

Урок №1

Работа с базой данных.

В этом уроке

- Вы узнаете, чем отличаются термины «БД» и «СУБД»;
- что такое реляционная база данных, и из чего она состоит;
- что такое нормальные формы;
- какие существуют типы связей между данными;
- как организовывать связи между сущностями;
- познакомитесь с языком баз данных SQL;
- научитесь работать с MySQL средствами PHP.

1. Отличие «БД» от «СУБД»

БД – база данных. Под этим термином понимается информация, которую вы храните.

СУБД – система управления базой данных. Это программа, которая предоставляет доступ внешним приложениям к базе данных, обеспечивает ее работу.

Существуют различные популярные СУБД: Oracle, Microsoft SQL Server, MySQL, Sybase, PostgreSQL и т.д.

Сайты PHP чаще всего работают в связке с MySQL, поэтому именно данную СУБД мы будем рассматривать в текущем уроке.

2. Реляционная база данных

Существует множество различных способов организации информации в БД. Основные типы баз данных:

- Иерархические
- Объектные
- Сетевые
- Объектно-реляционные
- Функциональные
- Реляционные

Нас интересует последний тип, так как именно с ним придётся работать при создании сайтов.

Реляционная база данных состоит из **таблиц** и **связей** между ними. Как известно из жизни, любая таблица состоит из столбцов и строк. Столбцов создаётся определённое количество, а строк может быть сколь угодно много. Столбец является более важным элементом, чем строка, так как последняя отвечает лишь за конкретную запись в таблице, а столбец – за наличие определённой характеристики у всех записей.

3. Нормальные формы

При проектировании базы следует учитывать определённый набор правил, который называется нормальными формами. Ниже будет приведено их краткое описание. С научными формулировками нормальных форм Вы можете ознакомиться на сайте Википедии: http://ru.wikipedia.org/wiki/Нормальные_формы

Первая нормальная форма – атомарность данных.

В одной ячейке (пересечение столбца и строки) должно записываться не более одного значения. Например, ошибкой является следующая организация хранения номеров телефонов сотрудников:

id_emp	last_name	content
1	Иванов	391-88-16,562-08-17
2	Петров	376-34-22

При таком подходе будут затруднены операции поиска и изменения информации. Правильное решение – создание отдельной таблицы с телефонными номерами.

Вторая нормальная форма – неключевой атрибут не должен функционально зависеть от потенциального ключа.

Сразу рассмотрим пример:

Сотрудник	Должность	Зарплата	Наличие компьютера
Иванов	Программист	50000	Есть
Петров	Кладовщик	30000	Нет
Сидоров	Программист	40000	Есть

Попробуем разглядеть функциональные зависимости. Как мы видим, наличие компьютера явно зависит от должности, т.е., неключевой атрибут функционально зависит от потенциального ключа. Теперь разберёмся, почему же это плохая организация.

Например, представим, что Петров отучился на курсах по программированию и меняет профессию. При изменении должности нужно будет также не забыть поменять наличие компьютера. Это неудобно. Организация в базе должна быть такой, чтобы мы исправляли значение только одного атрибута, а остальные автоматически менялись благодаря связям в БД. Правильное решение – создание отдельной таблицы, в которой будут храниться пары «должность – наличие компьютера».

Третья нормальная форма – отсутствие транзитивных зависимостей.

Транзитивной является зависимость вида $\{A\} \rightarrow \{B\} \rightarrow \{C\}$. Если при создании таблицы возникает подобная зависимость, её необходимо убирать с помощью разделения данных на две таблицы, которые, соответственно, будут содержать связи $\{A\} \rightarrow \{B\}$ и $\{B\} \rightarrow \{C\}$.

4. Примеры организации связей в БД

Всего существуют три типа связей между сущностями:

- один к одному;
- один ко многим;
- многие ко многим.

Рассмотрим примеры организации различных типов связей при создании таблиц в базе.

