Владимир Дронов

bhv®

Django: практика создания Web-сайтов на Python

- ЯЗЫК Python И БИБЛИОТЕКА Django
- МОДЕЛИ, КОНТРОЛЛЕРЫ И ШАБЛОНЫ
- ФОРМЫ И ВЫГРУЗКА ФАЙЛОВ
- РАЗГРАНИЧЕНИЕ ДОСТУПА И КОММЕНТИРОВАНИЕ
- ФОРМАТИРОВАНИЕ BBCode
- ОТПРАВКА ЭЛЕКТРОННОЙ ПОЧТЫ
- ВСТРОЕННЫЙ АДМИНИСТРАТИВНЫЙ САЙТ
- ПОЛНОФУНКЦИОНАЛЬНЫЙ САЙТ НА Ajax

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+ Материалы на www.bhv.ru

УДК 004.738.5+004.438Python
ББК 32.973.26-018.1
Д75

Дронов В. А.

Д75 Django: практика создания Web-сайтов на Python. — СПб.: БХВ-Петербург, 2016. — 528 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0421-8

Книга посвящена разработке Web-сайтов на популярном языке программирования Python с использованием библиотеки Django. Описывается создание моделей, контроллеров и шаблонов, применение форм для ввода данных и выгрузки на сайт файлов, реализация разграничения доступа, комментирование кода, работа со статичными страницами, применение сторонних библиотек для вывода миниатюр. Рассказывается о форматировании текста тегами BBCode, привязке к позициям тегов и выполнении поиска по тегам. Рассматриваются инструменты для генерирования каналов новостей RSS и Atom, рассылки электронной почты и настройка встроенного административного сайта Django под свои нужды. Детально описывается процесс разработки и публикации полнофункционального коммерческого Web-сайта, использующего, в том числе, технологию AJAX. Все исходные коды доступны для загрузки с сайта издательства.

Для широкого круга Web-программистов

УДК 004.738.5+004.438Python
ББК 32.973.26-018.1

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Редактор	*Григорий Добин*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Марины Дамбиевой*

Подписано в печать 31.08.15.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 42,57.
Тираж 1000 экз. Заказ №
"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

Первая Академическая типография "Наука"
199034, Санкт-Петербург, 9 линия, 12/28

ISBN 978-5-9775-0421-8

Оглавление

ГЛАВА 1

Введение в серверное Web-программирование

Не откладывая дела в долгий ящик, сразу же приступим к рассмотрению основных принципов серверного Web-программирования. Мы узнаем, что такое собственно серверные Web-приложения, как они работают, что есть базы данных, таблицы, поля, записи, индексы, модели, контроллеры и шаблоны. Без всего этого мы просто не поймем, о чем пойдет речь в следующих главах книги.

А начнем мы с того, что разберемся, в чем принципиальная разница между статичными Web-страницами и Web-приложениями, и рассмотрим их преимущества и недостатки.

Статичные Web-страницы и Web-приложения — две эпохи в развитии Интернета

В более чем тридцатилетней истории Интернета можно выделить две эпохи. Эпоха первая, давно прошедшая, представляла собой царство статичных Web-страниц — тех самых, что пишутся на языке *HTML* (HyperText Markup Language, язык гипертекстовой разметки) и хранятся в обычных текстовых файлах с расширениями htm или html. Эпоха вторая, которая продолжается и сейчас, ознаменована господством серверных Web-приложений, особых программ, которые получают данные из базы, обрабатывают их и генерируют на основе результатов обработки Web-страницы.

Но почему произошел такой резкий переход от статики к, можно сказать, динамике? Чем Web-дизайнеров не устраивали старые добрые Web-страницы?

Статичные Web-страницы

Написать статичную Web-страницу (для краткости их называют просто Web-страницами) — пара пустяков. Необходимый для их создания язык HTML сейчас знают даже школьники, равно как и язык *CSS* (Cascading Style Sheets, каскадные таблицы стилей), на котором в виде каскадных таблиц стилей и описывается их оформление.

> **WEB-СКРИПТЫ *НА ЯЗЫКЕ JAVASCRIPT***
>
> Существуют также *Web-сценарии*, или *Web-скрипты*, с помощью которых программируется поведение страниц или отдельных их элементов в ответ на действия посетителя или каких-либо событий, происходящих в Web-обозревателе. Они пишутся на языке программирования JavaScript. Однако в этой книге мы их касаться не будем.

А для разработки Web-страниц подойдет любой текстовый редактор — например, Блокнот, поставляемый в составе Windows.

