ВЛАДИМИР ДРОНОВ

bhv

PHP, MySQL и DREAMWEAVER MX 2004

РАЗРАБОТКА ИНТЕРАКТИВНЫХ WEB-САЙТОВ

НАЧАЛА WEB-ДИЗАЙНА

ВВЕДЕНИЕ В БАЗЫ ДАННЫХ

НАПИСАНИЕ WEB-ПРИЛОЖЕНИЙ НА ЯЗЫКЕ PHP

СОЗДАНИЕ СЕРВЕРНЫХ СТРАНИЦ СРЕДСТВАМИ DREAMWEAVER

ОБУЧЕНИЕ WEB-ПРОГРАММИРОВАН НА ПРИМЕРАХ

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

Владимир Дронов

PHP, MySQL и DREAMWEAVER MX 2004

РАЗРАБОТКА ИНТЕРАКТИВНЫХ WEB-САЙТОВ

Санкт-Петербург

«БХВ-Петербург»

2005

УДК 681.3.06
ББК 32.973.26-018.2
Д75

Дронов В. А.

Д75 PHP, MySQL и Dreamweaver MX 2004. Разработка интерактивных Web-сайтов. — СПб.: БХВ-Петербург, 2005. — 448 с.: ил.

ISBN 5-94157-598-X

В качестве базового средства разработки интерактивных Web-сайтов на платформе PHP-MySQL выбран популярный Web-редактор Macromedia Dreamweaver MX 2004. В качестве примера рассмотрено создание сайта — архива программ и электронных статей. Изложение построено по принципу: от простого — к сложному. Простейшие статичные Web-страницы создаются в редакторе Dreamweaver, попутно приводится краткое описание языка HTML. Простейшие серверные страницы, извлекающие данные из базы MySQL, также создаются в редакторе Dreamweaver, при этом подробно разбираются все сценарии PHP, созданные Dreamweaver, и описывается их работа. Параллельно дается введение в базы данных и приводится краткое описание языка PHP. Наиболее сложные Web-страницы создаются средствами PHP-MySQL без использования Dreamweaver. Приводятся примеры разработки элементов развитого Web-портала: выбираемой цветовой схемы, управления файлами через Web-интерфейс, собственного списка рассылки и др. Книга ориентирована на читателя, имеющего базовые понятия об интернет-технологиях.

Для Web-программистов

УДК 681.3.06
ББК 32.973.26.-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. гл. редактора	*Евгений Рыбаков*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инна Тачина*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 21.01.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 36,12.
Тираж 4000 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-598-X

© Дронов В. А., 2005
© Оформление, издательство "БХВ-Петербург", 2005

Оглавление

Введение **1**

О чем вообще идет речь? 1
Dreamweaver: "Делай, как я!" 2
Благодарности 3

ЧАСТЬ I. НАШ ПЕРВЫЙ WEB-САЙТ **5**

Глава 1. Современные интернет-технологии **7**

Основные принципы работы Интернета 7
Что такое Интернет. Сервисы Интернета 7
Клиенты и серверы 10
Протоколы 13
Интернет-адреса 15
Основные понятия WWW 18
Web-страницы и Web-сайты 18
Web-обозреватели 20
Web-серверы 23
Публикация Web-сайта в Интернете. Хостинг-провайдеры 24
Что дальше? 25

Глава 2. HTML — язык написания Web-страниц **26**

Введение в язык HTML 26
Теги HTML. Форматирование текста 27
Графика на Web-страницах. Внедренные элементы 31
Гиперссылки 34
Правильно оформленные Web-страницы 36
Иерархия тегов HTML 37
Средства оформления Web-страниц 38

Каскадные таблицы стилей (CSS) 38
Создание стилей 39
Три способа задания стилей 42
Почему "каскадные"? 43
Теги физического форматирования HTML 45
Кодирование текста. Проблема русских кодировок 47
Начала сайтостроения 49
Планирование сайта 49
Логическая структура Web-сайта 50
Проектируем наш первый Web-сайт 52
Что дальше? 53

Глава 3. Работа с Macromedia Dreamweaver MX 2004 54

Предварительная настройка Dreamweaver 55
Основы работы в Dreamweaver 57
Создание новой Web-страницы 58
Набор текста 59
Форматирование фрагментов текста 61
Форматирование абзацев 66
Специальные символы и нетекстовые элементы 68
Работа с таблицами 71
Создание таблиц 71
Работа с таблицей 74
Как формируются таблицы 74
Более сложные таблицы 76
Вставка графических изображений 79
Создание гиперссылок 82
Работа со стилями CSS 84
Предварительный просмотр Web-страниц 89
Вызов справки 89
Что дальше? 91

Глава 4. Работа с Web-сайтом в Dreamweaver 92

Подготовка к публикации сайта 93
Регистрация сайта в Dreamweaver 93
Работа с файлами сайта. Панель *Files* 97
Проверка Web-страниц 100
Проверка правильности HTML-кода 100
Проверка гиперссылок 102
Взаимодействие панели *Files* и окна документа 103
Публикация сайта 104
Публикация сайта на локальном Web-сервере 105

Публикация сайта на удаленном Web-сервере ... 108
Использование для публикации Web-сайта протокола FTP ... 108
Настройка Dreamweaver для публикации сайта по FTP ... 110
Публикация сайта по протоколу FTP ... 113
Что дальше? ... 113

ЧАСТЬ II. НАШИ ПЕРВЫЕ СЕРВЕРНЫЕ ПРОГРАММЫ ... 115

Глава 5. Введение в Web-программирование ... 117

Недостатки статичных Web-страниц и их преодоление ... 117
Данные и их представление ... 118
Недостатки статических Web-страниц ... 119
Серверные программы — радикальный способ отделить информацию от представления ... 120
Технологии создания серверных программ ... 122
Активные серверные Web-страницы ... 122
Другие технологии серверного программирования ... 124
Наш второй Web-сайт будет использовать активные серверные страницы ... 125
Что дальше? ... 126

Глава 6. Базы данных ... 127

Введение в реляционные базы данных ... 127
Что такое реляционные базы данных ... 127
Составные части реляционной базы данных ... 128
Таблицы, поля и записи ... 128
Правила ... 131
Индексы и ключи ... 131
Связи ... 135
Настольные и серверные реляционные СУБД ... 137
Язык обработки данных SQL ... 140
Зачем нужен SQL ... 140
Выборка записей из таблицы ... 141
Простейшие запросы выборки данных ... 141
Сортировка данных ... 143
Фильтрация данных ... 144
Задание связей между таблицами ... 146
Псевдонимы полей ... 147
Агрегатные функции SQL ... 148
Изменение записей таблицы ... 150
Добавление записи ... 150
Изменение записи ... 150
Удаление записи ... 152

Другие запросы SQL 152
Разграничение доступа. Права 152
Сервер данных MySQL и его возможности 154
Создаем базу данных для нашего сайта 157
Что дальше? 159

Глава 7. PHP — технология написания серверных приложений 160

Основные понятия PHP 160
Написание сценариев PHP 161
Операторы, аргументы и выражения 163
Переменные 164
Типы данных 166
Логический 166
Целочисленный 166
С плавающей точкой 166
Строковый 167
NULL 168
Операторы 168
Арифметические 169
Оператор объединения строк 170
Операторы присваивания 170
Операторы сравнения 171
Логические операторы 172
Как PHP вычисляет выражения, содержащие логические операторы 173
Совместимость и преобразование типов данных 173
Приоритет операторов 175
Сложные выражения PHP 177
Блоки 177
Условные выражения 177
Выражения выбора 179
Циклы 181
Цикл со счетчиком 181
Цикл с постусловием 182
Цикл с предусловием 183
Прерывание цикла 184
Функции 185
Создание функций 185
Вызов функций 186
Использование переменных внутри тела функции 188
Встроенные функции PHP 189
Массивы 190
Создание массивов и работа с ними 190

Цикл просмотра ... 192
Константы ... 193
Комментарии ... 194
Что дальше? ... 195

Глава 8. Простейшие серверные Web-страницы. Вывод данных ... 196

Подготовка к созданию серверных страниц ... 196
Регистрация базы данных в Dreamweaver ... 199
Создание простейших серверных страниц ... 203
Создание набора записей ... 204
Создание самой серверной страницы ... 207
Разбор сценариев PHP, используемых для вывода записей ... 212
Передача данных между серверными страницами ... 215
Метод передачи данных GET ... 215
Создание Web-страниц, передающих данные друг другу ... 217
Разбор PHP-кода, обеспечивающего обработку принятых данных ... 220
Более сложные серверные страницы ... 222
Написание сценариев PHP вручную ... 222
Одновременный вывод значений из нескольких полей ... 224
Страницы с несколькими наборами данных ... 227
Правильный вывод значений даты ... 229
Условный вывод элементов Web-страницы ... 230
Вывод сведений о наборе записей ... 232
Что дальше? ... 234

Глава 9. Реализация ввода и правки данных ... 235

Как реализуется ввод и передача данных ... 235
Ввод данных. Формы ... 236
Кодирование данных ... 238
Передача данных ... 239
Административные и пользовательские Web-страницы ... 241
Создание простых серверных Web-страниц для ввода и правки данных ... 243
Простая страница для добавления записи ... 243
Разбор сценариев PHP, используемых для добавления записи ... 249
Задание значений по умолчанию для элементов управления ... 253
Простая страница для правки записи ... 256
Разбор сценариев PHP, используемых для правки записи ... 261
Простая страница для удаления записи ... 262
Более сложные серверные страницы для ввода и правки данных ... 264
Частный случай ввода данных — поисковая машина ... 271
Что дальше? ... 273

ЧАСТЬ III. РЕШАЕМ ВОПРОСЫ БЕЗОПАСНОСТИ И ЦЕЛОСТНОСТИ ДАННЫХ 275

Глава 10. Введение в безопасность и целостность данных 277

Безопасность и разграничение доступа 277
Целостность данных 279
Что дальше? 280

Глава 11. Разграничение доступа 281

Создание таблицы списка пользователей 282
Создание страницы входа на сайт 283
Процесс создания страницы входа на сайт в Dreamweaver 283
Сессии. Переменные уровня сессии 286
Разбор кода PHP, выполняющего вход 288
Разграничение доступа к закрытым Web-страницам 292
Процесс разграничения доступа к страницам в Dreamweaver 292
Разбор кода PHP, выполняющего разграничение доступа 294
Создание страницы выхода с сайта 297
Процесс создания страницы выхода с сайта в Dreamweaver 297
Разбор кода PHP, выполняющего выход 299
Создание административных страниц для управления пользователями 300
Разграничение доступа к фрагментам Web-страниц 301
Что дальше? 305

Глава 12. Поддержание ссылочной целостности данных 306

Простейший способ поддержания ссылочной целостности 307
Второй — более сложный — способ поддержания ссылочной целостности 308
Недостаток первого способа поддержания ссылочной целостности и попытка его устранить 309
Блокировка таблиц MySQL и ее использование 311
Реализация второго способа поддержания ссылочной целостности 312
Что дальше? 313