1. Один к одному

Связь «один к одному» настолько очевидна, что мы о ней практически никогда не задумываемся. Например, имя и фамилия человека связана «один к одному». В базе они хранятся в одной таблице, поэтому связь не видна. Однако, бывают ситуации, когда такую связь имеет смысл показывать с помощью отдельного поля. Например, мы создаём таблицу сотрудников фирмы и хотим хранить отметки о семейных парах внутри фирмы. Тогда таблица может выглядеть следующим образом:

id_emp	first_name	middle_name	last_name	id_pair
1	Иван	Иванович	Иванов	4
2	Петр	Петрович	Петров	5
3	Павел	Павлович	Павлов	NULL
4	Елена	Ивановна	Иванова	1
5	Елена	Петровна	Петрова	2

В поле **id_pair** записывается номер сотрудника, который является мужем либо женой для него. У сотрудника №3 нет пары в фирме, поэтому **id_pair = NULL**, т.е. отсутствует.

2. Один ко многим

Связь «один ко многим» является наиболее распространённой. Например, мы делаем блог, в котором к новостям (статьям) можно писать комментарии. У одной статьи может быть много комментариев, но в свою очередь один комментарий относится ровно к одной новости. В БД создаём две таблицы:

articles

id_article	title	content
1	Первая статья	Какой-то текст
2	Вторая статья	Какой-то текст

comments

id_comment	id_article	name	text
1	1	Иван	Ну и бред!
2	1	Петр	Хорошая статья!
3	2	Павел	Есть о чём подумать!

Связь между сущностями здесь организуется по полю `id_article`. В таблице `comments` это поле показывает на то, какой статье принадлежит данный комментарий.

3. Многие ко многим

Это самый сложный тип связи для организации в базе, потому что он обычно требует создания трёх таблиц. Вспомните, наверняка, путешествуя по блогам Интернета, Вы замечали, что иногда у статей появляются метки – рубрики, к которым данные статьи принадлежат. По метке можно кликнуть и перейти ко всем статьям по указанной тематике.

Получается, что у одной статьи может быть несколько меток, а одной метке также могут принадлежать несколько статей. Действительно, здесь связь – «многие ко многим». Рассмотрим её организацию в БД:

articles

id_article	title	content
1	РФС. Увольнение клубных агентов.	Какой-то текст
2	Четвертьфинал лиги чемпионов	Какой-то текст

notes

id_note	name
1	Футбол
2	Политика

articles_notes

id_article	id_note
1	1
1	2
2	2

В связи «многие ко многим» ключевую роль играет перекрёстная таблица **articles_notes**. Рассмотрим её подробнее. Итак, у нас было две метки и две статьи. Первая статья относится и к рубрике «футбол», и к рубрике «политика», в то время, как вторая – только к рубрике «футбол». В таблице **articles_notes** создаются три записи, которые и обозначают пары соответствий.

5. Язык SQL

SQL (Structured Query Language - «язык структурированных запросов») - универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных.

То есть, SQL – это язык для общения с базой данных. Любую операцию, от создания таблицы до выборки данных, можно осуществить только посредством запроса на языке SQL.

Все запросы делятся на два вида:

- DDL
- DML

К DDL (Data Definition Language – «язык описания данных») относятся запросы, меняющие структуру базы данных. Например, создание таблицы, удаление таблицы, добавление столбца к существующей таблице.

К DML (Data Modification Language – «язык модификации данных») относятся запросы, меняющие содержимое базы данных, то есть, операции над строками таблиц. Сюда относится вставка, удаление, изменение и выборка строк.

Мы сосредоточимся на изучении именно DML-запросов, так как они осуществляются гораздо чаще. По сути, DDL-операции не должны осуществляться из PHP скрипта, так как структура базы данных должна быть заранее продумана и сделана один раз при её создании.

Пример DDL-операции – создание таблицы **depts**:

```
CREATE TABLE depts
(
    id_dept INT NOT NULL,
    name VARCHAR(32) NOT NULL,
    PRIMARY KEY (id_dept)
)
```

Обычно данные команды не прописываются вручную, а производятся с помощью специальной утилиты phpMyAdmin.

