[**'password'**]; // соленый пароль из БД

$password = md5($salt, $\_POST[**'password'**]); // соленый пароль от юзера

// Сравниваем соленые хеши  
 **if** ($password == $hash) {  
 // все ок, авторизуем...  
 } **else** {  
 // пароль не подошел, выведем сообщение  
 }  
} **else** {  
 // пользователя с таким логином нет, выведем сообщение  
}  
**?>**

*В целях безопасности пользователю обычно не сообщают, что именно не подошло - логин или пароль, чтобы усложнить подбор пар логин-пароль хакерами. Просто выводят сообщение о том, что пара логин-пароль неверна или что-то в таком роде.*

## **Используем функцию password\_hash**

На самом деле функция md5 и соление пароля с ее помощью считается устаревшим. Мы изучали ее, чтобы вы поняли дальнейший материал, а также потому, что вы можете столкнуться с этим, работая с чужими проектами.

Существует более совершенный способ получить соленый пароль. Для этого используется функция password\_hash. Первым параметром она принимает строку, а вторым - алгоритм шифрования (о нем позднее), и возвращает хеш этой строки вместе с солью.

Попробуйте несколько раз запустите этот код:

<?php echo password\_hash('12345', PASSWORD\_DEFAULT); ?>

Вы каждый раз будете получать разный результат и в этом результате первая часть строки будет являться солью, а вторая часть - соленым паролем.

Пусть у нас есть хеш, полученный из функции password\_hash и какой-то пароль. Чтобы проверить, это хеш этого пароля или нет, следует использовать функцию password\_verify - первым параметром она принимает пароль, а вторым - хеш, и возвращает true или false.

Давайте посмотрим на примере:

**<?php**$password = **'12345'**; // пароль  
$hash = **'$2y$10$xoYFX1mFPxBSyxaRe3iIRutxkIWhxGShzEhjYUVd3qpCUKfJE1k7a'**; // хеш

**if** (password\_verify($password, $hash)) {  
 // хеш от этого пароля  
} **else** {  
 // хеш не от этого пароля  
}  
**?>**

Что это дает нам на практике: мы можем не создавать в базе данных отдельное поле для хранения соли, не заморачиваться с генерированием этой соли - PHP все сделает за нас!

То есть получится, что в базе данных в поле password мы будем хранить соленый пароль вместе с его солью. При этом хешированный пароль будет иметь большую длину. Поэтому в базе данных нам нужно исправить размер поля с паролем и установить ее в 60 символов.

Теперь давайте поправим код регистрации. Вот то, что есть сейчас:

**<?php  
function** generateSalt()  
{  
 $salt = **''**;  
 $saltLength = 8; // длина соли

**for**($i = 0; $i < $saltLength; $i++) {  
 $salt .= chr(mt\_rand(33, 126)); // символ из ASCII-table  
 }

**return** $salt;  
}

$salt = generateSalt(); // соль  
$password = md5($salt . $\_POST[**'password'**]); // преобразуем пароль в соленый хеш  
**?>**

С помощью password\_hash мы сократим это до:

**<?php**$password = password\_hash($\_POST[**'password'**], **PASSWORD\_DEFAULT**);  
**?>**

Аналогичным образом подправится код авторизации:

**<?php**$login = $\_POST[**'login'**];

$query = **"SELECT \* FROM users WHERE login='**$login**'"**; // получаем юзера по логину  
$result = mysqli\_query($link, $query);  
$user = mysqli\_fetch\_assoc($result);

**if** (!**empty**($user)) {  
 $hash = $user[**'password'**]; // соленый пароль из БД

// Проверяем соответствие хеша из базы введенному паролю  
 **if** (password\_verify($\_POST[**'password'**], $hash)) {  
 // все ок, авторизуем...  
 } **else** {  
 // пароль не подошел, выведем сообщение  
 }  
} **else** {  
 // пользователя с таким логином нет, выведем сообщение  
}  
**?>**

## **Реализация профиля на PHP**

Давайте теперь реализуем просмотр профиля пользователя. Под профилем понимается информация, которую этот пользователь указал при регистрации.

Сделаем так, чтобы можно было смотреть профиль любого из пользователей. Для этого сделаем страницу profile.php, в которую GET параметром будем передавать id пользователя, которого мы хотим посмотреть.

На странице профиля мы будем показывать не всю информацию, которую указал о себе пользователь. К примеру, пароль там показывать точно не стоит. Кроме того, скорее всего показ емейла также будет лишним, так как в этом случае спамеры могут собирать эти емейлы программами-парсерами и рассылать спам на них.