ЧАСТЬ IV. НАНОСИМ ПОСЛЕДНИЕ ШТРИХИ 315

Глава 13. Вывод сообщений об ошибках 317

Страница, сообщающая об ошибке 404 318
Сообщения об ошибках в сценариях PHP 320
Что дальше? 326

Глава 14. Хранение данных на стороне клиента 327

Задание порядка сортировки записей 327
Хранение настроек посетителя 330
Способы хранения настроек 330
Cookie и их использование 332
Реализация хранения настроек в cookie 334
Какие данные стоит хранить в cookie 335
Что дальше? 336

Глава 15. Управление файлами через Web-интерфейс 337

Два способа управления файлами на сайте 338
Отправка файлов на Web-сервер 340
Как отправить файл из Web-обозревателя 340
Как принять отправленный файл 341
Реализация отправки файла 345
Управление файлами, находящимися на Web-сервере 347
Средства PHP для управления файлами 347
Выбор папки 348
Просмотр содержимого папки 348
Получение сведений о файлах и папках 350
Копирование, перемещение, переименование и удаление файлов 351
Создание и удаление папок 352
Создание Web-интерфейса для управления файлами 352
Просмотр содержимого папки 352
Переходы между папками 355
Управление файлами 358
Создание и удаление папок 360
Что дальше? 363

Глава 16. Организация почтовой рассылки 364

Введение в почтовые рассылки 365
Что такое почтовая рассылка 365
Как осуществляются почтовые рассылки 366
Реализация службы рассылки на PHP 367
Два способа реализовать службу рассылок на PHP 367
Средства PHP для отправки почты 369
Отправка почты через собственный почтовый сервер 369
Прямая отправка письма на почтовый сервер 371
Как отправить письмо сразу по нескольким адресам 373
Создание соответствующих Web-страниц 374

Страница регистрации нового подписчика 374
Страница выполнения рассылки 376
Страница отписки от рассылки 380
Что дальше? 381

Заключение 383

ПРИЛОЖЕНИЯ 387

Приложение 1. Установка Web-сервера Apache 389

Установка 389
Запуск и остановка 394
Предварительная настройка 395
Доступ к справочной системе Apache 396

Приложение 2. Установка сервера данных MySQL 397

Установка 397
Предварительная настройка 401
Запуск и остановка 402
Запуск и остановка под Windows 95, 98 и Me 402
Запуск и остановка под Windows NT 403
Запуск и остановка под Windows 2000, XP, 2003 404
Доступ к справочной системе MySQL 404
Вспомогательные программы 405

Приложение 3. Установка обработчика PHP 406

Установка 406
Предварительная настройка 407
Запуск и остановка 408
Доступ к справочной системе PHP 408

Приложение 4. Установка и использование клиента данных phpMyAdmin 409

Установка и настройка 409
Использование 411
Вход 411
Создание базы данных 412
Создание таблиц 413
Создание полей 413

Создание индексов 415
Правка и удаление полей, индексов, таблиц и баз данных 416
Правка и удаление полей 416
Правка и удаление индексов 417
Правка и удаление таблиц 418
Правка и удаление баз данных 418
Управление пользователями 419
Средства управления пользователями phpMyAdmin 419
Создание пользователя 420
Правка и удаление пользователей 424
Работы с данными 424
Выход 425
Другие программы клиентов данных MySQL 426

Предметный указатель 427

Введение

"Вот еще одна книга о PHP и MySQL. Боже, сколько их уже издано!.. Так нет, нужно опять тратить бумагу, чтобы рассказать о том же самом".

Так может сказать какой-нибудь знаток компьютерных и интернет-технологий, разглядывая разноцветные книжные обложки на прилавке магазина. Действительно, книг о создании Web-сайтов с использованием серверных Web-страниц PHP и баз данных MySQL сейчас написано очень (возможно, даже слишком) много. "PHP!", "MySQL!", "PHP и MySQL!", "MySQL и PHP!!!" — кричат обложки. Стоит ли писать еще одну книгу на ту же самую тему?

Вопрос не в том, *стоит ли* писать. И даже не в том, *о чем* писать. А *как*! Да-да, именно как писать подобные книги!

Но стоп, давайте обо всем по порядку. Потенциальный читатель наверняка приготовил уйму вопросов, и автору придется на них отвечать. А отвечать лучше всего не торопясь, спокойно, без лишнего шума. Итак...

О чем вообще идет речь?

Действительно, что это за штуки такие — PHP, MySQL и Dreamweaver? И с чем их едят?

Начнем с самого начала.

MySQL — это сервер данных. Особая программа, позволяющая хранить упорядоченные данные в особых структурах, называемых базами данных. Например, в такую базу можно записать инвентарную книгу, список работников, экзаменационные ведомости за несколько лет, список книг, хранящихся в библиотеке, и многое другое. И не просто записать и сохранить, а еще и получать эти данные по первому запросу. И не просто получать, а быстро, без особого труда и только те данные, которые были запрошены.

PHP — это платформа для создания серверных Web-страниц. Грубо говоря, с ее помощью можно писать программы, которые по запросу посетителя сайта создают Web-страницы и отправляют их ему. Такие Web-страницы могут содержать данные, хранящиеся в базе данных, в том числе, и MySQL.

MySQL и PHP впечатляют даже по отдельности. А уж вместе — это просто мечта! Например, серверная программа, написанная на PHP и являющаяся частью сайта интернет-магазина, может извлекать из базы данных MySQL список товаров и формировать из него красивую Web-страницу. А другая серверная программа, другая часть того же интернет-магазина, спрашивает посетителя, какой товар он хочет заказать и по какому адресу этот товар нужно выслать, и сама записывает в ту же базу данных сведения о новом заказе.

Неудивительно, что PHP и MySQL так популярны. Мало того, что это весьма мощные программы, на которых можно реализовать практически все что угодно. Они еще прекрасно работают в связке и — внимание, сторонники использования легальных программ! — абсолютно бесплатны. Находка для создателя некоммерческого Web-сайта с большим информационным наполнением. Да и вполне коммерческий интернет-магазин они тоже "потянут".

Dreamweaver же — мощнейший пакет для создания Web-страниц, обычных (статичных) и серверных, в том числе и написанных на языке PHP. С его помощью мы можем создавать простейшие Web-страницы PHP, вообще не зная этого языка и принципов программирования на нем. А возможностей у Dreamweaver столько, что о нем можно написать несколько толстенных книг, и все равно какие-то секреты наверняка останутся неописанными.

Dreamweaver создан фирмой Macromedia и — увы!.. — является платным. Однако доступна демонстрационная версия, работающая 30 дней. Так что все желающие могут попробовать его в действии, так сказать, "повертеть в руках".

Но знаток, собаку съевший на PHP и MySQL, недовольно кривит губы. "Прекрасно, — бормочет он. — PHP и MySQL — это понятно. Но причем тут Dreamweaver?"

Dreamweaver: "Делай, как я!"

А вот здесь мы вплотную подошли к ответу на вопрос — как писать. Вообще, на взгляд автора, это главный вопрос, на который автору любой книги, не только компьютерной тематики, нужно всегда знать ответ.

Писать книги о PHP и MySQL можно по-разному. Можно начать с описания языка, продолжить примерами простейших страничек, а закончить описанием работы с базами данных MySQL. Можно сразу начать с описания работы с базами данных, продолжить тонкостями составления запросов к ним, а закончить таким дебрями, что читателей дрожь пробьет. А еще можно почти полностью сосредоточиться на MySQL, а PHP описать поверхностно, только "чтобы было".

А можно и совсем по-другому — вот так. Значит, причем здесь Dreamweaver? А вот причем...

Ранее упоминалось, что Dreamweaver может сам создавать простейшие серверные Web-страницы PHP, извлекающие данные из баз MySQL. Так вот,

мы начнем с того, что будем пользоваться для создания своих первых страничек услугами Dreamweaver и внимательно разбирать код PHP, который он создаст. В этом нам поможет руководство по PHP, которое можно найти на официальном сайте этой платформы — **http://www.php.net**.

Когда мы приобретем кое-какой опыт, то сами начнем писать серверные страницы. В конце концов, возможности Dreamweaver в PHP-программировании не очень велики, многие нужные вещи он просто не сможет для нас сделать. А мы сможем!

Фактически мы будем учиться писать серверные страницы PHP на готовых примерах. Это, пожалуй, лучший способ обучения программированию. Автор сам точно так же изучал в свое время PHP — читая код, созданный Dreamweaver. "Делай, как я! — командует Dreamweaver. — Учись у меня!" И этому призыву стоит последовать. Хотя бы первое время.

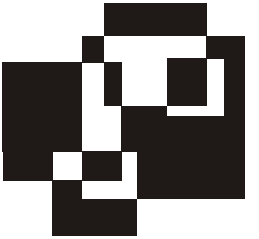
Знаток компьютеров пожимает плечами: "Программирование — это, прежде всего, искусство, и искусству компьютеры не обучены". Что ж, спору нет. Но не будем об этом забывать, что программы создают люди. А ведь именно люди создали искусство...

В качестве примера мы создадим Web-сайт — архив статей и файлов (программ, архивов и пр.). Списки статей и файлов мы будем хранить в базе данных MySQL. А извлекать их оттуда будут серверные страницы PHP, которые нам поможет создать Dreamweaver.

Благодарности

Автор приносит благодарности своим родителям, знакомым и коллегам по работе.

- Губине Наталье Анатольевне, начальнику отдела АСУ Волжского гуманитарного института (г. Волжский Волгоградской обл.), где работает автор, — за понимание и поддержку.
- Всем работникам отдела АСУ — за понимание и поддержку.
- Родителям — за терпение, понимание и поддержку.
- Архангельскому Дмитрию Борисовичу — за дружеское участие.
- Шапошникову Игорю Владимировичу — за содействие.
- Рыбакову Евгению Евгеньевичу, заместителю главного редактора издательства "BHV-Петербург", — за неоднократные побуждения к работе, без которых автор давно бы обленился.
- Издательству "BHV-Петербург" — за издание моих книг.
- Всем российским программистам, занятым в разработке MySQL и PHP, — за прекрасные программные продукты.
- Всем, кого я забыл здесь перечислить, — за все хорошее.



Часть I

Наш первый Web-сайт

Глава 1. Современные интернет-технологии

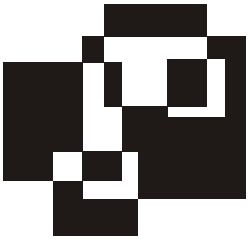
Глава 2. HTML — язык написания Web-страниц

Глава 3. Работа с Macromedia Dreamweaver MX 2004

Глава 4. Работа с Web-сайтом в Dreamweaver

В первой части мы займемся тем, что создадим свой самый первый Web-сайт. Да-да! Сначала, конечно, будет вводный курс интернет-технологий, потом мы выясним, как же создаются Web-страницы, далее изучим сам пакет Macromedia Dreamweaver MX 2004, — он очень поможет нам в работе. А уж вооружившись столь фундаментальными знаниями, мы приступим к работе над сайтом.

