Работа с MySQL в PHP[¶](https://xsltdev.ru/php/tutorial/db/mysql/#mysql-php)

PHP поддерживает работу с базой данных MySQL. Специальные встроенные функции для работы с MySQL позволяют просто и эффективно работать с этой СУБД: выполнять любые запросы, читать и записывать данные, обрабатывать ошибки.

Сценарий, который подключается к БД, выполняет запрос и показывает результат, будет состоять всего из нескольких строк. Для работы с MySQL не надо ничего дополнительно устанавливать и настраивать; всё необходимое уже доступно вместе со стандартной поставкой PHP.

Что такое mysqli?[¶](https://xsltdev.ru/php/tutorial/db/mysql/#mysqli)

*mysqli* (MySQL Improved) — это расширение PHP, которое добавляет в язык полную поддержку баз данных MySQL. Это расширение поддерживает множество возможностей современных версий MySQL.

Как выглядит работа с базой данных[¶](https://xsltdev.ru/php/tutorial/db/mysql/#_1)

Типичный процесс работы с СУБД в PHP-сценарии состоит из нескольких шагов:

1. Установить подключение к серверу СУБД, передав необходимые параметры: адрес, логин, пароль.
2. Убедиться, что подключение прошло успешно: сервер СУБД доступен, логин и пароль верные и так далее.
3. Сформировать правильный SQL запрос (например, на чтение данных из таблицы).
4. Убедиться, что запрос был выполнен успешно.
5. Получить результат от СУБД в виде массива из записей.
6. Использовать полученные записи в своём сценарии (например, показать их в виде таблицы).

Функция mysqli connect: соединение с MySQL[¶](https://xsltdev.ru/php/tutorial/db/mysql/" \l "mysqli-connect-mysql" \o "Permanent link)

Перед началом работы с данными внутри MySQL, нужно открыть соединение с сервером СУБД.

В PHP это делается с помощью стандартной функции mysqli\_connect(). Функция возвращает результат — ресурс соединения. Данный ресурс используется для всех следующих операций с MySQL.

Но чтобы выполнить соединение с сервером, необходимо знать как минимум три параметра:

* Адрес сервера СУБД;
* Логин;
* Пароль.

Если вы следовали стандартной процедуре установки MySQL или используете OpenServer, то адресом сервера будет localhost, логином — root. При использовании OpenServer пароль для подключения — это пустая строка ‘’, а при самостоятельной установке MySQL пароль вы задавали в одном из шагов мастера установки.

Базовый синтаксис функции mysqli\_connect():

mysqli\_connect(<адрес сервера>, <имя пользователя>, <пароль>, <имя базы данных>);

Проверка соединения[¶](https://xsltdev.ru/php/tutorial/db/mysql/#_2)

Первое, что нужно сделать после соединения с СУБД — это выполнить проверку, что оно было успешным.

Эта проверка нужна, чтобы исключить ошибку при подключении к БД. Неверные параметры подключения, неправильная настройка или высокая нагрузка заставит MySQL отвеграть новые подключения. Все эти ситуации приведут к невозможности соединения, поэтому программист должен проверить успешность подключения к серверу, прежде чем выполнять следующие действия.

Соединение с MySQL устанавливается один раз в сценарии, а затем используется при всех запросах к БД.

Результатом выполнения функции mysqli\_connect() будет значение специального типа — ресурс.

Если подключение к MySQL не удалось, то функция mysqli\_connect() вместо ресурса вернет логическое значение типа «ложь» — false.

Хорошей практикой будет всегда проверять значение результа выполнения этой функции и сравнивать его с ложью.

Соединение с MySQL и проверка на ошибки:

<?php

$link = mysqli\_connect("localhost", "root", "");

if ($link == false){

print("Ошибка: Невозможно подключиться к MySQL " . mysqli\_connect\_error());

}

else {

print("Соединение установлено успешно");

}

Функция mysqli\_connect\_error() просто возвращает текстовое описание последней ошибки MySQL.

