

МДК 02.02. Взаимодействие PHP и MySQL

База данных – это совокупность связанных между собой таблиц.

SQL – это структурированный язык запросов, созданный для управления реляционными БД. Позволяет пользователям взаимодействовать с базами данных.

MySQL – это одна из наиболее популярных и эффективных систем управления базами данных, которая используется при построении современных веб-сайтов.

MySQL – это сервер баз данных.

Основы языка SQL

Группы операторов языка SQL

1. Операторы описания данных: **CREATE, ALTER, DROP** и др.
2. Операторы манипуляции данными: **INSERT, DELETE, SELECT, UPDATE** и др.
3. Операторы задания прав доступа в базе данных.
4. Операторы защиты, восстановления данных и прочие операторы.

Создание базы данных

Синтаксис оператора

CREATE DATABASE имя_БД

Создание таблицы

Оператор **CREATE TABLE** предназначен для описания структуры таблицы.

Синтаксис оператора

```
CREATE TABLE имя_таблицы (  
имя_столбца тип_данных [NOT NULL|NULL]  
[AUTO_INCREMENT]  
имя_столбца тип_данных [NOT NULL|NULL]  
[AUTO_INCREMENT]  
...  
PRIMARY KEY (имя_столбца)  
KEY (имя_индекса|имя_столбца)  
INDEX (имя_индекса|имя_столбца)  
)
```

NOT NULL|NULL – запрещает|разрешает в таблице пустые ячейки в данном столбце

AUTO_INCREMENT – устанавливает столбец, как поле с автонумерацией

PRIMARY KEY – описывает первичный ключ

KEY – описывает внешний ключ

INDEX – описывает индекс

Первичный ключ (primary key, PK) – это уникальный индекс, который применяется для уникальной идентификации записей таблицы. Никакие из двух записей таблицы не могут иметь одинаковых значений первичного ключа.

Внешний ключ (foreign key, FK) является ссылкой на первичный ключ, устанавливая однозначную логическую связь между записями таблиц. Важная часть механизма обеспечения ссылочной целостности данных.

Типы данных

Числовые типы

- ✓ **TINYINT** – 1 байт;
- ✓ **SMALLINT** – 2 байта;
- ✓ **MEDIUMINT** – 3 байт;
- ✓ **INT** – 4 байта;
- ✓ **BIGINT** – 8 байт;
- ✓ **DECIMAL** – с фиксированной точкой;
- ✓ **FLOAT** – с плавающей точкой.

Символьные типы

- ✓ **CHAR** – строка символов фиксированной длины;
- ✓ **VARCHAR** – строка символов переменной длины.

Типы даты и времени

- ✓ **DATE** – дата;
- ✓ **TIME** – время;
- ✓ **DATETIME** и другие.

Пример создания таблицы student в базе данных stud

```
CREATE TABLE `stud`.`student` (  
  `id_stud` INT(3) NOT NULL AUTO_INCREMENT ,  
  `fam` VARCHAR(30) NOT NULL ,  
  `name` VARCHAR(30) NOT NULL ,  
  `age` INT(2) NOT NULL , `id_group` INT(3) NOT  
  NULL ,  
  PRIMARY KEY (`id_stud`),  
  INDEX (`id_group`)) ENGINE = InnoDB;
```

Модификация таблицы

Оператор **ALTER TABLE** – используется для добавления, изменения или удаления столбцов в таблице.

Синтаксис оператора

ALTER TABLE имя_таблицы спецификация

Спецификация оператора ALTER TABLE

ADD имя_столбца [FIRST AFTER имя_столбца]	Добавление нового столбца.
ADD INDEX [имя_индекса] (имя_столбца,...)	Добавление индекса для столбца
ADD PRIMARY KEY (имя_столбца,...)	Установка столбца или группы столбцов первичным ключом таблицы

Спецификация оператора ALTER TABLE

CHANGE имя_столбца новое_имя_столбца type	Изменение столбца на другой столбец с указанным именем и типом type
DROP COLUMN имя_столбца	Удаление столбца с указанным именем
DROP PRIMARY KEY	Удаление первичного ключа таблицы
DROP INDEX имя_индекса	Удаление индекса

Удаление таблицы

Оператор **DROP TABLE** позволяет удалить одну или несколько таблиц из базы данных.

Синтаксис оператора

DROP TABLE имя_таблицы1, имя_таблицы2,...;

Операторы манипуляции данными (MySQL запросы)

Добавление записей в таблицу

Оператор **INSERT** используется для вставки одной записи или несколько записей в таблицу

Синтаксис оператора

INSERT INTO имя_таблицы

(имя_столбца1, имя_столбца2, ...)

VALUES

(значение1, значение2, ...),

...;

Пример

```
INSERT INTO proizvod  
(id, name)  
VALUES  
(14, 'Acer');
```


Удаление записей из таблицы

Оператор **DELETE** используется для удаления одной записи или нескольких записей из таблицы в MySQL.