Если Вы используете denwer, то можете открыть её по следующему адресу:
<http://localhost/tools/phpMyAdmin>

Если Вы никогда не работали с phpMyAdmin, обязательно посмотрите видео:

http://youtu.be/WHSzVOv3_Eo

Теперь перейдём к рассмотрению DML-операций.

5.1 Вставка строк

Для вставки строк в языке SQL служит оператор **INSERT**. Вот так можно наполнить базу данных сотрудников и отделов:

```
INSERT INTO depts (id_dept, name) VALUES ('1', 'Бухгалтерия');
INSERT INTO depts (id_dept, name) VALUES ('2', 'ИТ');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
VALUES ('1', '2', 'Иван', 'Иванович', 'Иванов');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
VALUES ('2', '1', 'Петр', 'Петрович', 'Петров');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
VALUES ('3', '2', 'Павел', 'Павлович', 'Павлов');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
VALUES ('4', '2', 'Елена', 'Ивановна', 'Иванова');

INSERT INTO emps (id_emp, id_dept, last_name, first_name, middle_name)
VALUES ('5', '1', 'Елена', 'Петровна', 'Петрова');
```

5.2 Удаление строк

Предположим, руководство решило уволить всех сотрудников отдела маркетинга. В этом случае поможет оператор **DELETE**, удаляющий строки из таблицы:

```
DELETE FROM emps  
WHERE id_dept = '2'
```

Увольнение конкретного сотрудника выглядит похожим образом:

```
DELETE FROM emps  
WHERE id_emp = '2'
```

Обратите внимание, запись о сотруднике мы удаляем, используя не его фамилию, а первичный ключ. Связано это с тем, что однофамильцы могут присутствовать в фирме, а первичный ключ всегда уникален.

5.3 Изменение строк

Елена Петровна вышла замуж за Павла Павловича и поменяла фамилию. Для изменения строк таблицы служит оператор **UPDATE**:

```
UPDATE emps  
SET last_name = 'Иванова'  
WHERE id_emp = '5'
```

Главное, не забывать указывать при изменении строк параметр WHERE. В противном случае, при отсутствии WHERE в данном примере, фамилии поменялись бы у всех сотрудников.

5.4 Выборка строк

За выборку строк отвечает оператор SELECT. С его помощью можно составлять сложнейшие запросы, выбирающие данные сразу из множества таблиц. В этом уроке нас интересуют лишь самые простые примеры.

Все сотрудники:

```
SELECT *  
FROM emps
```

Сотрудники ИТ-отдела:

```
SELECT *  
FROM emps  
WHERE id_dept = '1'
```

Сколько всего в фирме работает человек?

```
SELECT count(*)  
FROM emps;
```

Сколько сотрудников в ИТ-отделе?

```
SELECT count(*)
  FROM emps
 WHERE id_dept = '2';
```

Сотрудники отдела бухгалтерии, отсортированные по фамилии, имени, отчеству:

```
SELECT *
  FROM emps
 WHERE id_dept = '1'
 ORDER BY last_name, first_name, middle_name ASC
```

Сотрудники отдела бухгалтерии, отсортированные в обратном порядке:

```
SELECT *
  FROM emps
 WHERE id_dept = '1'
 ORDER BY last_name, first_name, middle_name DESC
```

На данном этапе мы рассмотрели основные DML-операции. Очень важно, что мы видели синтаксис именно языка SQL, т.е., в PHP-коде нельзя просто брать и писать подобные команды. Для этого нужно применять специальные функции, которые мы рассмотрим в следующем разделе.

6. Основные функции PHP для работы с MySQL

Порядок работы с базой данных в PHP похож на порядок работы с файлами, но имеет определённые особенности:

- 1) установить соединение с сервером;
- 2) выбрать конкретную базу данных;
- 3) выполнить операции;
- 4) закрыть соединение.