Давайте создадим Web-страницу с вот таким кодом:

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Страница 1</title>
  </head>
  <body>
    <h1>Это страница 1</h1>
    <p>Щелкните гиперссылку внизу, чтобы перейти на страницу 2.</p>
    <p><a href="pages/2.html">Перейти</a>.</p>
  </body>
</html>
```

Эта страница очень проста — она включает лишь заголовок, два абзаца и гиперссылку для перехода на вторую страницу, которую мы скоро сделаем.

Сохраним нашу первую Web-страницу в какой-либо папке в кодировке UTF-8 под именем 1.html.

Теперь создадим еще одну страницу. Ее HTML-код будет таким:

```
<!doctype html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Страница 2</title>
  </head>
  <body>
    <h1>Это страница 2</h1>
    <p>Щелкните гиперссылку внизу, чтобы вернуться на страницу 1.</p>
    <p><a href="../1.html">Перейти</a>.</p>
  </body>
</html>
```

Здесь все то же самое, за единственным отличием — гиперссылка ведет на первую страницу.

Создадим в папке, где хранится первая страница, вложенную папку pages и сохраним в этой папке вторую страницу также в кодировке UTF-8, дав ей имя 2.html.

Откроем страницу 1.html в Web-обозревателе. Выглядеть она будет так, как показано на рис. 1.1.

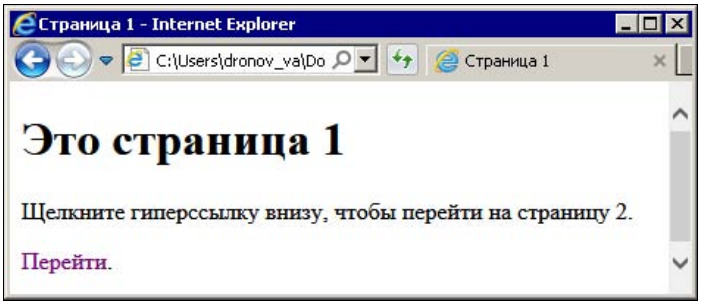

Рис. 1.1. Статичная Web-страница

Щелкнем на гиперссылке, чтобы перейти на вторую страницу, после чего вернемся на первую. Если мы не допустили в HTML-коде ошибок, все должно получиться.

Как видим, сайт на статичных Web-страницах делается очень просто — мы создаем страницы и раскладываем их по папкам. Структура папок отражает структуру самого сайта.

Обрабатываются статичные страницы также очень просто. Когда посетитель набирает в строке ввода адреса Web-обозревателя интернет-адрес нашего сайта и нажимает клавишу <Enter>, Web-обозреватель отправляет компьютеру, располагающемуся в Сети по этому адресу, особый запрос. Операционная система удаленного компьютера передает этот запрос программе *Web-сервера*. Та извлекает из запроса имя файла запрошенной Web-страницы, считывает этот файл и отправляет его Web-обозревателю. Последнему остается лишь получить файл, обработать его и вывести страницу на экран.

Ключевых преимуществ у статичных Web-страниц два. Во-первых, их очень просто создавать — для этого достаточно знать языки HTML и CSS, о которых говорилось ранее. Во-вторых, развернуть на компьютере статичный Web-сайт не составляет ни малейшего труда — нужно лишь установить на него и настроить программу Web-сервера, что можно сделать буквально за пять минут.

Сайты, основанные на статичных Web-страницах, активно создаются до сих пор. Это всевозможные домашние страницы, рекламные сайты и сайты-визитки.

Web-приложения

А теперь представим себе такую ситуацию. Мы создали основанный на статичных страницах корпоративный Web-сайт некоей фирмы, содержащий в своем составе каталог товаров. Какое-то время этот сайт работал и благополучно привлекал клиентов, пока начальство не поставило нам следующие задачи:

1. Реализовать в каталоге фильтрацию товаров по ключевым словам и сортировку их по наименованиям и цене.
2. Создать гостевую книгу.
3. Создать блог для сотрудников фирмы, где они смогли бы публиковать тематические статьи.

Конечно, соорудить некое подобие гостевой книги и блога на статичных Web-страницах можно. Посетители сайта и сотрудники фирмы станут присылать нам свои сообщения и статьи по электронной почте, а мы будем вставлять их в код Web-страниц. Неудобно, но вполне реализуемо.

Но как сделать фильтрацию и сортировку товаров в каталоге? Средствами HTML и CSS это реализовать всяко не получится.

Выход здесь один — создать некую программу, которая будет:

- работать совместно с Web-сервером;
- перехватывать запросы, получаемые Web-сервером, и активизироваться лишь при обращении к интернет-адресу списка товаров;
- извлекать из запросов введенные посетителем ключевые слова для поиска и критерий сортировки (такие данные обычно отправляют как часть интернет-адреса — методом GET);
- считывать из базы данных список товаров, сортировать его согласно заданным критериям и отфильтровывать лишь те товары, чьи наименования включают указанное ключевое слово;
- формировать на основе результирующего списка товаров обычную Web-страницу и передавать ее Web-серверу, который, в свою очередь, отправит ее посетителю.

В этом случае Web-страницы нашего сайта не будут храниться в файлах на жестких дисках компьютера, а станут генерироваться на «лету» особым приложением. Сразу видно, что такой подход решит огромное количество проблем, более того, мы, немного поразмыслив, без труда сможем создать на сайте, в том числе, и гостевую книгу с блогом, написав соответствующие программы.

Такие приложения, работающие совместно с Web-сервером и генерирующие страницы на основе данных, хранящихся в базе и введенных посетителем, носят название *Web-приложений*. Точнее, *серверных Web-приложений*, поскольку они функционируют на стороне Web-сервера.

> ***Клиентские Web-приложения***
>