## **Личный кабинет на PHP**

Под личным кабинетом понимается место, где пользователь может редактировать данные своего профиля. Давайте сделаем страницу account.php, зайдя на которую пользователь увидит форму для редактирования данных своего профиля (кроме логина и пароля, их следует обрабатывать особым образом).

Обратите внимание на то, что id пользователя мы не передаем GET параметром - мы сделаем так, чтобы каждый пользователь на странице account.php видел данные своего профиля, а не чужого.

Для этого следует при авторизации пользователя мы должны в сессию записать id этого пользователя, вот так:

**<?php  
if** (password\_verify($\_POST[**'password'**], $hash)) {  
 $\_SESSION[**'auth'**] = **true**;  
 $\_SESSION[**'id'**] = $user[**'id'**];  
}  
**?>**

Затем по заходу на страницу [account.php](https://vk.com/away.php?to=http%3A%2F%2Faccount.php&cc_key=" \t "_blank) мы будем делать SELECT запрос, который будет доставать из БД пользователя с id из сессии:

**<?php**$id = $\_SESSION[**'id'**];  
$query = **"SELECT \* FROM users WHERE login='**$id**'"**;

$result = mysqli\_query($link, $query);  
$user = mysqli\_fetch\_assoc($result);  
**?>**

Данные пользователя мы должны показать в форме для редактирования. Пусть, к примеру, это будут имя и фамилия пользователя:

<**form action="" method="POST"**>  
 <**input name="name" value="<?=** $user[**'name'**] **?>"**>  
 <**input name="surname" value="<?=** $user[**'surname'**] **?>"**>  
 <**input type="submit" name="submit"**>  
</**form**>

После нажатия на кнопку отправки формы мы должны будем сделать запрос на обновление юзера:

**<?php  
if** (!**empty**($\_POST[**'submit'**])) {  
 $name = $\_POST[**'name'**];  
 $surname = $\_POST[**'surname'**];

$query = **"UPDATE users SET name='**$name**', surname='**$surname**' WHERE id=**$id**"**;  
 mysqli\_query($link, $query);  
}  
**?>**

## **Смена пароля на PHP**

Смену пароля нельзя просто сделать в личном кабинете. Дело в том, что пользователь может оставить свой компьютер без присмотра (например, в офисе), будучи авторизованным на нашем сайте. В этом случае, если разрешить просто менять пароль, то злоумышленник-недоброжелатель сможет сменить пароль на другой, что, конечно же, очень плохо.

Необходимо сделать так, чтобы смена пароля на новый требовала ввода старого пароля.

Давайте реализуем страницу changePassword.php, зайдя на которую пользователь увидит форму с двумя инпутами - в первый он должен будет ввести свой старый пароль, а во вторую - новый:

<**form action="" method="POST"**>  
 <**input name="old\_password"**>  
 <**input name="new\_password"**>  
 <**input type="submit" name="submit"**>  
</**form**>

По нажатию на кнопку отправки мы должны сделать следующее:

**<?php**$id = $\_SESSION[**'id'**]; // id юзера из сессии  
$query = **"SELECT \* FROM users WHERE id='**$id**'"**;

$result = mysqli\_query($link, $query);  
$user = mysqli\_fetch\_assoc($result);

$hash = $user[**'password'**]; // соленый пароль из БД  
$oldPassword = $\_POST[**'old\_password'**];  
$newPassword = $\_POST[**'new\_password'**];

// Проверяем соответствие хеша из базы введенному старому паролю  
**if** (password\_verify($oldPassword, $hash)) {  
 $newPasswordHash = password\_hash($newPassword, **PASSWORD\_DEFAULT**);

$query = **"UPDATE users SET password='**$newPasswordHash**' WHERE id='**$id**'"**;  
 mysqli\_query($link, $query);  
} **else** {  
 // старый пароль введен неверно, выведем сообщение  
}  
**?>**

## **Подтверждение пароля**

Есть еще один нюанс: старый и новый пароли следует вводить в инпуты с типом password. Получается, что пользователь не будет видеть, что именно он ввел вместо нового пароля, что чревато ошибками. Необходимо сделать так, чтобы для нового пароля вводился сам пароль и его подтверждение:

<**form action="" method="POST"**>  
 <**input type="password" name="old\_password"**>  
 <**input type="password" name="new\_password"**>  
 <**input type="password" name="new\_password\_confirm"**>  
 <**input type="submit" name="submit"**>  
</**form**>

## **Удаление аккаунта на PHP**

Давайте теперь сделаем возможность пользователю удалить свой аккаунт. Реализуйте для этого отдельную PHP страницу. По заходу на нее пользователь должен увидеть форму, в которую необходимо вбить пароль. Только после того, как будет вбит верный пароль следует удалить аккаунт.