Он будет совсем простеньким, наш первый сайт. Ведь для нас сейчас главное — научиться, не так ли?



Глава 1

Современные интернет-технологии

Каждая практика должна предваряться теорией. Без теории никуда, ведь, прежде чем приниматься что-то делать, нужно выяснить, зачем, как и почему именно так это "что-то" делается. А иначе у нас ничего не получится.

Так и с интернет-технологиями. Без необходимого багажа знаний мы просто ничего не сможем сотворить (по крайней мере, ничего стоящего у нас точно не получится). Так давайте же отключим компьютер — пусть отдохнет! — и почитаем.

Основные принципы работы Интернета

Сначала мы поговорим о том, что такое Интернет и как он работает — рассмотрим некоторые общие вопросы.

Что такое Интернет. Сервисы Интернета

И первый же вопрос, на который нам нужно получить ответ, — это что, собственно, такое Интернет и что он может нам дать. А разбором принципов его работы мы займемся далее.

Итак, *Интернет* — это всемирная компьютерная сеть. (Вы уже наверняка это знаете.) Ее, кстати, так часто и называют: Всемирная Сеть, или даже просто Сеть с большой буквы. Протянутая по всему земному шару паутина медных проводов, волоконно-оптических линий и радиоканалов, связывающих друг с другом многочисленные компьютеры, — вот что такое Интернет. Разумеется, все это подчиняется общим стандартам (о которых мы поговорим позже), а иначе эта суперсеть просто не будет работать.

Если же быть совсем точным, то Интернет — это не единая сеть, а совокупность более мелких сетей, связанных друг с другом общими каналами и стандартами. Таких сетей превеликое множество: огромные территориальные сети, раскинувшиеся на целые области, штаты и государства, и ведомственные сети,

объединяющие родственные организации, и локальные компьютерные сети отдельных организаций, и так называемые кампусные сети — сети, объединяющие компьютеры одного или нескольких близлежащих районов города. Благодаря проложенным между ними каналам высокоскоростной связи, они составляют единое целое, имя которому Интернет.

Даже частные пользователи, подключающиеся к Интернету по модему, выделенной линии или поддерживающему такую возможность сотовому телефону, тоже, по сути дела, являются частью Сети. Так что когда мы включаем наш модем и дозваниваемся до нашего *интернет-провайдера* (организации, предоставляющей пользователям доступ в Интернет), то приобщаемся к единому целому. А что, разве это не повод для законной гордости?

Сеть Интернет имеет одну замечательную особенность — она очень устойчива к сбоям. Так, если где-то порвется провод, соединяющий два участка (или, как говорят профессионалы-сетевики, сегмента) Сети, мы этого не заметим. А все потому, что данные, которые мы запрашиваем, пойдут в этом случае по другому каналу. Специалисты говорят, что Интернет децентрализован — он не имеет единого центра, из которого ведется управление пересылкой данных, поэтому в случае аварии автоматически переконфигурируется и продолжает нормально работать.

Еще одна замечательная особенность Интернета — его глобальность, всемирность. Не вставая из-за компьютера, мы можем совершить путешествие по всему миру, побывать в США, Австралии, Германии, Зимбабве, Огненной Земле и даже в Антарктиде (да, и туда протянулись вездесущие провода!). Для этого нужно всего лишь набрать нужный нам адрес.

Ну что ж, что такое Интернет, мы выяснили. Теперь совершим небольшое путешествие в прошлое и посмотрим, как все начиналось.

Интернет имеет достаточно долгую и бурную историю. Он появился еще в первой половине 70-х годов XX века, когда американское Министерство обороны финансировало проект создания компьютерной сети, устойчивой к сбоям. Разумеется, создавалась эта сеть для нужд обороны, да и название имела другое — ARPANET. Позднее же, в начале 80-х, эта сеть отошла к ученым, а военные приступили к созданию другой сети, которой пользуются до сих пор. И в то же самое время ARPANET был переименован в Internet, или, если по-русски, Интернет.

Первоначально, еще во времена ARPANET, эта сеть использовалась для пересылки электронной почты и обмена файлами. Web-странички, ради которых мы, в основном, и путешествуем по Сети, появились только в конце 80-х. Именно тогда Интернет и "пошел в народ", перестав быть сетью ученых и превратившись в сеть для всех.

В Россию, точнее, в СССР, Интернет пришел в 1990 году, но популярность среди широких масс компьютерщиков приобрел только в середине 90-х.

В настоящее время в России миллионы людей пользуются услугами Сети для общения и поиска информации.

Раз уж мы заговорили об услугах, предоставляемых Интернетом, или, как говорят профессионалы, *сервисах* Интернета, то давайте узнаем о них побольше. В конце концов, нам ими пользоваться.

Итак, самый старый и самый популярный до сих пор сервис Интернета — это электронная почта (e-mail). Ежедневно в мире отправляются и принимаются сотни миллионов электронных писем, и это количество в будущем будет только увеличиваться. В самом деле, электронная почта доступна, удобна, быстра и бесплатна, в отличие от почты "бумажной", которую пользователи Интернета уже успели презрительно прозвать "улиточной" (по-английски — snail mail). Конечно, эти доступность, удобство, быстрота и бесплатность несут некоторые неудобства, вроде "спама" — несанкционированных рекламных рассылок, но с ними можно бороться.

Еще один сервис Интернета, почти такой же старый, как почта, — это пересылка файлов. Пользователи Интернета называют его FTP (File Transfer Protocol, протокол передачи файлов; почему так — мы узнаем чуть позже). Сейчас FTP уже не имеет той популярности, как на заре существования Интернета, но все еще часто используется для пересылки файлов.

Третий сервис Интернета, который мы подробно рассмотрим, — это Всемирная Паутина, или WWW (World Wide Web, повсеместно протянутая паутина), или просто Web. Это и есть те самые Web-страницы и Web-сайты, которые мы просматриваем в Web-обозревателе. Пожалуй, самый впечатляющий сервис, собственно, и приведший к тому, что сеть ученых стала сетью для всех и получила такую популярность. Вот о нем-то и пойдет речь в этой книге.

Осталось упомянуть еще о нескольких сервисах Интернета, не имеющих такой широкой популярности.

- *Новости.* Чем-то этот сервис похож на электронную почту: пользователь пишет письмо в так называемую *группу новостей* — своего рода электронную доску объявлений. Другие пользователи прочитывают это письмо и пишут ответы, которые помещаются на эту же самую "доску". Совокупность первоначального письма и ответов на него образуют *обсуждение* (по-английски — thread), участие в котором может принять любой пользователь, подписавшийся на эту группу новостей. Когда-то сервис новостей был весьма популярен, но сейчас постепенно сдает свои позиции.
- *Потоковое вещание.* Это своего рода теле- и радиовещание через Интернет появилось совсем недавно, несколько лет назад, но быстро набирает популярность. Единственный серьезный недостаток — необходимость иметь достаточно быстрый канал — сейчас перестает быть актуальным.
- *Интернет-пейджеры.* Этот сервис также похож на электронную почту: пользователи пересылают друг другу короткие "записки" по аналогии

с обычным пейджером. Интернет-пейджеры работают, как правило, быстрее, чем обычная электронная почта, и временами создают иллюзию непосредственного общения. В качестве примера можно вспомнить популярнейший ICQ и его менее известных "коллег": Miranda, Odigo и пр.

- *Чаты* (от английского chat — "болтовня"). Это своего рода "разговор" через Интернет, еще более напоминающий непосредственное общение. Пользователь набирает на клавиатуре текст, который в мгновение ока пересылается его собеседнику или собеседникам. По популярности чаты превосходят интернет-пейджеры и приближаются к WWW.

Ну, вот и все. Совсем уж малоизвестные и узкоспециализированные сервисы Интернета мы рассматривать не будем. В конце концов, сведения о них (как и о многом другом) можно найти в том же самом Интернете. А тема этой книги совсем другая.

Клиенты и серверы

Продолжим наше погружение в электронные дебри Интернета. На этот раз речь пойдет о двух разновидностях программ, с помощью которых предоставляются интернет-услуги.

В самом деле, каким образом мы пользуемся всем тем богатством, что дает нам Всемирная сеть? С помощью особых программ! Это Web-обозреватель, клиент электронной почты, программа просмотра интернет-телевидения и интернет-радио, ICQ и "чатилка". Все они очень хорошо нам знакомы.

Но программ, используемых для предоставления нам сервисов Интернета, гораздо больше. И очень многие из них нам, если так можно сказать, "не видны". То есть мы не общаемся с ними напрямую. Вообще, существуют две принципиально разные категории интернет-программ. И сейчас мы их рассмотрим.

Программы, относящиеся к первой категории, взаимодействуют непосредственно с пользователями Интернета и помогают им получать различную информацию: электронные письма, Web-страницы, сообщения интернет-пейджеров, чатов и пр. Это Web-обозреватели, клиенты электронной почты, чатов, интернет-пейджеры — все те, с которыми мы имеем дело на своих компьютерах. Такие программы называются программами-*клиентами* (а компьютеры, на которых они работают, — наши с вами компьютеры! — *клиентскими*).

Информация, с которой мы работаем посредством программ-клиентов, все эти Web-сайты, письма, звуковые и видеофайлы, хранится на других компьютерах — *серверных*. За выдачу ее клиентским программам, а значит, и нам, отвечают программы, относящиеся ко второй категории, — *серверы*. Для каждого сервиса Интернета существует свой класс серверов (и, как мы уже знаем, клиентов): Web-серверы, серверы электронной почты, чата, интернет-пейджеров, потокового вещания и пр.

Примечание

Очень часто понятие "сервер" распространяется и на серверный компьютер, и на саму программу-сервер. Это, вообще-то, неправильно, так как на одном серверном компьютере может быть установлено несколько программ-серверов, но вошло в практику.

Теперь давайте поговорим подробнее о том, как же клиенты взаимодействуют с серверами. Причем процессы приема и отправки данных мы рассмотрим отдельно.

В общем, процесс получения информации от сервера клиентов включает пять шагов.

1. Пользователь запрашивает с помощью программы-клиента некую информацию.
2. Клиент устанавливает соединение с сервером и посылает тому особый информационный блок, называемый *клиентским запросом*. Структура этого запроса жестко стандартизирована, чтобы сервер его понял.
3. Сервер принимает запрос и расшифровывает его.
4. Сервер извлекает нужный клиенту файл или фрагмент данных, записанных в файле, и посылает его клиенту в виде другого информационного блока — *серверного ответа*. Разумеется, этот ответ также жестко стандартизирован. Если же нужных данных нет, или сервер почему-то не смог понять клиентский запрос, он возвращает в составе ответа *сообщение об ошибке* — особый информационный блок, содержащий описание возникшей ошибки.
5. Клиент получает ответ от сервера, расшифровывает его и выдает полученную информацию пользователю. Если получено сообщение об ошибке, клиент уведомляет об этом пользователя либо предпринимает какие-то действия самостоятельно. После принятия ответа от сервера клиент разрывает соединение с ним.