> Существуют также *клиентские Web-приложения*, представляющие собой сложные Web-сценарии и обрабатывающие данные, как полученные от серверного Web-приложения, так и введенные посетителем, на стороне клиента — Web-обозревателя. Но, поскольку в этой книге не рассматривается клиентское Web-программирование, мы говорить о них не будем.

Как правило, сложные сайты включают в себя сразу несколько серверных приложений. Так, отдельное приложение формирует *главную Web-страницу*, которая выводится при обращении непосредственно по адресу сайта, отдельные же приложения генерируют список товаров, гостевую книгу, блог, фотогалерею и прочие разделы сайта. Обычные страницы, наподобие сведений о фирме и сайте или списка контактов, могут также генерироваться разными приложениями, равно как и создаваться одним «универсальным» приложением — это зависит от реализации.

Как мы уже знаем, Web-сайт, основанный на статичных страницах, представляет собой набор файлов и папок. Получив запрос, Web-сервер просто извлекает из него имя запрошенного файла, после чего считывает его с диска и пересылает Web-обозревателю. Например, получив запрос **http://www.somesite.ru/pages/3.html**, Web-сервер прочтет файл 3.html из папки pages, что находится в папке, где помещается сам сайт.

В Web-сайте, основанном на серверных приложениях, все несколько иначе. Каждое составляющее его приложение привязывается к определенному интернет-адресу, который можно рассматривать как «папку», не существующую в реальности. Web-сервер, получив запрос на обращение к такой «папке», запускает соответствующее ей серверное приложение. Скажем, если обратиться по интернет-адресу **http://www.somesite.ru/goods/**, будет запущено приложение, выводящее список товаров.

Самое интересное, что «структура» таких «папок» совершенно не обязана совпадать со структурой файлов и папок, составляющих сайт. С одной стороны, это может внести некоторую путаницу, но, с другой, позволяет реализовать весьма интересные сценарии работы...

Подробный рассказ об интернет-адресах и привязке их к приложениям еще впереди. А пока что давайте рассмотрим преимущества сайтов, основанных на серверных приложениях.

- Возможность хранить данные отдельно (как правило, в базе данных), или, говоря другими словами, отделить их от обработки и вывода. Это может быть полезно, если эти данные используются не только в сайте, но и где-либо еще, скажем, в бухгалтерском или складском приложении.
- Возможность реализовать обработку данных перед их выводом: фильтровать их, сортировать и объединять в группы. Мы можем даже считать количество позиций в списке, вычислять средние значения для параметров позиций и пр.
- Возможность получить данные от посетителя и сохранить их в базе.
- Повысить безопасность, заблокировав доступ к файлам сайта, не предназначенным для чужих глаз (той же базе данных).

Недостатков у подобных сайтов всего два, и оба не являются критичными:

- для создания серверных приложений необходимо дополнительно изучить язык программирования, на котором будут писаться эти приложения, и платформу их разработки. Что, впрочем, несложно;
- помимо собственно Web-сервера, нам понадобится также установить и настроить программное ядро соответствующей платформы. Сделать это также несложно — весь процесс установки и первоначальной настройки занимает несколько минут.

Платформ для разработки серверных приложений существует несколько. Прежде всего, это PHP (Pretty Home Page, симпатичная домашняя страница) — на данный момент имеющая наибольшую популярность. Приложения для этой платформы создаются на особом языке, который также называется PHP. А чтобы облегчить

труд Web-программиста, написано несколько библиотек, реализующих типовые задачи, которые в противном случае придется решать самому разработчику: Zend Framework, Yii и многие другие.

Мы же будем использовать платформу Django. Эта платформа служит для создания Web-приложений на языке Python (и сама, кстати, написана на этом языке).

Базы данных. Реляционные базы данных

Ранее мы уже говорили, что данные, с которыми работают серверные Web-приложения, хранятся в базах данных. Настала пора узнать, что это такое.

Что такое реляционная база данных?

Итак, *база данных* — это файл (или набор файлов), хранящий структурированную определенным образом информацию. Для обработки этой информации используются особые программы, называемые *системами управления базами данных*, или *СУБД*. Примеры СУБД: MySQL, PostgreSQL, Oracle, Microsoft SQL Server и Microsoft Access.

Базы данных делятся на несколько видов по способу структурирования содержащейся в них информации. Мы будем пользоваться *реляционными базами данных* — они служат для хранения информации, организованной в виде связанных друг с другом таблиц, и в настоящее время имеют наибольшее распространение.

Кстати, все упомянутые ранее СУБД обрабатывают реляционные базы данных.

Что хранит реляционная база данных?

Реляционная база хранит структуры, относящиеся к трем различным типам. Давайте их рассмотрим.

Таблицы, поля и записи

Таблица — это набор структурированных данных. Пример таблицы представлен на рис. 1.2.

Эта таблица содержит список товаров из пяти столбцов: категория товара, его наименование, описание, цена и признак того, имеется ли товар в наличии.