Рассмотрим по очереди все эти шаги. Для подключения к базе в PHP существует специальная функция **mysql_connect**. Чаще всего она вызывается с тремя параметрами:

```
mysql_connect($server, $username, $password);
```

Если вы тестируете сайт на локальном компьютере, и у вас установлен пакет Денвер, то параметры должны быть следующими:

```
$server = 'localhost'; // Адрес сервера, на котором находится база
$username = 'root';    // Имя пользователя
$password = '' ;       // Пароль
```

После установки соединения с сервером необходимо выбрать конкретную базу данных. Для этого служит функция **mysql_select_db**:

```
mysql_select_db ($db_name);
```

где \$db_name – имя базы, с которой Вы хотите работать.

Теперь можно совершать действия, которые мы рассматривали в предыдущем пункте. За выполнение запроса к базе отвечает функция **mysql_query**, которая принимает строку с командой на языке SQL и возвращает результат её выполнения. Например:

```
$result = mysql_query("DELETE FROM emps WHERE id_emp = '2'");
```

Для запросов, не подразумевающих получение данных из базы (INSERT, UPDATE, DELETE) функция возвращает true в случае успешного выполнения операции и false в противном случае.

Для SELECT-запросов функция возвращает дескриптор с результатом выборки в случае успешного выполнения операции и false в противном случае.

Получается, что после выполнения запроса на выборку данных, нам необходимо его обработать. Как правило, это представляет собой перегонку системного типа данных в массивы, например:

```
$result = mysql_query('SELECT * FROM emps');
$emps = array();

while($row = mysql_fetch_assoc($result))
    $emps[] = $row;
```

Внимательно разберёмся с данным кодом. В переменной \$result оказывается результат выборки из базы - список всех сотрудников.

Однако, пока что он хранится в непригодном для нас виде. Поэтому создаём пустой массив \$emps, в который будем добавлять информацию о каждой записи.

Теперь запускаем цикл, который ориентируется на функцию **mysql_fetch_assoc**. Она извлекает очередную строку из выборки данных и возвращает её в переменную \$row в виде ассоциативного массива. Ключами данного массива являются названия столбцов таблицы, а значениями – данные из конкретной строки. Когда строки закончатся, **mysql_fetch_assoc** вернёт значение false, и цикл завершится.

На выходе мы получим двумерный массив \$emps с информацией о сотрудниках. В нём будет столько же элементов, сколько строк вернула база в результате выборки. Каждый его элемент – ассоциативный массив с информацией об одном сотруднике.

Данная перегонка является стандартной. Её достаточно освоить один раз, а затем лишь применять на практике.

Также существуют и другие полезные функции при работе с базой, например:

mysql_num_rows – число строк, содержащееся в результате выборки данных;

mysql_affected_rows – число строк, затронутых последним запросом INSERT, UPDATE или DELETE;

mysql_error – сообщение о последней ошибке, возникшей в ходе запроса;

mysql_insert_id – id записи, добавленной последним запросом INSERT;

mysql_close – закрывает соединение с сервером MySql.

Самоконтроль

- ✓ Зачем нужна база данных
- ✓ Чем БД отличается от СУБД
- ✓ Из чего состоит реляционная база данных
- ✓ Из чего состоят таблицы в БД
- ✓ Что в таблице важнее: столбец или строка
- ✓ Первая нормальная форма
- ✓ Вторая нормальная форма
- ✓ Третья нормальная форма
- ✓ В чём отличие DDL и DML операций
- ✓ Какие из вышеперечисленных операций совершаются чаще
- ✓ Как создать базу данных и таблицы через phpMyAdmin
- ✓ Вставка строк
- ✓ Удаление строк
- ✓ Изменение строк
- ✓ Выборка строк
- ✓ Порядок работы с базой данных в PHP

Домашнее задание

Создать базу и таблицу для хранения статей.

Написать пять простых скриптов:

- 1) index.php – выборка всех записей
- 2) article.php – выборка одной записи
- 3) add.php – добавление записи
- 4) delete.php – удаление одной записи
- 5) edit.php – изменений одной записи

В скриптах 2,4 и 5 номер статьи передаётся через GET-параметр.

Скрипты 3 и 5 не принимают название и содержание статьи от пользователя – оно просто жёстко прописано в коде.

Данные скрипты являются отдельными несвязанными между собой файлами.