Процесс отправки клиентом данных серверу включает примерно те же пять шагов.

1. Пользователь каким-то образом передает программе-клиенту отправляемую информацию.
2. Клиент устанавливает соединение с сервером и посылает тому отправляемую информацию в составе клиентского запроса. При этом отправляемая информация, как правило, особым образом шифруется.
3. Сервер принимает запрос, расшифровывает его и извлекает отправленную информацию.
4. Сервер записывает отправленную клиентом информацию в файл. После этого в случае успешной записи он отправляет клиенту в составе ответа

так называемое *подтверждение* — особый информационный блок, сообщающий о том, что все прошло нормально. В случае неуспешной записи отправляется сообщение об ошибке.

5. Клиент получает ответ от сервера, расшифровывает его и уведомляет пользователя об успешной или неуспешной отправке данных либо предпринимает какие-то действия самостоятельно. После принятия ответа от сервера клиент разрывает соединение с ним.

Весь процесс "общения" клиента и сервера, начиная с отправки клиентом запроса и заканчивая принятием им ответа от сервера, называется *сеансом.*

Ранее было сказано, что перед отправкой клиентом запроса серверу, то есть перед началом сеанса, клиент должен соединиться с сервером, иначе говоря — установить *соединение.* Это соединение длится ровно столько времени, сколько нужно для завершения сеанса, и поэтому называется *сеансовым*, или *временным.*

Каждое соединение требует компьютерных ресурсов. Серверу нужно хранить в памяти сведения о клиенте, установившем это соединение, а если таких соединений было установлено много (так часто и бывает), то памяти расходуется очень много. Поэтому сеансовые соединения имеют большое преимущество — они длятся очень недолго, ровно столько, сколько нужно для отправки серверного ответа, после чего разрываются, и отведенная им компьютерная память автоматически освобождается.

Но сеансовые соединения имеют огромнейший недостаток — с их помощью начать сеанс обмена данными может только клиент. А в случае, например, чата или интернет-пейджера серверу самому приходится начинать сеансы, чтобы отправить новые сообщения "отдыхающим" клиентам. В этом случае используются *постоянные*, или *многосеансовые*, подключения с другим сценарием работы.

1. Клиент устанавливает постоянное соединение с сервером.
2. Клиент и сервер обмениваются данными. При этом сеанс обмена может быть начат как клиентом, так и сервером.
3. Клиент разрывает соединение с сервером. Обмен данными окончен.

Здесь нужно обратить внимание на то, что соединение устанавливается только клиентом. Сервер же работает через уже установленное соединение.

Как правило, серверные компьютеры — настоящие монстры, содержащие несколько процессоров, огромные дисковые массивы впечатляющей емкости, мощные средства для подключения к Интернету и специальное программное обеспечение, у которого достаточно "сил", чтобы управлять всей этой мощью. Все в них нацелено на то, чтобы обслужить как можно больше клиентов, обработать как можно больше запросов, чтобы пользователи получили запрошенную информацию за приемлемое время. Но часто, если

клиентов и запросов оказывается слишком много, ресурсов серверного компьютера не хватает, начинаются проблемы. Они могут проявляться в том, что сервер просто отказывается обслужить "лишних" клиентов, предлагая им подождать немного, когда нагрузка немного снизится, а то и в том, что могучий серверный компьютер просто-напросто "зависает". Такое тоже случается, и не так уж редко.

Ну да не будем о грустном! Не стоит начинать знакомство с таким притягательным миром интернет-технологий со столь печальных вещей, как системные сбои. Чем их меньше и чем реже они случаются, тем лучше для всех нас.

Итак, мы только что познакомились с особой *архитектурой* (принципом построения компьютерных систем), называемой *двухзвенной*, или архитектурой *"клиент-сервер"*. Эта архитектура используется для реализации практически всех современных интернет-сервисов и пока что себя оправдывает.

Примечание

Некоторые интернет-сервисы, в частности, так называемые *файлообменные сети* (Napster, Gnutella, Kazaa и пр.), используют принципиально другую архитектуру — *однозвенную*. Здесь все компьютеры, подключенные к Интернету и реализующие этот сервис, фактически равны между собой; любой из них может выступать в роли как клиентского (запрашивать информацию у других компьютеров), так и серверного (предоставлять хранящуюся на нем информацию другим компьютерам). Само собой, здесь используется особое программное обеспечение, имеющее функции и клиента, и сервера.

Протоколы

Люди, чтобы понимать друг друга, должны разговаривать на одном языке. Точно так и с компьютерами, подключенными к сети, неважно какой — всемирной или локальной. Обмен данными по этим сетям должен проходить по единым стандартам, иначе начнется новое вавилонское столпотворение.

Стандарт, по которому кодируются данные для отправки по сети, называется *протоколом*. В Интернете для обмена данными используются несколько протоколов, которые мы здесь вкратце рассмотрим.

Самый главный, если так можно сказать, протокол Интернета — это TCP/IP (Transfer Control Protocol/Internet Protocol, протокол управления передачей/протокол Интернета). Это так называемый *протокол низкого уровня*, определяющий только самые основные параметры передаваемых данных: длина отдельных порций (*пакетов*) данных, способ кодирования, указания адресов получателя и отправителя, а также защита от ошибок. Можно сказать, что TCP/IP занимается исключительно передачей данных по каналам Интернета, не вникая, что же именно он передает.

На протоколе TCP/IP базируются другие протоколы, уже *высокого уровня*. Эти протоколы описывают формат клиентских запросов и серверных ответов: особые команды, пересылаемые клиентом серверу при запросе или передаче данных, и способ представления передаваемой информации. Кодированием передаваемой информацией особым помехоустойчивым кодом, разбиением ее на пакеты и собственно передачей занимается "чернорабочий" TCP/IP.

Примечание

Строго говоря, существуют еще *протоколы физического уровня*, располагающиеся "ниже" даже TCP/IP. Они определяют электрические параметры сигнала, кабелей, разъемов и пр.

Каждый сервис Интернета использует свой собственный высокоуровневый протокол (а то и несколько, предназначенных для разных задач или разработанных конкурирующими организациями). Давайте рассмотрим протоколы, с которыми мы столкнемся в будущем.

Начнем мы, конечно, с WWW. Для передачи данных Всемирная Паутина использует протокол HTTP (HyperText Transfer Protocol, протокол передачи гипертекста). Он задает набор команд для запроса данных и управления ими, пересылаемых клиентом (Web-обозревателем) Web-серверу, и способы представления пересылаемых в обе стороны данных. Пожалуй, это самый широкоизвестный протокол Интернета — всем более-менее грамотным интернетчикам знакомы эти четыре буквы.

Сервис пересылки файлов FTP использует протокол, который так и называется — FTP. Он также определяет набор команд для управления файлами на сервере (загрузка, помещение на сервер, копирование, перемещение, удаление и т. д.) и способы кодирования файлов для пересылки по каналам связи. В этом смысле протоколы HTTP и FTP весьма похожи.

А вот электронная почта использует целых два протокола. Первый протокол — SMTP (Simple Mail Transfer Protocol, простой протокол пересылки почты) — используется для пересылки почты клиентом серверу. Для получения же почты от сервера клиент общается с ним по протоколу POP3 (Post-Office Protocol, протокол почты).

Замечание

Нужно также упомянуть протокол IMAP (Internet Message Access Protocol, протокол доступа к почте Интернета), применяемый также для получения клиентом почты от сервера. По сравнению с более старым POP3 он предоставляет больше возможностей, но распространен не так широко.

Сервис новостей использует для работы протокол NNTP (Network News Transfer Protocol, протокол передачи сетевых новостей). Остальные сервисы используют свои протоколы. Но мы не будем на них останавливаться.

С понятием протокола очень тесно связано понятие порта TCP/IP. *Порт* — это своего рода воображаемый канал, по которому передаются данные с использованием одного из протоколов высокого уровня. Можно сказать, что любой канал передачи данных Интернета разделен на 65 535 небольших пронумерованных "канальчика" — именно столько портов предусматривает протокол TCP/IP.

Каждый существующий протокол высокого уровня использует для передачи данных свой собственный порт (так называемый *порт по умолчанию*). В табл. 1.1 перечислены некоторые протоколы и "занимаемые" ими порты.

Таблица 1.1. *Порты TCP/IP, используемые по умолчанию для передачи данных некоторых протоколов высокого уровня*

Протокол	Используемый порт
HTTP	80
FTP	21
SMTP	25
POP3	110

Но почему такое странное название — "порт по умолчанию"? Давайте разберемся.

Дело в том, что все более-менее серьезные серверы предоставляют возможность изменить порт, используемый протоколом, который они обслуживают, на другой. Например, Web-сервер может быть настроен так, чтобы использовать для "общения" с клиентами не 80-й порт, а, скажем, 8000-й. (Автору этой книги время от времени встречаются Web-серверы, настроенные таким образом.) Это вполне возможно, но не рекомендуется, чтобы не обескураживать пользователей, и поэтому применяется очень редко, только в крайних случаях.

Если же сервер специально не настраивать на использование другого порта, то он будет использовать именно порт по умолчанию. Так изначально задано в его настройках.

Интернет-адреса

Теперь давайте поговорим о том, каким образом идентифицируются компьютеры, подключенные к Интернету. А именно — об интернет-адресах.

Интернет-адрес — это уникальное числовое или строковое значение, позволяющее точно идентифицировать компьютер в Сети. Именно по интернет-адресу клиент находит нужный ему сервер. Именно по интернет-адресу происходит отправка данных. Интернет-адрес — это своего рода "имя" сервера.

Изначально, на заре эпохи Интернета, в качестве интернет-адреса использовался *IP-адрес* — числовое значение, идентифицирующее компьютер для протокола TCP/IP. Как мы помним, TCP/IP разбивает передаваемую информацию на пакеты. Так вот: в каждом таком пакете содержатся IP-адреса компьютера-отправителя и компьютера-получателя.

IP-адрес замечательно подходит для компьютеров, но очень плохо — для людей. Он имеет такой вид:

```
192.168.1.10
```

Не очень-то наглядно, правда? Именно поэтому с расширением Интернета была введена в строй новая система интернет-адресов, которой мы пользуемся до сих пор. Это так называемые доменные адреса, о которых стоит поговорить подробно.

Но прежде чем мы начнем разговор о доменных адресах, давайте выясним, что такое домен. *Домен*, или *доменная зона*, — это участок Интернета, созданный для удобства управления им. Такой участок может быть крупным или мелким или вообще состоять из одного компьютера. Каждый домен обозначается строкой текста, состоящей из английских букв.