Дело в том, что удаление аккаунта - это важная операция, а на все операции такого рода необходимо запрашивать пароль, чтобы убедится в том, что это не злоумышленник получил доступ к компьютеру пользователя.

Покажу ключевой кусочек кода:

**<?php**$id = $\_SESSION[**'id'**];  
$query = **"SELECT \* FROM users WHERE id='**$id**'"**;

$result = mysqli\_query($link, $query);  
$user = mysqli\_fetch\_assoc($result);

$hash = $user[**'password'**]; // соленый пароль из БД

// Проверяем соответствие хеша из базы введенному старому паролю  
**if** (password\_verify($\_POST[**'password'**], $hash)) {  
 $query = **"DELETE FROM users WHERE id='**$id**'"**;  
 mysqli\_query($link, $query);  
} **else** {  
 // пароль введен неверно, выведем сообщение  
}  
**?>**

## **Права доступа на PHP**

Чаще всего на сайте не один тип пользователей, а несколько и все они имеют разные права. К примеру, у вас могут быть обычные пользователи и админы. Админы будут иметь больше прав, чем обычные юзеры.

Каким образом это реализуется: создадим в таблице с юзерами еще одно поле, назовем его status и для каждого пользователя будем хранить его статус: пусть для администраторов слово 'admin', а для обычных пользователей - слово 'user'.

Теперь при авторизации пользователя в $\_SESSION['status'] запишем статус пользователя из базы данных:

**<?php  
if** (password\_verify($\_POST[**'password'**], $hash)) {  
 $\_SESSION[**'auth'**] = **true**;  
 $\_SESSION[**'id'**] = $user[**'id'**];  
 $\_SESSION[**'status'**] = $user[**'status'**]; // записываем статус  
}  
**?>**

Пусть теперь у нас на сайте есть какая-то страница, к который имеют доступ только админы. Сделаем так, чтобы только админы видели контент этой страницы:

**<?php  
if** (!**empty**($\_SESSION[**'auth'**]) **and** $\_SESSION[**'status'**] === **'admin'**) {  
 // покажем контент страницы только админам  
}  
**?>**

## **Изменения в регистрации**

Также нам необходимо внести изменения в нашу регистрацию. Теперь при регистрации пользователя мы должны в INSERT запросе указывать его статус.

Обычно при начальной регистрации все пользователи нашего сайта получают самый низший статус, то есть в нашем случае 'user':

**<?php**$query = **"INSERT INTO users  
 SET login='**$login**', password='**$password**', 'status'='user'"**;  
**?>**

Статусы повыше обычно раздает администратор. Он в админке видит список пользователей и может любого сделать, к примеру, администратором.

Как же на сайте в таком случае появится первый администратор? Самое простое: зарегистрировать обычного пользователя и через PhpMyAdmin сделать его админом.

Более сложное: при первом запуске на хостинге запускать установку сайта, спросить с помощью формы логин и пароль и зарегистрировать самого первого пользователя сайта, которому и присвоить статус администратора.

## **Нормализируем базу данных**

Сейчас мы храним статусы наших пользователей в той же таблице, где и самих пользователей. Это, однако, неправильно - у нас получается не нормализованная таблица, ведь слова 'user' и 'admin' повторяются много раз.

Необходимо выполнить нормализацию - вынесем наши статусы в отдельную таблицу statuses:



А в таблице users сделаем колонку status\_id. Теперь при регистрации мы в колонку status\_id будем записывать id статуса из таблицы statuses:

**<?php**$query = **"INSERT INTO users  
 SET login='**$login**', password='**$password**', status\_id='1'"**;  
**?>**

Самые сложные изменения произойдут при авторизации: для того, чтобы получить статус пользователя, нужно будет выполнить LEFT JOIN:

**<?php**$login = $\_POST[**'login'**];

// Получаем юзера по логину и джойним статус:  
$query = **"SELECT \*, statuses.name as status FROM users  
 LEFT JOIN statuses  
 ON users.status\_id=statuses.id WHERE login='**$login**'"**;

$result = mysqli\_query($link, $query);  
$user = mysqli\_fetch\_assoc($result);

**if** (!**empty**($user)) {  
 $hash = $user[**'password'**];

**if** (password\_verify($\_POST[**'password'**], $hash)) {  
 $\_SESSION[**'auth'**] = **true**;  
 $\_SESSION[**'status'**] = $user[**'status'**]; // статус  
 } **else** {

}  
} **else** {

}  
**?>**