Установка кодировки[¶](https://xsltdev.ru/php/tutorial/db/mysql/#_3)

Первым делом после установки соединения крайне желательно явно задать кодировку, которая будет использоваться при обмене данными с MySQL. Если этого не сделать, то вместо записей со значениями, написанными кириллицой, можно получить последовательность из знаков вопроса: ?????????????????. Вызовите эту функцию сразу после успешной установки соединения: mysqli\_set\_charset($con, "utf8");

Выполнение запросов[¶](https://xsltdev.ru/php/tutorial/db/mysql/#_4)

Установив соединение и определив кодировку мы готовы выполнить SQL-запросы. Запросы из php-сценария. Помогут в этом несколько встроенных функций языка.

Два вида запросов[¶](https://xsltdev.ru/php/tutorial/db/mysql/#_5)

Следует разделять все SQL-запросы на две группы:

* Чтение информации (SELECT).
* Модификация (UPDATE, INSERT, DELETE).

При выполнении запросов из среды PHP, запросы из второй группы возвращают только результат их исполнения: успех или ошибку.

Запросы первой группы при успешном выполнении возвращают специальный ресурс результата. Его, в свою очередь, можно преобразовать в ассоциативный массив (если нужна одна запись) или в двумерный массив (если требуется список записей).

*Добавление записи*[*¶*](https://xsltdev.ru/php/tutorial/db/mysql/#_6)

Выражение INSERT INTO используется для добавления новых записей в таблицу базы данных.

Составим корректный SQL-запрос на вставку записи с именем города, а затем выполним его путём передачи этого запроса в функцию mysqli\_query(), чтобы добавить новые данные в таблицу.

<?php

$link = mysqli\_connect("localhost", "root", "");

$sql = 'INSERT INTO cities SET name = "Санкт-Петербург"';

$result = mysqli\_query($link, $sql);

if ($result == false) {

print("Произошла ошибка при выполнении запроса");

}

Обратите внимание, что первым параметром для функциии mysqli\_query() передаётся ресурс подключения, полученный от функции mysqli\_connect(), вторым параметром следует строка с SQL-запросом.

При запросах на изменение данных (не SELECT) результатом выполнения будет логическое значение — true или false.

false будет означать, что запрос выполнить не удалось. Для получения строки с описанием ошибки существует функция mysqli\_error($link).

Функция insert id: как получить идентификатор добавленной записи[¶](https://xsltdev.ru/php/tutorial/db/mysql/#insert-id)

**Пример:**

Города хранятся в таблице cities

Записи погоде хранит таблица weather\_log

Добавим погодную запись для нового города.

Чтобы сослаться на город, необходимо знать идентификатор записи из таблицы cities. Здесь пригодится функция mysqli\_insert\_id().

Она принимает единственный аргумент — ресурс соединения, а возвращает идентификатор последней добавленной записи.

Вот как будет выглядеть комплексный пример с подключением к MySQL и добавлением двух новых записей:

<?php

$link = mysqli\_connect("localhost", "root", "");

if ($link == false){

print("Ошибка: Невозможно подключиться к MySQL " . mysqli\_connect\_error());

}

else {

$sql = 'INSERT INTO cities SET name = "Санкт-Петербург"';

$result = mysqli\_query($link, $sql);

if ($result == false) {

print("Произошла ошибка при выполнении запроса");

}

else {

$city\_id = mysqli\_insert\_id($link);

$sql = 'INSERT INTO weather\_log SET city\_id = ' . $city\_id . ', day = "2017-09-03", temperature = 10, cloud = 1';

$result = mysqli\_query($link, $sql);

if ($result == false) {

print("Произошла ошибка при выполнении запроса");

}

}

}

*Чтение записей*[*¶*](https://xsltdev.ru/php/tutorial/db/mysql/#_7)

Другая частая операция при работе с базами данных в PHP — это получение записей из таблиц (запросы типа SELECT).

Составим SQL-запрос, который будет использовать SELECT выражение. Затем выполним этот запрос с помощью функции mysqli\_query(), чтобы получить данные из таблицы.

В этом примере показано, как вывести все существующие города из таблицы cities:

<?php

$sql = 'SELECT id, name FROM cities';

$result = mysqli\_query($link, $sql);

while ($row = mysqli\_fetch\_array($result)) {

print("Город: " . $row['name'] . "; Идентификатор: . " . $row['id'] . "<br>");

}

В примере выше результат выполнения функции mysqli\_query() сохранён в переменной $result.