Структура доменов похожа на матрешку: мелкие домены "вложены" внутрь крупных, а крупные, в свою очередь, — внутрь гигантских. Гигантские домены называются *доменами верхнего уровня*, а вложенные в них более мелкие — *доменами нижнего уровня*.

Домены верхнего уровня бывают интернациональными и национальными. *Интернациональные домены* объединяют компьютеры по какому-то признаку; к ним относятся домены com (коммерческие серверы), edu (образовательные), mil (военные), org (организации, не занимающиеся компьютерами и Интернетом), net (организации, занимающиеся компьютерами и Интернетом) и некоторые другие. *Национальные домены* объединяют компьютеры по территориальному признаку и выдаются целым странам; это домены us (США), uk (Великобритания), fr (Франция), de (Германия), ru (Россия) и др.

Что касается доменов нижнего уровня, то они выдаются, как правило, отдельным организациям или, опять же, по территориальному признаку. Их текстовое обозначение часто совпадает с названием этой организации или района.

Если теперь записать обозначения всех доменов, в которых находится нужный нам компьютер, в порядке от более мелких к более крупным, разделив их точками, мы получим *доменное имя* этого компьютера. Так, если у нас сам компьютер имеет имя comp45, отдел, в котором он стоит, — buh (бухгалтерия), организация, включающая этот отдел, — department, а страна — ru (Россия), то мы получим такое доменное имя:

```
comp45.buh.department.ru
```

Согласитесь — запомнить это гораздо проще, чем невразумительный IP-адрес.

Да, но проблема в том, что протокол TCP/IP не понимает доменные имена! Что делать? Как преобразовать доменное имя в понятный ему IP-адрес?

Для этого используются особые программы, называемые *серверами DNS* (Domain Name System, система сетевых имен). Занимаются они тем, что принимают от компьютеров, которым нужно куда-то отправить данные по протоколу TCP/IP, доменные имена и возвращают соответствующие этим именам IP-адреса. Такие серверы DNS имеются в каждом домене; кроме того, несколько самых мощных в мире серверов DNS находятся как бы "выше" всех доменов, даже доменов верхнего уровня. И всем им хватает работы.

Но вернемся к доменным именам и рассмотрим некоторые детали, которые помогут нам в дальнейшем.

Доменное имя идентифицирует сам серверный компьютер, а не выполняющуюся на нем программу-сервер. А таких серверов на одном компьютере может быть несколько: Web, FTP, почта, чат и пр. Чтобы обратиться к нужному серверу, не беспокоя остальных, перед доменным именем указывается обозначение протокола, по которому этот сервер "общается" с клиентами. Вот так (обозначение протокола выделено полужирным шрифтом):

http://comp45.buh.department.ru

ftp://comp45.buh.department.ru

В первом случае мы обращаемся к Web-серверу, а во втором — к серверу FTP, находящимся на одном и том же компьютере comp45.buh.department.ru.

Примечание

Существует, правда, возможность дать одному компьютеру сразу несколько доменных имен и даже IP-адресов. Но используется это нечасто и в особых случаях. В дальнейшем для простоты мы будем считать, что одно доменное имя (и один IP-адрес) — это один компьютер.

Если же какой-либо сервер использует порт, отличный от порта по умолчанию, то номер нужного порта записывается после доменного имени серверного компьютера и отделяется от него двоеточием. Вот так (номер порта выделен полужирным шрифтом):

http://comp45.buh.department.ru:**8000**

Ну вот, с основными принципами работы Интернета мы ознакомились. Теперь давайте сосредоточимся на WWW — в основном, именно этим сервисом мы будем пользоваться на протяжении всей книги.

Основные понятия WWW

Здесь мы выясним все о Web-страницах и Web-сайтах, узнаем, чем сайт отличается от страницы, поговорим о Web-клиентах и Web-серверах и уясним несколько новых понятий.

Web-страницы и Web-сайты

Что такое *Web-страница*? Ответить на этот вопрос могут многие. Это интернет-документ, предназначенный для распространения через Интернет посредством сервиса WWW. А если уж говорить по-простонародному, это то, что показывает в своем окне программа для просмотра Web-страниц — *Web-обозреватель*.

С технической точки зрения, Web-страница — это текстовый файл, содержащий собственно текст, некоторые команды его форматирования и сохраненный на жестких дисках серверного компьютера. Получив от Web-обозревателя запрос по протоколу HTTP, *Web-сервер* (серверная программа, обеспечивающая работу сервиса WWW) извлекает этот файл и отправляет его Web-обозревателю.

Но как Web-обозреватель дает понять Web-серверу, какая Web-страница ему нужна? Очень просто — он пересылает в составе запроса имя и полный путь файла, в котором она сохранена. Скажем, вот так:

```
http://comp45.buh.department.ru/somepage.html
```

Этот запрос заставит Web-сервер извлечь и отправить Web-обозревателю файл somepage.html.

А что такое *Web-сайт*? Это набор Web-страниц, подчиненных общей тематике и объединенных в единое целое (как — будет рассказано в *главе 2*). Web-сайт также сохраняется на жестких дисках серверного компьютера в виде набора файлов, находящихся в папках. (Конечно, папки использовать необязательно, но так удобнее, особенно если файлов много и все они разных типов.) Как видим, чисто технических отличий у Web-страницы и Web-сайта не слишком много.

А теперь поговорим о некоторых технических деталях сохранения сайта на дисках серверного компьютера.

Прежде всего, для хранения всех файлов, составляющих сайт (или Web-страницу, если рассматривать ее как "вырожденный" случай сайта), на диске серверного компьютера создается особая папка, называемая *корневой*. Все, что не находится в этой папке, автоматически исключается Web-сервером из состава сайта.

Все файлы, составляющие Web-сайт, да и Web-страницу тоже, если рассматривать ее как "вырожденный" случай сайта, сохраняются на дисках сер-

верного компьютера в особой папке. Эту папку создает для сайта человек, занимающийся настройкой и обслуживанием программы Web-сервера (или же всего серверного компьютера), — *администратор*. Полный путь данной папки заносится в настройки Web-сервера, чтобы последний смог ее найти. Все содержимое сайта должно находиться в корневой папке.

Примечание

Вообще-то, все серьезные программы Web-серверов предоставляют возможность создания так называемых *виртуальных папок*. Виртуальная папка — это папка, находящаяся в любом месте файловой системы компьютера, но считаемая Web-сервером частью сайта. В дальнейшем и мы будем считать виртуальные папки частью сайта, если специально не сказано иначе.

Когда Web-обозреватель присылает Web-серверу запрос вида:

```
http://www.somesite.ru/somepage.html
```

Web-сервер находит файл somepage.html в корневой папке сайта и отправляет его Web-обозревателю. Если же Web-обозревателю понадобится файл, находящийся не в самой корневой папке сайта, а в одной из вложенных в нее папок, он должен прислать такой запрос:

```
http://www.somesite.ru/somefolder1/somefolder2/somepage.html
```

В этом случае Web-сервер отправит Web-обозревателю файл somepage.html, находящийся в папке somefolder1/somefolder2, вложенной, опять же, в корневую папку сайта.

Примечание

Для обращения к файлу, находящемуся в виртуальной папке, используется аналогичный запрос:

```
http://www.somesite.ru/somevirtualfolder1/somepage.html
```

Так, все прекрасно, все замечательно и все исключительно ясно! Но ведь мы крайне нечасто набираем в поле ввода интернет-адреса Web-обозревателя такие запросы, указывающие непосредственно на нужную нам Web-страницу. Много чаще наши запросы выглядят чуть "скромнее", например, так:

```
http://www.somesite.ru
```

То есть они не указывают на файл. Как поступает Web-сервер в таком случае?

Дело в том, что одна из страниц сайта задается в качестве так называемой *страницы по умолчанию*. Именно она отправляется Web-обозревателю, если он не прислал запрос на конкретную страницу (и вообще на конкретный файл). Имя файла этой страницы задается администратором Web-сервера в его настройках — как правило, default.htm[l] или index.htm[l].

И если мы наберем в поле ввода интернет-адреса нашего любимого Web-обозревателя нечто, похожее на

```
http://www.somesite.ru
```

Web-обозреватель выведет нам страницу default.html, хранящуюся в корневой папке сайта.

Ранее мы рассмотрели так называемые *абсолютные* интернет-адреса, содержащие как адрес самого Web-сервера, так и имя файла нужной Web-страницы. Но интернет-адрес файла можно также указать относительно уже открытой в Web-обозревателе (*текущей*) страницы:

```
page2.html
```

Получив этот запрос, Web-сервер отправит нам страницу page2.html, находящуюся в той же папке, что и текущая. Отметим, что имени сервера этот адрес не включает, так как подразумевается, что файл page2.html находится на том же сервере, что и файл текущей страницы.

```
folder/page2.html
```

Этот запрос заставит Web-сервер искать страницу page2.html в папке folder, вложенной в папку, в которой хранится текущая Web-страница.

```
../folder2/page3.html
```

А этот запрос вернет нам страницу page3.html из папки folder2, находящейся в той же папке, что и папка, в которой хранится текущая Web-страница.

Осталось только сказать, что интернет-адрес, указывающий имя файла относительно файла текущей страницы и не содержащий имени сервера, так и называется — *относительным*.

А теперь давайте поговорим немного подробнее о программах Web-обозревателей и Web-серверов.

Web-обозреватели

Мы уже знаем, что Web-обозреватели — это программы для просмотра Web-страниц и Web-сайтов. Основная их задача — это отправить Web-серверу корректно, в соответствии со всеми стандартами сформированный клиентский запрос, принять серверный ответ и вывести полученную страницу на экран. Для этого окно Web-обозревателя содержит поле ввода интернет-адреса и область, в которую собственно выводится Web-страница. (Разумеется, оно также содержит заголовок, меню и панели инструментов, как и многие окна приложений Windows.)

А теперь — нечто новенькое. После получения от сервера файла Web-страницы (и всех связанных с ней файлов, так как страница может состоять из множества файлов; подробнее об этом мы поговорим в *главе 2*) Web-

обозреватель сохраняет их на жестком диске клиентского компьютера в особой области, называемой *кэшем*. Этот кэш может иметь вид как обычной папки (кэш Microsoft Internet Explorer или Opera), так и большого файла (кэш Netscape Navigator или Mozilla).

Зачем это нужно? Да хотя бы затем, чтобы мы смогли впоследствии просмотреть данную страницу, не подключаясь к Интернету. Все современные Web-обозреватели поддерживают так называемый *автономный режим* (offline mode), когда они отображают только те страницы, что находятся в кэше. (Кстати — исключительно удобная вещь!) Если же мы попытаемся просмотреть страницу, которой нет в кэше, Web-обозреватель предложит нам подключиться к Интернету и загрузить ее.

Теперь познакомимся с программами Web-обозревателей, имеющими в настоящее время наибольшую популярность. Все они, в общем, следуют одним и тем же стандартам и отличаются друг от друга только поддержкой деталями, не оговоренными в этих стандартах, и удобством для пользователей.