Синтаксис оператора

```
DELETE FROM имя_таблицы  
[WHERE условие];
```

Пример

```
DELETE FROM proizvod  
WHERE name = 'Acer';
```

Обновление записей в таблице

Оператор **UPDATE** используется для обновления существующих записей в таблице в базе данных MySQL.

Синтаксис оператора

UPDATE имя_таблицы

SET имя_столбца1 = значение1,

имя_столбца2 = значение2,

...

[WHERE условие];

Пример 1 Обновление одного столбца

```
UPDATE proizvod  
SET name = 'Acer'  
WHERE id = 12;
```

Пример 2 Обновление нескольких столбцов

```
UPDATE customers  
SET state = 'Nevada',  
    customer_rep = 23  
WHERE customer_id > 200;
```

Пример 3 Обновление нескольких таблиц

```
UPDATE customers, suppliers  
SET customers.city = suppliers.city  
WHERE customers.customer_id =  
suppliers.supplier_id;
```

Выборка записей из таблиц

Оператор **SELECT** используется для извлечения записей из одной или нескольких таблиц.

Синтаксис оператора

SELECT имя_столбца

FROM имя_таблицы

[**WHERE** условие];

Символ ***** используется для выбора всех столбцов из таблицы.

Необязательные операторы

Оператор **ORDER BY** используется в SELECT для сортировки записей в результирующем наборе.

Синтаксис оператора

ORDER BY имя_столбца [ASC | DESC];

- ✓ **ASC** – сортирует результирующий набор в порядке возрастания (по умолчанию, если атрибут не указан);
- ✓ **DESC** – сортирует результирующий набор в порядке убывания.

Оператор GROUP BY используется в SELECT предложении для сбора данных по нескольким записям и группировки результатов по одному или нескольким столбцам.

Синтаксис оператора

GROUP BY имя_столбца1, имя_столбца2, ...;

Использование функций в запросах

Функция **SUM** – определяет сумму значений поля.

Функция **COUNT** – определяет количество записей.

Функция **MIN** – определяет минимальное значение.

Функция **MAX** – определяет максимальное значение.

Функция **AVG** – определяет среднее значение.

Пример 1 Выборка всех полей из одной таблицы

```
SELECT *  
FROM order_details  
WHERE quantity >= 100  
ORDER BY quantity DESC;
```

Пример 2 Выборка отдельных полей из одной таблицы

```
SELECT order_id, quantity, unit_price
```

```
FROM order_details
```

```
WHERE quantity < 300
```

```
ORDER BY quantity ASC, unit_price DESC;
```

Пример 3 Выборка полей из нескольких таблиц

```
SELECT order_details.order_id,  
customers.customer_name  
FROM customers  
INNER JOIN order_details  
ON customers.customer_id = order_details.customer_id  
ORDER BY order_details.order_id;
```

В SELECT указывается **имя_таблицы.имя_столбца**

Оператор JOINS используется для извлечения данных из нескольких таблиц.

MySQL условия

AND	логический оператор и
OR	логический оператор или
LIKE NOT LIKE	<p>позволяет использовать шаблоны в операторе WHERE для операторов SELECT, INSERT, UPDATE или DELETE.</p> <p>% – позволяет сопоставлять любую строку любой длины (например, 'Ver%', '%ns%')</p> <p>_ – позволяет сопоставлять один символ (например, 'Ver_ard', '123_')</p>


MySQL условия

IN	позволяет проверить, соответствует ли какое-либо одно из значений выражению IN (value1, value2, value n)
NOT	используется для отрицания условия в операторах SELECT, INSERT, UPDATE или DELETE
BETWEEN	используется для извлечения значений внутри диапазона в операторе SELECT, INSERT, UPDATE или DELETE (например, WHERE contact_id BETWEEN 50 AND 100;)

Операторы сравнения MySQL

=, <=>, <>, !=, >, >=, <, <=.

Использование PhpMyAdmin для взаимодействия с базой данных MySQL



Добро пожаловать в phpMyAdmin

Язык - *Language*

Русский - Russian ▼

Авторизация ⓘ

Пользователь:

Пароль:

Вперёд

Библиотека **php_mysql.dll** предоставляет современные методы доступа к базе данных MySQL

Установка соединения

1. `$db = mysql_connect(<Имя хоста>, <Имя пользователя>, <Пароль>, <База данных>);`
2. `$db = new mysqli(<Имя хоста>, <Имя пользователя>, <Пароль>, <База данных>);`

Пример

```
if (@$db = mysqli_connect("localhost", "root",  
"123456", "tests")) {  
// Выполняем работу с базой данных  
}  
else {  
echo "Не удалось установить подключение к базе  
данных";  
}
```