Каждая таблица, хранящаяся в базе данных, должна иметь уникальное в пределах этой базы имя. Это нужно для того, чтобы СУБД (да и мы сами) смогла найти эту таблицу.

Отдельная строка таблицы, содержащая реальные данные, называется *записью*. (Строка заголовка, выделенная на рис. 1.2 черным фоном, записью не является, т. к. содержит не реальные данные, описывающие какую-либо статью, а служебные сведения — заголовки столбцов.) На каждый товар, занесенный в таблицу, отводится одна запись.

category	name	description	price	in_stock
Веники	В-1	Классическая конструкция из экологически чистых материалов	100	*
Веники	В-2М	Усовершенствованная конструкция, выполненная с применением особо прочной синтетики	200	
Веники	В-2000	Современная конструкция, выполненная с применением нанотехнологий	2000	*
Метлы	М-1	Классическая конструкция из экологически чистых материалов	150	*
Метлы	М-2014О	Выпущена в честь Зимней олимпиады в Сочи 2014 года	15000	*

Рис. 1.2. Таблица — список товаров

Отдельная ячейка отдельной строки-записи называется *полем*. Можно сказать, что поле — это порция данных, составляющих запись. А сами данные, помещенные в поле, называются его *значением*.

Каждое поле обязано иметь уникальное в пределах таблицы имя. Имена полей, кстати, и приведены в строке заголовка таблицы на рис. 1.2.

Поле способно хранить данные какого-то одного типа: строки, числа, даты и т. п. *Тип* хранимых в поле данных задается при создании поля (и может быть потом изменен, если был задан ошибочно). СУБД не позволит записать, скажем, дату в поле, предназначенное для хранения строк. Типы данных, поддерживаемые большинством баз данных, приведены в табл. 1.1.

Таблица 1.1. *Типы данных, поддерживаемые большинством форматов баз данных*

Название	**Описание**
Строковый	Текст фиксированной длины, содержащий любые символы: буквы, цифры, знаки препинания, пробелы и пр. Максимальная длина текста, хранимого в таком поле, задается при его создании
Целочисленный	Целые числа
С плавающей точкой	Дробные числа
Логический	Значения вида «истина» (true) или «ложь» (false)
Дата	Значения даты
Дата и время	Объединенное значение даты и времени
Memo	Текст произвольной длины, содержащий любые символы: буквы, цифры, знаки препинания, пробелы и пр. Длина хранимого в таком поле текста не ограничена (по крайней мере, очень велика)
Счетчик	Постепенно увеличивающиеся и уникальные в пределах таблицы целые числа. Поля такого типа используются для специальных целей, в частности, в качестве ключевого поля (см. далее)

НЕСКОЛЬКО РАЗНОВИДНОСТЕЙ ТИПОВ ДАННЫХ

На самом деле, существует несколько разновидностей целочисленного типа данных и типа с плавающей точкой, различающихся величинами чисел, которые могут быть записаны в поле данного типа. Мы поговорим о них потом.

Посмотрим еще раз на рис. 1.2. Представленная там таблица имеет пять полей. А какого они типа? Давайте подумаем.

- Поле категории (`category`) получит строковый тип, поскольку название категории представляет собой слово. (Для него еще нужно указать предельную длину, но это можно сделать и потом.)
- Поле наименования товара (`name`) — также строкового типа. (Ему тоже следует задать предельную длину.)
- Полю описания товара (`description`) мы дадим тип memo. Такого рода данные могут иметь достаточно большой размер.
- В случае поля цены (`price`) вариант один — число с плавающей точкой. (Впрочем, раз цены на наши товары указываются в целых рублях, то можно использовать и целочисленный тип.)
- В случае поля признака, указывающего на наличие товара (`in_stock`), у нас тоже нет особого выбора — логический тип подходит для этого наилучшим образом.

Большинство форматов баз данных позволяют задать для поля *правила* — характеристики, которым должны удовлетворять записываемые туда данные. Такими условиями могут быть:

- обязательное наличие в поле какого-либо значения (*обязательное поле*);
- значение, которое должно быть помещено в поле при создании новой записи (*значение поля по умолчанию*);
- дополнительные условия (например, диапазон значений, в которые должно укладываться число).

Осталось только сказать, что набор полей с их именами, типами данных и условиями называется *структурой* таблицы. Сами реальные данные — содержимое полей и записей — в структуру не входят.

Индексы и ключи

Предположим, что мы создали таблицу, изображенную на рис. 1.2, заполнили ее данными и теперь пишем приложение, которое выводит на экран список товаров. И нам требуется отсортировать товары по какому-либо полю — например, по цене. В этом случае СУБД будет вынуждена:

1. Прочитать из базы данных все записи и поместить их в оперативную память, создав особый список.
2. Создать в памяти еще один список, содержащий все значения поля, по которому выполняется сортировка.
3. Переупорядочить эти значения, чтобы они шли по нарастанию или убыванию, и создать на их основе еще один список — третий по счету.

4. Соответственно переупорядочить записи таблицы и поместить их в отдельный список, который станет уже четвертым.