Настоящий король виртуальных просторов — это, конечно, Microsoft Internet Explorer. Он имеется на любом компьютере, работающем под управлением Windows (что, как говорят злые языки, и обусловило его популярность). Однако это очень мощная, быстрая, весьма нетребовательная к ресурсам и исключительно удобная программа. Автор данной книги для просмотра Web-страниц пользуется именно Internet Explorer. В настоящее время доступна версия 6.0 и, по слухам, разрабатывается новая версия, которая войдет в состав новой Windows, хотя фирма Microsoft, по своему обыкновению, хранит по этому поводу молчание.

Второе место по популярности занимает норвежская программа Opera, разработанная одноименной фирмой. Эта достаточно мощная и очень быстрая программа, поддерживающая все официальные Web-стандарты, тем не менее, весьма требовательна к системным ресурсам, особенно при отображении сложных Web-страниц. Кроме того, она является платной, в бесплатной же версии показывает рекламу. Последняя вышедшая в свет версия носит номер 7.60 и, скорее всего, после выхода книги устареет, так как новые версии Opera появляются очень часто.

Некогда властелин WWW Netscape Navigator сейчас не популярен — им пользуются менее 1 % интернетчиков. Хотя последняя версия Navigator под номером 7.2 выглядит весьма неплохо, поддерживает все современные стандарты Интернета, корректно отображает большинство Web-страниц и не очень требовательна к системным ресурсам. Но все равно Navigator по многим параметрам проигрывает и Internet Explorer, и Opera, к тому же, работа над ним давно прекращена.

Самый "младший" Web-обозреватель носит имя Mozilla. Эта программа распространяется бесплатно, более того, ее исходные тексты открыты для изу-

чения и модификации. Она построена на том же программном ядре, что и Navigator 7, а точнее наоборот — Navigator 7 построен на основе Mozilla. (Собственно, Mozilla и создавался для обкатки нового программного ядра Navigator, но в дальнейшем вырос в самостоятельный продукт.) Этот новичок весьма неплох, поддерживает все Web-стандарты, нетребователен к системным ресурсам, довольно быстр и имеет множество интересных и весьма полезных возможностей, которыми пока не может похвастаться ни один из его конкурентов. Пока что он не очень популярен, но в дальнейшем, возможно, еще себя покажет. Последняя версия, доступная в момент написания книги, — 1.7.

Нужно еще сказать, что все перечисленные ранее программы — на самом деле настоящие программные пакеты. Кроме собственно Web-обозревателей, они содержат уйму других программ: почтовые клиенты, клиенты новостей, проигрыватели мультимедийных файлов, клиенты чатов, интернет-пейджеры и даже утилиты для разработчиков Web-сайтов. Такая практика — объединение в одном пакете множества программ для работы с Интернетом — существует уже более десяти лет, и только недавно ей положили конец разработчики Mozilla. Они выделили из состава этого пакета сам Web-обозреватель (плюс несколько мелких утилит) и назвали его Firefox. В настоящее время его версия 1.0 постепенно набирает популярность, оттесняя и своего "прародителя" Mozilla (тем более что разработчики в дальнейшем собираются вообще "прикрыть" последний), и даже Internet Explorer.

Совсем недавно фирма Apple, производящая широко известные в узких кругах компьютеры Macintosh, объявила о выпуске своего собственного Web-обозревателя Safari. Утверждается, что это самый быстрый в мире Web-обозреватель, быстрее даже Opera. Пока трудно сказать, что в действительности этот Safari собой представляет; подождем версии для Windows (которая вроде бы вскоре обещается) и посмотрим. Исходные коды его также открыты для изучения и модификации.

В настоящее время просторы WWW "бороздят" практически только пять перечисленных ранее программ (шесть, если считать Firefox отдельным проектом). Существует, однако, еще несколько малоизвестных Web-обозревателей, а также довольно многочисленная когорта программ, построенных на основе программного ядра Internet Explorer и расширяющих его возможности. Мы не будем их рассматривать.

Осталось только сказать, что выбор Web-обозревателя — это личное дело каждого. Все они поддерживают одни и те же стандарты (правда, зачастую по-своему) и предоставляют пользователю примерно одинаковый набор возможностей (хотя, не все найдут его удобным). Так что, как в песне поется, "думайте сами, решайте сами".

Web-серверы

Поскольку мы не только пользователи, но и уже наполовину разработчики, нас будут интересовать не только Web-обозреватели, но и Web-серверы. Давайте поговорим и о них.

"Зоопарк" Web-серверов ничуть не меньше "зоопарка" (или, если учесть, что Web-обозреватели жестоко конкурируют друг с другом, "серпентария") Web-обозревателей, так что мы можем подобрать себе программу "по вкусу". И, в отличие от Web-обозревателей, среди Web-серверов нет безоговорочного лидера — даже самые распространенные из них не занимают больше половины рынка.

Начнем наш краткий обзор с двух пакетов фирмы Microsoft: Personal Web Server и Internet Information Server. Оба этих пакета поставляются в составе Microsoft Windows, первая программа — в составе Windows 98 и Me, а вторая — в составе Windows NT, 2000, XP и 2003. Со своими обязанностями они справляются очень хорошо, не транжирят системные ресурсы, легко настраиваются, поддерживают множество передовых интернет-технологий и при надлежащей настройке легко затыкают за пояс конкурентов. Кроме собственно Web-сервера, они содержат также серверы FTP и почты, а также некоторое количество дополнительных программ.

Web-сервер Apache — пожалуй, самый распространенный. Среди его достоинств: полная бесплатность (более того — его исходные тексты открыты), легкость настройки, довольно высокая производительность, хорошая поддержка. По крайней мере, для Web-сайтов с небольшой загрузкой — это идеальный выбор.

В свое время фирма Netscape — разработчик известного Web-обозревателя Navigator — создала и Web-сервер, который так и называется — Netscape Web Server. В смысле производительности и поддержки передовых интернет-технологий он не уступает своему конкуренту от Microsoft, но — увы! — не завоевал его популярности.

Есть еще один весьма примечательный Web-сервер — Sambar. Он поддерживает такое количество интернет-технологий (многие из них — эксклюзивные, больше никем не поддерживаемые), что просто оторопь берет — как же всем этим богатством воспользоваться и куда его применить? Недостатка у Sambar всего два: малая известность и не очень удобная настройка. (Кстати, автор этой книги пользуется именно этим сервером.)

Менее известные и специализированные серверы мы рассматривать не будем, т. к. их довольно много. Поговорим лучше о том, как разместить созданный нами сайт в Интернете.

Публикация Web-сайта в Интернете. Хостинг-провайдеры

Итак, предположим, что мы создали свой сайт (а мы его и создадим, пока будем изучать интернет-технологии по этой книге). Теперь нам нужно сделать так, чтобы все желающие увидеть его собственными глазами, а именно — разместить, или, как говорят опытные интернетчики, *опубликовать* его в Интернете. А значит, нам нужно подключение к Интернету и Web-сервер.

Если наш компьютер подключен к Интернету по скоростному каналу или находится в локальной сети, работающей по тем же стандартам, что и Интернет (так называемый *интранет*), то мы можем просто установить на него Web-сервер и сюда же поместить наш Web-сайт. Это самый простой способ, хотя, конечно, нам придется попутно освоить профессию администратора.

Для тех "счастливчиков", что выходят в Интернет по телефонным каналам (как автор этой книги), существуют три способа донести свое Web-творение до страждущих масс. Давайте перечислим их в порядке от простых к более хлопотным.

Большинство солидных интернет-провайдеров, кроме собственно доступа в Интернет, предоставляют своим клиентам и другие услуги: электронную почту, доступ на свой сайт с новостями, документацией и файловым архивом и пр. Так вот, среди этих "пр." есть и такая услуга, как предоставление на жестких дисках серверного компьютера места для размещения Web-сайтов клиентов. В этом и заключается первый способ: выяснить условия публикации сайта на сервере интернет-провайдера и, следуя этим условиям, опубликовать его.

Если же интернет-провайдер почему-то жадничает, можно прибегнуть ко второму способу. В Интернете существует довольно много серверов, предоставляющих место для сайтов бесплатно. Процесс и в этом случае очень прост: заходим на сайт такого сервера, регистрируемся, выясняем условия публикации сайта и публикуем.

Всем хороши бесплатные серверы: и денег не берут, и позволяют публиковать сайты. Но бесплатного сыра много не бывает. Как правило, объем предоставляемого под сайт дискового пространства сильно ограничен, значит, большой сайт таким образом не опубликуешь. Еще администратор может ограничить количество пользователей, которые могут одновременно зайти на наш сайт. Да и с поддержкой некоторых технологий, используемых для создания Web-сайтов, дело может обстоять не очень хорошо.

Если же нам нужно нечто большее, чем предложения бесплатных серверов, то придется достать свой кошелек и пойти третьим путем — опубликовать сайт на платном сервере. Благо сейчас их довольно много, и услуги их довольно дешевы.

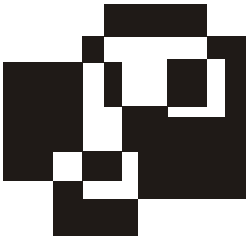
Кстати, организации, предоставляющие место на своих серверах для публикации Web-сайтов, называются *хостинг-провайдерами*.

Что дальше?

Вот и все об интернет-технологиях. Общие вопросы мы рассмотрели, пора приступать к созданию Web-страниц и Web-сайтов.

В следующей главе мы выясним, что же представляют собой Web-страницы и как они создаются. Мы изучим язык написания Web-страниц HTML, узнаем, из чего должна состоять всякая уважающая себя Web-страница, как работать с текстом и графикой и какими средствами можно немного "разукрасить" наши творения.

Но сначала мы определимся, на каком примере будем изучать создание Web-страниц. А именно определим структуру нашего первого Web-сайта.



Глава 2

HTML — язык написания Web-страниц

Вот мы и подошли вплотную к тому, чтобы создать нашу первую Web-страницу. Пока что это будет просто небольшой пример, нужный для того, чтобы изучить принципы создания Web-документов, которые будут нормально отображаться во всех Web-обозревателях. Самим же сайтом мы займемся чуть позже.

На самом деле, в процессе создания Web-страниц нет ничего особо сложного. Уяснив несколько основных терминов и держа под рукой кое-какие справочники и пару программ, можно лепить Web-страницы, как пирожки. Неудивительно, что сейчас профессия Web-дизайнера так широко распространена.

Другое дело — приличный Web-сайт. Его создание начинается с довольно долгого этапа планирования. Нужно продумать его структуру, собрать все необходимые материалы (которые нужно опубликовать в Интернете — ведь именно ради этого и создается сайт) и привести их к виду, поддерживаемому Web-обозревателями. А после создания всех Web-страниц, входящих в сайт, их нужно связать друг с другом и обязательно проверить, нет ли ошибок в этих связках. Ведь вполне может оказаться так, что на какие-то страницы посетитель сайта вообще не сможет попасть — тот еще сюрпризец.