@ - подавляет вывод ошибки функции

Функция **mysqli_connect_errno()** проверяет отсутствие ошибок при подключении

```
@$db = new mysqli("localhost", "root", "123456",  
"tests");
```

```
if (!mysqli_connect_errno()) {
```

```
// Выполняем работу с базой данных
```

```
}
```

```
else {
```

```
echo "Не удалось установить подключение к базе  
данных";
```

```
}
```

Заккрытие соединения

Процедурный стиль

Функция `mysqli_close()`:

```
mysqli_close(<Идентификатор>);
```

Объектный стиль

Используется метод `close()`:

```
<Экземпляр класса>->close();
```

Процедурный стиль

```
if (@$db = mysqli_connect("localhost", "root",  
"123456", "tests")) {  
// Выполняем работу с базой данных  
mysqli_close($db); // Закрываем соединение  
}  
else {  
echo "Не удалось установить подключение к базе  
данных";  
}
```

Объектный стиль

```
@$db = new mysqli("localhost", "root", "123456",  
"tests");  
if (!mysqli_connect_errno()) {  
// Выполняем работу с базой данных  
$db->close(); // Закрываем соединение  
}  
else {  
echo "Не удалось установить подключение к базе  
данных";  
}
```

Выбор базы данных

Функция **mysqli_select_db()** служит для выбора базы данных уже после подключения

Формат

mysqli_select_db(<Идентификатор>, <Имя базы данных>);

Пример

```
if (@$db = mysqli_connect("localhost", "root",  
"123456")) {  
    mysqli_select_db($db, "tests");  
    // Выполняем работу с базой данных  
    mysqli_close($db);  
}  
else {  
    echo "Не удалось установить подключение к базе  
данных";  
}
```

При объектном стиле используется метод
select_db().

Формат:

<Экземпляр класса>->**select_db**(<Имя базы данных>);

Пример

```
@$db = new mysqli("localhost", "root", "123456");  
if (!mysqli_connect_errno()) {  
    $db->select_db("tests");  
    // Выполняем работу с базой данных  
    $db->close();  
}  
else {  
    echo "Не удалось установить подключение к базе  
    данных";  
}
```

Выполнение запроса к базе данных

Выполнить запрос к базе данных в процедурном стиле позволяет функция **mysqli_query()**.

Функция имеет следующий формат:

```
mysqli_query(<Идентификатор>, <SQL-запрос>);
```

Для удаления идентификатора результата и освобождения используемых ресурсов применяется функция **mysql_free_result()**.

Формат:

mysql_free_result(<Идентификатор результата>);

Выполнить запрос к базе данных при объектном стиле позволяет метод `query()`.

Формат:

`<Экземпляр класса>->query(<SQL-запрос>);`

Метод возвращает экземпляр результата.

Для удаления экземпляра результата применяется метод `close()`.

Формат:

`<Экземпляр результата>->close();`

Обработка результата запроса

Процедурный стиль

1. **mysqli_num_rows**(<Идентификатор результата>) возвращает количество записей в результате
2. **mysqli_field_count**(<Идентификатор соединения>) возвращает количество полей в результате последнего SQL-запроса
3. **mysqli_fetch_array**(<Идентификатор результата>, [<Флаг>]) возвращает результат в виде списка и (или) ассоциативного массива

Параметр **Флаг** может принимать следующие значения:

- **MYSQLI_BOTH** – результат в виде списка и ассоциативного массива (значение по умолчанию);
 - **MYSQLI_NUM** – результат в виде списка;
 - **MYSQLI_ASSOC** – результат в виде ассоциативного массива.
- 4. `mysqli_fetch_row`(**<Идентификатор результата>**)** возвращает результат в виде списка
- 5. `mysqli_fetch_assoc`(**<Идентификатор результата>**)** возвращает результат в виде ассоциативного массива

6. **mysql_fetch_object**(<Идентификатор результата>) возвращает результат в виде объекта
7. **mysql_data_seek**(<Идентификатор результата>, <Смещение>) перемещает указатель результата на выбранную строку. Нумерация начинается с нуля

Объектный стиль

1. **num_rows** возвращает количество записей в результате
2. **field_count** возвращает количество полей в результате
3. **fetch_array([<Флаг>])** возвращает результат в виде списка и (или) ассоциативного массива в зависимости от значения необязательного параметра <Флаг>.

Параметр **Флаг** может принимать следующие значения:

- **MYSQLI_BOTH** – результат в виде списка и ассоциативного массива (значение по умолчанию);
- **MYSQLI_NUM** – результат в виде списка;
- **MYSQLI_ASSOC** – результат в виде ассоциативного массива

4. **fetch_row()** возвращает результат в виде списка
5. **fetch_assoc()** возвращает результат в виде ассоциативного массива
6. **fetch_object()** возвращает результат в виде объекта
7. **data_seek(<Смещение>)** перемещает указатель результата на выбранную строку. Нумерация начинается с нуля