Если записей в таблице мало, этот процесс не займет много ни времени, ни оперативной памяти. А если записей там уже пара сотен?

Чтобы ускорить обработку записей, мы можем указать СУБД создать в базе отдельный массив данных, включающий все значения определенного поля таблицы, которые уже упорядочены нужным нам образом, и ссылки на соответствующие им записи. Понятно, что сортировка в этом случае будет выполняться много быстрее — ведь эти значения уже отсортированы, и СУБД остается лишь:

1. Прочитать из базы данных содержимое этого массива.
2. Прочитать из базы данных все записи таблицы.
3. Соответственно переупорядочить записи.

В этом случае, помимо ускорения обработки, потребуется еще и заметно меньше оперативной памяти — ведь будут созданы три списка, а не четыре, как ранее.

Такой список значений называется *индексом*, а поле, значения которого хранятся в индексе, — *индексированным*.

Индексы поддерживаются абсолютно всеми форматами баз данных и используются очень часто. В самом деле, индекс занимает немного места на диске и в памяти, а ускоряет операцию сортировки очень заметно. Единственный недостаток: при добавлении, изменении или удалении любой записи СУБД будет вынуждена соответственно изменить индекс, что отнимает некоторое время. Поэтому не стоит без необходимости создавать слишком много индексов.

Кроме сортировки, индексы также могут помочь при выполнении фильтрации записей. СУБД считывает индекс в память, ищет в нем значения, удовлетворяющие заданному критерию, и извлекает нужные записи из таблицы. Просто и быстро!

Изначально, при открытии таблицы, СУБД не считывает ни один индекс — они задействуются только при сортировке и фильтрации. Но имеется возможность сделать один из индексов загружаемым при открытии таблицы — при этом таблица будет изначально отсортирована согласно этому индексу. Такой индекс называется *ключевым*, или *ключом*, а задействованное в нем поле — *ключевым*. Ключевой индекс может быть только один на всю таблицу.

Ключевое поле должно удовлетворять следующим условиям:

- оно должно содержать значение (т. е. быть обязательным полем);
- оно должно содержать уникальные в пределах таблицы значения (быть уникальным полем).

Обычно в качестве ключевого применяется поле типа счетчика (см. табл. 1.1). Такие поля подходят для этого наилучшим образом. (Хотя, конечно, никто не мешает нам применить для этого поле любого другого типа.)

Ключевые индексы используются для того, чтобы однозначно идентифицировать какую-либо запись для изменения хранящихся в ней значений и ее удаления. Они также применяются для установления межтабличных связей.

Связи

Кстати, поговорить о межтабличных связях сейчас самое время. И вот почему...

Давайте посмотрим на таблицу, что показана на рис. 1.2. В частности, на поле `category`, где хранится категория товара. Чем примечательны хранящиеся в нем значения? И чем неоптимален такой способ указания категории?

Тем, что в этом поле записывается само ее название. Во-первых, оно довольно длинное и, соответственно, занимает немало места в базе данных. (Да, сейчас у нас категории имеют короткие названия. Но это сейчас...) Во-вторых, при вводе мы можем по ошибке указать название категории неправильно и тем самым нарушить работу приложения. В-третьих, если решим изменить название какой-либо категории, нам придется перебрать все относящиеся к ней товары и внести нужные правки в соответствующие им записи, что отнимет много времени.

Простое и красивое решение предлагает нам сам формат реляционных баз, который представляет данные как набор связанных таблиц.

Создадим в базе еще одну таблицу — для хранения списка категорий (рис. 1.3). Она будет содержать следующие поля:

- `id` — уникальный идентификатор записи, тип — счетчик, ключевое;
- `name` — название категории, тип — строковый.

id	name
1	Веники
2	Метлы

Рис. 1.3. Таблица — список категорий товаров

Теперь удалим из таблицы-списка товаров поле `category` и создадим в ней поле с тем же именем, но целочисленного типа. В этом поле будет храниться значение поля `id` таблицы-списка категорий, соответствующее данному товару. То есть вместо того, чтобы хранить в списке товаров сами названия категорий, мы будем помещать туда лишь ссылки на них.

Преимуществ у такого подхода два. Во-первых, мы храним в таблице не длинную строку, а короткое целое число, за счет чего размер базы данных станет меньше, а процесс ее обработки — быстрее. Во-вторых, мы не будем иметь никаких проблем, если вдруг захотим поменять название категорий, — ведь для этого нам потребуется исправить всего одну запись в таблице — списке категорий.

Можем себя поздравить — мы только что создали первую в нашей практике *связь* между таблицами (рис. 7.4).

В нашем случае одна запись списка категорий связана с произвольным количеством записей списка товаров. (В реальности так и бывает — в одну категорию могут входить множество товаров.) Это связь *один-ко-многим*. При этом таблица-список категорий будет *первичной*, или *родительской*, поскольку она, можно сказать, под-

чиняет себе связанные записи. А таблица-список товаров, напротив, станет *вторичной* или *дочерней*. Обе таблицы при этом станут *связанными*.