Важно понимать, что в этой переменной находятся не данные из таблицы, а специальный тип данных — так называемая ссылка на результаты запроса.

Чтобы получить действительные данные, то есть записи из таблицы, следует использовать другую функцию — mysqli\_fetch\_array() — и передать ей единственным параметром эту самую ссылку.

Теперь каждый вызов функции mysqli\_fetch\_array() будет возвращать следующую запись из всего результирующего набора записей в виде ассоциативного массива.

Цикл while здесь используется для «прохода» по всем записям из полученного набора записей. Значение поля каждой записи можно узнать просто обратившись по ключу этого ассоциативного массива.

*Как получить сразу все записи в виде двумерного массива*[*¶*](https://xsltdev.ru/php/tutorial/db/mysql/#_8)

Иногда бывает удобно после запроса на чтение не вызывать в цикле mysqli\_fetch\_array для извлечения очередной записи по порядку, а получить их сразу все одним вызовом. PHP так тоже умеет. Функция mysqli\_fetch\_all($res, MYSQLI\_ASSOC) вернёт двумерный массив со всеми записями из результата последнего запроса.

Перепишем пример с показом существующих городов с её использованием:

<?php

$sql = 'SELECT id, name FROM cities';

$result = mysqli\_query($link, $sql);

$rows = mysqli\_fetch\_all($result, MYSQLI\_ASSOC)

foreach ($rows as $row) {

print("Город: " . $row['name'] . "; Идентификатор: . " . $row['id'] . "<br>");

}

Защита от SQL инъекций

**Внедрение SQL-кода (SQL инъекция)** — один из распространённых способов взлома сайтов, работающих с базами данных. Способ основан на внедрении в запрос произвольного SQL-кода.

Внедрение SQL позволяет хакеру выполнить произвольный запрос к базе данных (прочитать содержимое любых таблиц, удалить, изменить или добавить данные).

Атака этого типа возможна, когда недостаточно фильтруются входные данные при использовании в SQL-запросах.

## Принцип атаки внедрения SQL[¶](https://xsltdev.ru/php/tutorial/db/injections/#sql_1)

Допустим, на нашем сайте есть страница показа истории погодных наблюдений для одного города. Идентификатор этого города передаётся в ссылке в параметре запроса: /weather.php?city\_id=<ID>, где ID — это первичный ключ города.

В PHP-сценарии используем этот параметр для подстановки в SQL запрос:

<?php

$city\_id = $\_GET['city\_id'];

$res = mysqli\_query($link, "SELECT \* FROM weather\_log WHERE city\_id = " . $city\_id);

Если на сервере передан параметр city\_id, равный 10 (/weather.php?city\_id=10), то выполнится SQL-запрос:

SELECT \* FROM weather\_log WHERE city\_id = 10

Но если злоумышленник передаст в качестве параметра id строку -1 OR 1=1, то выполнится запрос:

SELECT \* FROM weather\_log WHERE city\_id = -1 OR 1=1

Добавление во входные параметры конструкций языка SQL (вместо простых значений) изменяет логику выполнения всего SQL запроса!

В этом примере вместо показа данных по одному городу, будут получены данные по всем городам, потому что выражение 1=1 всегда истинно. Вместо выражения SELECT ... могло быть выражение на обновление данных, и тогда последствия были бы ещё серьезнее.

Отсутствие должной обработки параметров SQL-запроса — это одна из самых серьёзных уязвимостей. Никогда не вставляйте данные от пользователя в SQL запросы «как есть»!

## Приведение к целочисленному типу[¶](https://xsltdev.ru/php/tutorial/db/injections/#_1)

В SQL-запросы часто подставляются целочисленные значения, полученные от пользователя. В примерах выше использовался идентификатор города, полученный из параметров запроса. Этот идентификатор можно принудительно привести к числу. Так мы исключим появление в нём опасных выражений. Если хакер передаст в этом параметре вместо числа SQL код, то результатом приведения будет ноль, и логика всего SQL-запроса не изменится.

PHP умеет присваивать переменной новый тип. Этот код принудительно назначит переменной целочисленный тип:

<?php

$city\_id = $\_GET['city\_id'];

settype($city\_id, 'integer');

После преобразования переменную $city\_id можно без опаски использовать в SQL-запросах.