Теперь давайте включим компьютер — он уже достаточно отдохнул — и займемся делом. И начнем мы с изучения языка написания Web-страниц HTML.

Введение в язык HTML

HTML (HyperText Markup Language, язык гипертекстовой разметки) — это особый язык, на котором пишутся Web-страницы. Сейчас мы с ним познакомимся, а заодно попробуем силы в написании первых своих, пока еще совсем простых Web-страниц.

Теги HTML. Форматирование текста

Давайте запустим небольшой текстовый редактор Блокнот, поставляющийся в составе Windows, и наберем в нем вот такой текст:

```
Пример Web-страницы
Это простейшая Web-страничка, созданная в стандартном
↳Блокноте и отображенная в Microsoft Internet Explorer.
```

Сохраним этот текст в файле под именем 2.1.txt. И откроем его в Web-обозревателе Microsoft Internet Explorer, также поставляемом в составе Windows. Мы увидим то, что показано на рис. 2.1.

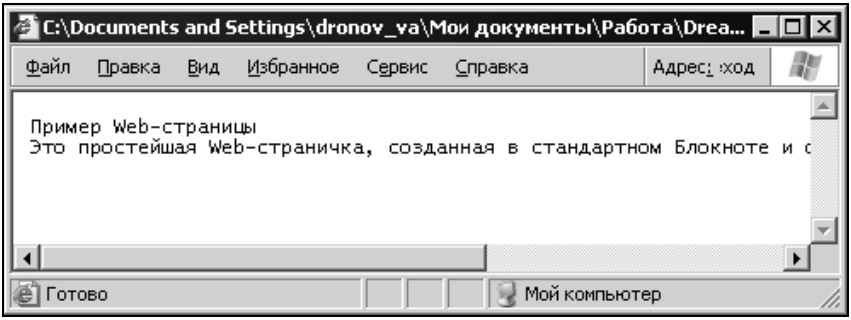


Рис. 2.1. Простой текст, открытый в Internet Explorer

Да, это еще не Web-страница. Это простой текстовый файл, открытый в Web-обозревателе (Web-обозреватели могут открывать также и простые текстовые файлы). Давайте превратим его в настоящую Web-страницу.

Для начала отредактируем файл 2.1.txt, чтобы его содержимое выглядело так (добавленные фрагменты выделены полужирным шрифтом):

```
<P>Пример Web-страницы</P>
<P>Это простейшая Web-страничка, созданная в стандартном
↳Блокноте и отображенная в Microsoft Internet Explorer.</P>
```

После этого сохраним отредактированный текст в другом файле — 2.2.html. (Только когда будем вводить имя файла в стандартном окне сохранения, заключим его в кавычки, иначе Блокнот по доброте душевной добавит расширение txt, и наш файл получит имя 1.1.htm.txt.) Открыв его в Internet Explorer, увидим то, что изображено на рис. 2.2. Видна разница?

Текстовый файл 2.1.txt Internet Explorer вывел "как есть", без всяких изысков. В частности, он не разбил слишком длинный абзац на две строки (хотя это не помешало бы). А все потому, что обычный текст не содержит команд для его форматирования.

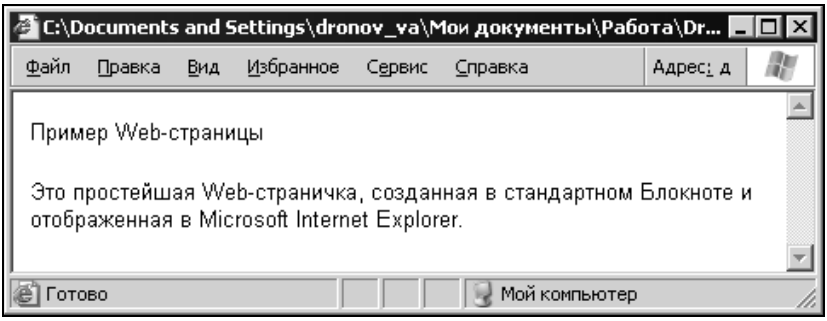


Рис. 2.2. Наша первая Web-страница, открытая в Internet Explorer

Совсем другое дело — код HTML, сохраненный в файле 2.2.html. Internet Explorer сформировал два абзаца — это хорошо заметно на рис. 2.2 — и разбил слишком длинный на строки так, чтобы ширина текста не превышала ширину его окна. А все потому, что он встретил в тексте две знакомые ему команды, описывающие форматирование текста, — так называемые *теги* HTML. Это теги текстового абзаца `<P>` и `</P>`.

В общем случае, теги HTML представляют собой английские слова, заключенные между знаками `<` и `>`. Первый из тегов (в нашем случае — `<P>`) задает начало фрагмента текста, попадающего под действие тега, и называется *открывающим*. Второй тег (`</P>`) отличается тем, что в нем между знаком `<` и собственно наименованием тега стоит символ `/` ("слеш"); он задает конец фрагмента текста и называется *закрывающим*. А сам фрагмент текста, на который оказывает влияние тег, называется его *содержимым*.

Абсолютное большинство тегов HTML являются *парными*, то есть имеют открывающий тег, закрывающий тег и содержимое. Но есть и немногочисленные *одинарные* теги, состоящие только из одного тега вида `<IMG>` и не имеющие содержимого. Мы познакомимся с ними чуть позже.

Исходя из сказанного ранее, мы можем сформулировать определение Web-страницы. Итак, Web-страница — это обычный текстовый файл, созданный в любом простейшем текстовом редакторе (подойдет, например, стандартно поставляющийся в составе Windows Блокнот) и сохраненный с расширением htm или html.

Манипулируя тегами HTML, мы можем форматировать текст нашей первой странички, как хотим. Давайте, например, выделим названия двух упомянутых в нем программ курсивом. Для этого нам будет нужно использовать парный тег `<EM>…</EM>`. Вот так (добавленные теги выделены полужирным шрифтом):

```
<P>Пример Web-страницы</P>
<P>Это простейшая Web-страничка, созданная в стандартном
↳<EM>Блокноте</EM> и отображенная в <EM>Microsoft Internet
↳Explorer</EM>.</P>
```

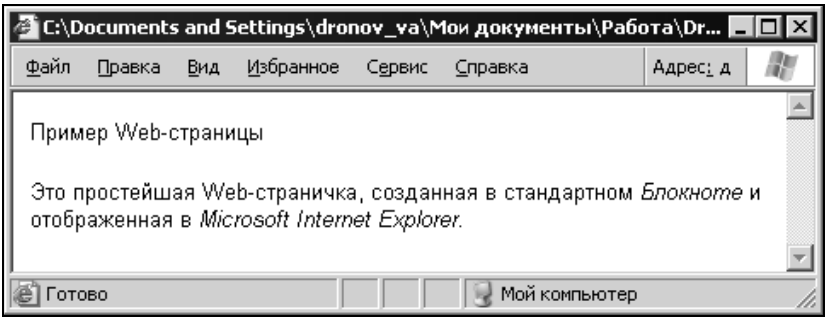


Рис. 2.3. Наша первая Web-страница — названия программ выделены курсивом

Результат этих священнодействий показан на рис. 2.3.

Если мы захотим выделить название фирмы Microsoft полужирным шрифтом, то нам будет нужно использовать парный тег `<STRONG>…</STRONG>`. Вот так:

```
<P>Пример Web-страницы</P>
<P>Это простейшая Web-страничка, созданная в стандартном
↳<EM>Блокноте</EM> и отображенная в <EM><STRONG>Microsoft</STRONG>
↳Internet Explorer</EM>.</P>
```

Здесь мы вложили тег `<STRONG>` внутрь тега `<EM>`. В результате этого слово "Microsoft" будет набрано полужирным курсивом. Вообще, *вложенность* тегов HTML друг в друга — обычное дело, и нам следует к ней привыкнуть как можно быстрее.

Начинающие Web-дизайнеры очень часто допускают такую ошибку: они располагают закрывающие теги не в порядке, не строго противоположном порядку открывающих тегов. Так, приведенный далее код содержит подобную ошибку: теги `</EM>` и `</STRONG>` поменялись местами:

```
<P>Пример Web-страницы</P>
<P>Это простейшая Web-страничка, созданная в стандартном
↳<EM>Блокноте</EM> и отображенная в <EM><STRONG>Microsoft</EM></STRONG>
↳Internet Explorer.</P>
```

Поведение Web-обозревателя, встретившего такой код, непредсказуемо (хотя Internet Explorer славится своим умением исправлять мелкие ошибки Web-дизайнера). Так что стоит запомнить на будущее простое правило: закрывающие теги должны повторяться в порядке, обратном порядку соответствующих им открывающих тегов.

Ну и напоследок давайте дадим нашей странице большой "кричащий" заголовок. Для этого в первом абзаце текста заменим тег `<P>…<P>` на `<H1>…<H1>`:

```
<H1>Пример Web-страницы</H1>
<P>Это простейшая Web-страничка, созданная в стандартном
```

```
↳<EM>Блокноте</EM> и отображенная в <EM><STRONG>Microsoft</STRONG>
↳Internet Explorer</EM>.</P>
```

Окончательный вариант нашей первой страницы показан на рис. 2.4. Видно, что Web-обозреватель сам выделил содержимое тега <H1> очень большим шрифтом, чтобы оно стало заметнее.

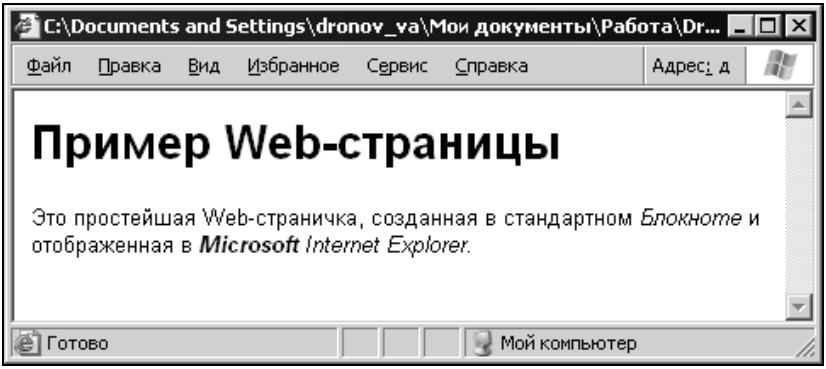


Рис. 2.4. Наша первая Web-страница с заголовком

Кстати, тег <H1> так и называется — тегом заголовка первого уровня. Почему первого? А потому, что стандарт HTML предусматривает целых шесть уровней заголовков, начиная от самого высокого и заканчивая самым низким. Эти заголовки задаются с помощью тегов <H2> (второй уровень), <H3> (третий) и так вплоть до <H6> (шестой, самый низкий уровень). Заголовками первого уровня обычно даются названия страниц целиком, заголовком второго — заголовки отдельных частей страниц и т. д.