Поле вторичной таблицы, содержащее ссылки на записи первичной таблицы (у нас это поле `category`), называется *внешним индексом*. «Внешний» — потому что это поле внешнее по отношению к первичной таблице, а «индекс» — потому что практически всегда на основе этого поля создается индекс.

category	name	description	price	in_stock
1	В-1	Классическая конструкция из экологически чистых материалов	100	*
1	В-2М	Усовершенствованная конструкция, выполненная с применением особо прочной синтетики	200	
1	В-2000	Современная конструкция, выполненная с применением нанотехнологий	2000	*
2	М-1	Классическая конструкция из экологически чистых материалов	150	*
2	М-2014О	Выпущена в честь Зимней олимпиады в Сочи 2014 года	15000	*

id	name
1	Веники
2	Метлы

Рис. 1.4. Таблица-список категорий, связанная с таблицей-списком товаров

Правила построения реляционных баз данных требуют выделения одинаковых значений в отдельные связанные таблицы. В таком случае нам самим будет легче как поддерживать базу впоследствии, так и писать приложения, которые станут с ней работать.

Но что случится, если мы попытаемся удалить запись первичной таблицы, на которую ссылаются записи таблицы вторичной? СУБД просто не позволит это сделать, выведя нам сообщение об ошибке нарушения ссылочной целостности. Так что разорвать связь между записями, случайно или преднамеренно, у нас не получится.

Еще мы можем создать между таблицами связь типа *один-к-одному*, когда на одну запись первичной таблицы может ссылаться только одна запись таблицы вторичной. Если же мы попытаемся привязать к записи первичной таблицы еще одну запись вторичной, то, опять же, получим сообщение об ошибке. Однако такие связи на практике применяются довольно редко.

На этом ударный курс теории баз данных можно считать законченным.

Основные принципы разработки серверных Web-приложений

На очереди — не менее ударный курс теории программирования серверных Web-приложений.

Каждое более или менее сложное приложение (это относится к программам любого типа, не только к серверным) состоит из нескольких программных модулей, выполняющих различные задачи. И это понятно — ведь написать большое приложение, состоящее из одного модуля, очень трудно, если вообще возможно.

Все модули, из которых состоит серверное приложение, можно разделить на четыре разновидности: модели, контроллеры, шаблоны и служебные. Рассмотрим их поочередно.

Модели

Модель — это программный модуль, входящий в состав приложения, который служит своего рода посредником между остальными его модулями и базой данных. Или, говоря другими словами, модель — суть представление базы данных, ее таблиц, полей, индексов и связей в терминологии языка программирования, на котором пишется данное приложение.

Модель выполняет следующие задачи:

- описывает таблицы базы данных и их структуру в терминологии используемого языка программирования. Благодаря этому мы можем получать данные из базы, не прибегая к сторонним средствам;
- представляет считанные из базы данные в терминологии используемого языка программирования. Так что мы, считав с помощью модели какую-либо запись таблицы, сможем обработать ее средствами выбранного нами языка, опять же, не привлекая сторонние инструменты;
- реализует механизм выборки данных, их фильтрации и сортировки;
- реализует механизм добавления в таблицы новых записей, а также правки и удаления существующих;
- следит за корректностью данных, позволяя обрабатывать возникающие ошибки средствами выбранного языка программирования;
- возможно, расширяет набор средств, предоставляемых принятым форматом баз данных, добавляя к нему дополнительные инструменты, которые созданы разработчиком приложения или сторонними программистами.

Если уж совсем коротко, то модель — наш пропуск в базу данных.

Приложение может включать в свой состав произвольное количество моделей. Обычно каждая модель соответствует определенной таблице в базе данных.

Отметим сразу, что модели в приложении всегда играют подчиненную роль. Они вызываются другими модулями, относящимися к другой разновидности — кон-

троллерам, когда им требуется обратиться к базе данных, и вызываются явно, указанием в программном коде особой команды.

Контроллеры

Контроллер — это модуль приложения, выполняющий непосредственно обработку данных. Это главная действующая часть приложения, его сердце.

Обязанности у контроллера следующие:

- выборка данных из базы посредством явно вызываемых моделей;
- обработка полученных данных: фильтрация и сортировка;
- получение данных, отправленных пользователем;
- занесение полученных от пользователя данных в базу, опять же, посредством явно вызванных моделей, или иная их обработка;
- запуск формирования на основе результата обработки данных Web-страницы, которую увидит посетитель сайта.

Приложение может содержать произвольное количество контроллеров. Каждый контроллер соответствует определенному действию, выполняемому приложением, — так, выборка списка товаров выполняется одним контроллером, а добавление нового товара в список — другим.