## Экранирование значений[¶](https://xsltdev.ru/php/tutorial/db/injections/#_2)

Что делать, если в SQL запрос требуется подставить строковое значение? Например, на сайте есть возможность поиска города по его названию. Форма поиска передаст поисковый запрос в GET-параметр, а мы используем его в SQL-запросе:

<?php

$city\_name = $\_GET['search'];

$sql = "SELECT \* FROM cities WHERE name LIKE('%$city\_name%')";

Но если в параметре city\_name будет символ кавычки, то смысл запроса можно кардинально изменить. Передав в search\_text значение ')+and+(id<>'0, мы выполним запрос, что выведет список всех городов:

SELECT \* FROM cities WHERE name LIKE('%') AND (id<>'0%'))

Смысл запроса поменялся, потому что кавычка из параметра запроса считается управляющим символом: MySQL определяет окончание значение по символу кавычки после него, поэтому сами значения кавычки содержать не должны.

Очевидно, приведение к числовому типу не подходит для строковых значений. Поэтому, чтобы обезопасить строковое значение, используют операцию экранирования.

Экранирование добавляет в строке перед кавычками (и другими спецсимволами) знак апострофа \. Такая обработка лишает кавычки их статуса — они больше не определяют конец значения и не могут повлиять на логику SQL-выражения.

За экранирование значений отвечает функция mysqli\_real\_escape\_string().

Этот код обработает значение из параметра, сделав его безопасным для использования в запросе:

<?php

$city\_name = mysqli\_real\_escape\_string($link, $\_GET['search']);

$sql = "SELECT \* FROM cities WHERE name LIKE('%$city\_name%')";

## Подготовленные выражения[¶](https://xsltdev.ru/php/tutorial/db/injections/#_3)

Вид атак типа «SQL-инъекция» возможен, потому что значения (данные) для SQL-запроса передаются вместе с самим запросом. Так как данные не отделены от SQL-кода, они могут влиять на логику всего выражения. К счастью, MySQL предлагает способ передачи данных отдельно от кода. Такой способ называется подготовленными запросами.

Выполнение подготовленных запросов состоит из двух этапов: вначале формируется шаблон запроса — обычное SQL-выражение, но без действительных значений, а затем, отдельно, в MySQL передаются значения для этого шаблона.

Первый этап называется подготовкой, а второй — выражением. Подготовленный запрос можно выполнять несколько раз, передавая туда разные значения.

### Этап подготовки[¶](https://xsltdev.ru/php/tutorial/db/injections/#_4)

На этапе подготовки формируется SQL-запрос, где на месте значений будут находиться знаки вопроса — плейсхолдеры. Эти плейсхолдеры в дальнейшем будут заменены на реальные значения. Шаблон запроса отправляется на сервер MySQL для анализа и синтаксической проверки.

Пример:

<?php

$sql = "SELECT \* FROM cities WHERE name = ?";

$stmt = mysqli\_prepare($link, $sql);

Этот код сформирует подготовленное выражение для выполнения вашего запроса.

За подготовкой идёт выполнение. Во время запуска запроса PHP привязывает к плейсхолдерам реальные значения и посылает их на сервер. За передачу значений в подготовленный запрос отвечает функция mysqli\_stmt\_bind\_param(). Она принимает тип и сами переменные:

<?php

mysqli\_stmt\_bind\_param($stmt, 's', $\_GET['search']);

После выполнения запроса получить его результат в формате mysqli\_result можно функцией mysqli\_stmt\_get\_result():

<?php

$res = mysqli\_stmt\_get\_result($stmt);

// чтение данных

while ($row = mysqli\_fetch\_assoc($res)) {

// ассоциативный массив с очередной записью из результата

var\_dump($row);

}

Значения привязанных к запросу переменных сервер экранирует автоматически. Привязанные переменные отправляются на сервер отдельно от запроса и не могут влиять на него. Сервер использует эти значения непосредственно в момент выполнения, уже после того, как был обработан шаблон выражения. Привязанные параметры не нуждаются в экранировании, так как они никогда не подставляются непосредственно в строку запроса.