Каждый контроллер ставится в соответствие определенному интернет-адресу приложения. Например, контроллер, выводящий список товаров, ставится в соответствие интернет-адресу **/goods/**, а контроллер, который добавляет товар в список, — интернет-адресу **/goods/add/**. Отслеживанием запрошенных посетителем интернет-адресов занимается особый служебный модуль, входящий в программное ядро приложения, он же выполняет запуск нужного контроллера при обращении к «его» адресу.

Контроллеры в процессе работы загружают и запускают остальные модули: модели и шаблоны. Так что их вполне можно назвать хозяевами положения... или приложения...

Шаблоны

Шаблон — это модуль приложения, единственное назначение которого — принять подготовленные контроллером данные и сформировать на их основе результирующую Web-страницу. Если совсем коротко, то шаблоны занимаются выводом данных.

Если модели и контроллеры представляют собой программы, написанные на языке программирования, то шаблон — это фактически обычная Web-страница, созданная на языке HTML. За единственным исключением — в ее код вставлены особые теги (*теги шаблона*), которые указывают, какие данные и в каком формате следует сюда поместить.

Приложение может включать в свой состав произвольное количество шаблонов. Правило здесь очень простое — каждой Web-странице, формируемой приложением, соответствует свой шаблон.

Как и модели, шаблоны явно вызываются контроллерами, когда им потребуется вывести обработанные данные.

Служебные модули

Что касается служебных модулей, входящих в состав приложения, то они включают в себя:

- модуль, отслеживающий запрошенные посетителем интернет-адреса и запускающий соответствующие им контроллеры (*диспетчер*);
- модуль, обрабатывающий теги шаблонов, т. е. подставляющий на их место отформатированные указанным образом данные (*шаблонизатор*);
- модуль настроек приложения;
- разнообразные модули, выполняющие типовые задачи Web-программирования: разграничение доступа, кэширование и пр.

Служебные модули составляют так называемое *программное ядро* приложения. Оно обеспечивает само функционирование приложения, не зависит от структуры создаваемого сайта и выполняемых им задач, вследствие чего может быть использовано в самых разных сайтах, обычно пишется один раз и модифицируется крайне редко, как правило, лишь с целью существенно расширить его функциональность.

Служебные модули могут как вызываться явно, когда в них возникнет нужда, так и постоянно функционировать «за кулисами». Одни служебные модули задействуются в любом случае (например, диспетчер), а другие могут включаться и отключаться в настройках приложения в зависимости от того, наличествует ли в них необходимость или нет.

Осталось лишь сказать, что принцип построения приложения, или, говоря другими словами, его *архитектура*, когда функциональность разделяется между моделями, контроллерами и шаблонами, носит название *модель-контроллер-шаблон*. (В зарубежной литературе применяется термин *model-view-controller*, *MVC*.)

Что дальше?

В этой главе мы выяснили, что такое Web-приложения и какие преимущества они несут по сравнению со статичными Web-страницами, что такое база данных и что она хранит. Еще мы узнали об архитектуре построения приложений модель-контроллер-шаблон и познакомились с моделями, контроллерами и шаблонами.

Следующая глава будет целиком посвящена языку программирования Python, на котором мы будем писать наши Web-приложения. Язык этот очень прост, в чем мы скоро и убедимся.

ГЛАВА 2

Язык программирования Python

В предыдущей главе мы рассмотрели основные принципы серверного Web-программирования и теорию баз данных. В этой главе мы приблизимся к практике, приступив к изучению языка программирования Python.

Python — высокоуровневый, объектно-ориентированный, тьюринг-полный язык программирования, одинаково хорошо подходящий для разработки как простейших командных скриптов, так и сложных настольных и Web-приложений. В комплекте с ним поставляется богатейшая стандартная библиотека, включающая мощные средства для обработки текстов, поддержки шифрования, работы с файлами, реализации обмена данных через Интернет и многое другое.

Но нас пока что интересуют базовые возможности Python, его синтаксис, поддерживаемые им типы данных, управляющие структуры и инструменты для работы с классами и объектами. Ими-то мы здесь и займемся.

Интерактивный интерпретатор Python

Разговор об этом языке будет сопровождаться большим количеством примеров. Чтобы проверить их в действии, мы можем использовать *интерактивный интерпретатор* Python, входящий в комплект его поставки.

После установки Python на компьютер в системном меню **Пуск** появится группа с именем вида **Python <*номер версии без последней цифры*>**. (Например, у автора, установившего версию 3.3.4, эта папка носит имя **Python 3.3**.) В ней имеется ярлык **IDLE (Python GUI)**, который и запускает интерактивный интерпретатор.

Окно этой программы показано на рис. 2.1. Оно содержит большую область редактирования, куда вводится Python-код, и вследствие этого напоминает окно текстового редактора.

Введем в это окно следующее выражение:

```
2 + 3
```

и нажмем клавишу <Enter>. Тем самым мы дадим Python указание вывести на экран результат сложения чисел 2 и 3. Результат этот, равный 5, появится в следующей строке.

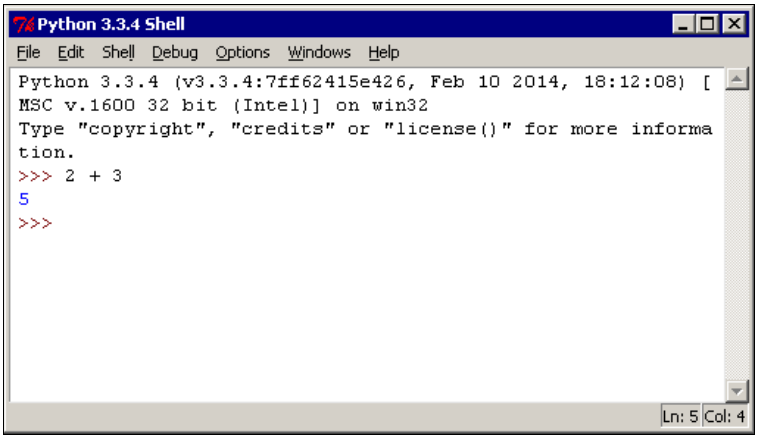

Рис. 2.1. Окно интерактивного интерпретатора Python

Что ж, по крайней мере, складывать числа этот язык умеет. Посмотрим, на что он еще способен...

Основные понятия Python

Начнем мы с самых простых понятий Python-программирования. Это выражение, оператор, функция и переменная.

Выражения

Ранее мы назвали команду `2 + 3`, введенную в окне интерактивного интерпретатора, *выражением*. И тем самым озвучили первый термин из всех, что нам предстоит запомнить.

Выражение в терминологии программирования — это команда, выполняющая законченное действие. Таким действием может быть вычисление некоего значения (как в нашем случае), создание какой-либо структуры данных, команда, управляющая выполнением программного кода, вызов функции или метода (о них мы поговорим потом) или что-то иное.

Любое выражение в Python должно завершаться символами возврата каретки и перевода строки, которые вставляются в программный код нажатием клавиши <Enter>.

Давайте рассмотрим примеры еще нескольких выражений.

- `3 * 4 + 8`

  Умножаем 3 на 4, прибавляем к получившейся сумме 8 и получаем 20. Операция умножения выполняется перед операцией сложения, т. к. она имеет больший приоритет.

- `5 / 6`

  Делим 5 на 6 и получаем длинный результат — 0.8333333333333334.