HTML-код имеет еще одно преимущество перед обычным текстом. Обычный текст, как мы уже видели на рис. 2.1, отображается так, как написан — со всеми пробелами и разрывами строк. Web-обозреватель даже не удосужился перенести слишком длинную строку, чтобы избавить нас от нудной прокрутки содержимого своего окна по горизонтали.

А в случае кода HTML Web-обозреватель форматирует содержимое тега <P> так, чтобы оно поместилось в его окно и чтобы пользователю не пришлось прокручивать содержимое окна по горизонтали. Web-обозреватель сам разбивает абзац на строки, не обращая внимания на то, как мы разбили его в HTML-коде страницы. При этом все разрывы строк он считает пробелами, что очень удобно для нас.

Предположим, что мы должны отредактировать код HTML нашей страницы 2.2.html вот так:

```
<H1>Пример Web-страницы</H1>
<P>Это простейшая Web-страничка,
```

```
созданная в стандартном
<EM>Блокноте</EM> и отображенная в
<EM><STRONG>Microsoft</STRONG>
Internet Explorer</EM>.</P>
```

Мы разбили слишком длинный второй абзац на несколько более коротких строк — так намного удобнее держать его код перед глазами. (Кстати, давайте и в дальнейшем так делать.) Если мы теперь откроем отредактированную страницу 2.2.html в Web-обозревателе, увидим, что ничего в его восприятии нашего кода не изменилось — см. рис. 2.4.

А теперь давайте немного отдохнем и узнаем за время отдыха кое-что новенькое.

Понятно, что для того, чтобы различные Web-обозреватели отображали одну и ту же Web-страницу одинаково, язык HTML должен быть стандартизирован. Его стандартизацией (а также множеством других стандартов Интернета) занимается особая организация, называемая World Wide Web Consortium или, сокращенно, WWWC или, что встречается чаще, W^3C. Это название дословно можно перевести как "Комитет Всемирной паутины".

W^3C издает весьма увесистые труды, описывающие различные версии стандарта HTML. Последняя версия этого языка — 4.01 — вышла в конце 90-х годов прошлого века. Все современные версии Web-обозревателей поддерживают именно эту версию HTML.

Примечание

Судя по всему, версия 4.01 HTML станет действительно последней. В дальнейшем язык HTML будет постепенно заменен своим потомком — языком XHTML (eXtensible HypetText Markup Language, расширяемый язык гипертекстовой разметки). Фактически XHTML — это более строгий, усовершенствованный и очищенный от устаревших тегов потомок HTML. По крайней мере, по набору тегов языки HTML и XHTML очень похожи. В настоящее время язык XHTML еще не очень распространен и, судя по всему, более-менее значительную популярность получит не скоро.

Графика на Web-страницах. Внедренные элементы

Продолжим наши эксперименты с языком HTML и Web-страницами. И на этот раз поговорим о том, как поместить на нашу страничку графическое изображение.

Но сначала давайте введем еще один термин, который поможет нам в дальнейшей работе. Пусть любой абзац текста, любой его фрагмент, являющийся содержимым парного тега, а также все, что помещается на Web-страницу одинарным тегом, называется *элементом страницы*.

Ранее мы имели дело только с *текстовыми элементами* Web-страниц. Это пресловутые абзацы текста, заголовки, фрагменты курсива и полужирного текста. Они создаются с помощью уже знакомого нам кода HTML, который пишется в текстовом редакторе и сохраняется в текстовых файлах с расширением htm[l]. А теперь разговор пойдет о тех элементах страниц, которые нельзя закодировать с помощью HTML, — о *внедренных элементах*. Все внедренные элементы хранятся в отдельных файлах, а в самом коде HTML проставляется ссылка на нужный файл.

Внедренные элементы — это, в частности, графические изображения. В самом деле, графическую информацию невозможно сохранить в текстовом виде; для хранения рисунков, фотографий, схем придуманы свои собственные форматы — GIF, JPEG, BMP, TIFF и др. Для работы с ними текстовые редакторы не подходят — нужны специальные программы графических редакторов.

Предположим, у нас имеется некая картинка, которую мы хотим поместить на нашу Web-страницу. Что для этого нужно сделать?

Прежде всего, нам нужно определить, в какое место Web-страницы будет помещен рисунок, и найти соответствующий участок HTML-кода. После этого остается вставить туда особый одинарный тег `<IMG>`:

```
<H1>Пример Web-страницы</H1>
<P>Это простейшая Web-страничка, созданная в стандартном
<EM>Блокноте</EM> и отображенная в <EM><STRONG>Microsoft</STRONG>
Internet Explorer</EM>.</P>
<IMG SRC="exclam.gif">
```

Встретив этот тег, Web-обозреватель пошлет Web-серверу еще один запрос — на файл exclam.gif. Получив этот файл, он выведет его содержимое на Web-странице в том месте, где стоит тег `<IMG>` (см. рис. 2.5).

Здесь мы столкнулись с так называемыми *атрибутами тега* — особыми дополнительными параметрами тегов, задающими некоторые дополнительные условия или значения. В нашем случае тег `<IMG>` содержит атрибут `SRC`, которому присвоено *значение* `exclam.gif` — имя нужного нам файла. (Все значения пишутся в двойных кавычках "...".)

Атрибуты бывают обязательными и необязательными. *Обязательный* атрибут в любом случае должен присутствовать в теге. *Необязательный* же атрибут присутствовать не обязан; если же он не указан, то Web-обозреватель считает, что ему присвоено некое значение по умолчанию. Атрибут `SRC` тега `<IMG>` — обязательный, ведь чтобы поместить на Web-страницу изображение, нужно указать Web-обозревателю, откуда его брать. Типичный пример необязательного атрибута — это `ALT` того же тега `<IMG>`, задающий текст, который будет выводиться на страницу, если файл с изображением почему-то не удастся загрузить (*текст-замена*):

```
<IMG SRC="exclam.gif" ALT="Здесь должен быть восклицательный знак">
```

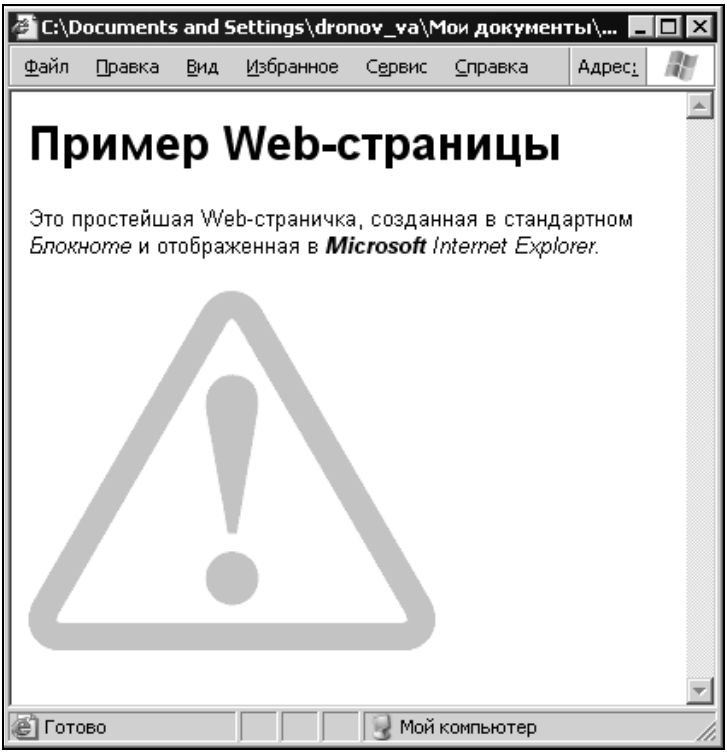


Рис. 2.5. Наша первая Web-страница с графическим изображением

На Web-страницах используют графические изображения, сохраненные в одном из следующих форматах: GIF, JPEG и PNG. Давайте кратко рассмотрим их.

Формат GIF (Graphic Interchange Format, формат обмена графикой) замечательно подходит для сохранения *штриховых* изображений, состоящих из набора линий разной длины, формы и цветов. Поэтому его используют для хранения элементов оформления страниц (линии, маркеры списков и т. п.), всяческих схем, чертежей, рисунков карандашом и т. п. Кроме того, формат GIF имеет одну очень интересную особенность — с его помощью можно сохранить в одном файле целый фильм (*анимированный GIF*), что незаменимо в рекламе.

Формат JPEG (Joint Pictures Encoding Group, группа кодирования неподвижных изображений), напротив, замечательно подходит для хранения *полутоновых* изображений, состоящих из множества точек, имеющих разный цвет. Поэтому картины и сканированные фотографии хранят только в этом формате.

Формат PNG (Portable Network Graphics, перемещаемая сетевая графика), как говорят его создатели, объединяет возможности GIF и JPEG, не "при-

хватывая" заодно с собой их недостатки. Надо сказать, что задумка получилась удачная: в PNG можно сохранять и штриховые, и полутоновые изображения без потери качества. Только вот что-то он пока никак не завоюет заслуженную популярность.

Примечание

В Интернете, кроме трех перечисленных ранее, применяются также и другие форматы графики. В частности, очень популярен формат Shockwave/Flash, разработанный фирмой Macromedia и позволяющий создавать, кроме статичных изображений, также фильмы и даже целые программы. Но, в отличие от GIF, JPEG и PNG, эти форматы не поддерживаются Web-обозревателями напрямую и требуют дополнительных программ, а для помещения их на Web-страницы используются другие теги.

Гиперссылки

Ранее говорилось, что Web-сайт — это набор Web-страниц, связанных друг с другом. Но вот как эти самые страницы связываются, сказано не было. Пора прояснить данный вопрос.

Для этого используются *гиперссылки* — особые связи, ведущие от одной Web-страницы к другой. Они имеют вид фрагментов текста, выделенных особым образом (как правило, цветом и подчеркиванием). Если по такой гиперссылке щелкнуть мышью, Web-обозреватель загрузит другую страницу, интернет-адрес которой указан в параметрах гиперссылки.

Гиперссылки создаются с помощью особого парного тега `<A>` и имеют следующий вид:

```
<A HREF="page125.html">Страница N125</A>
```

Видно, что интернет-адрес страницы, на которую должна вести гиперссылка, задается с помощью атрибута `HREF`. В данном случае гиперссылка ведет на страницу page125.html, находящуюся в той же папке, что и *текущая* (открытая в данный момент в Web-обозревателе).

Если нужно получить доступ к странице, находящейся в другой папке, код гиперссылки будет таким:

```
<A HREF="/folder1/page126.html">Страница N126</A>
```

Если же нужная страница находится вообще на другом сайте, то интернет-адрес должен включать его доменное имя (или IP-адрес):

```
<A HREF="http://www.othersite.ru/folder1/page3.html">Страница N3</A>
```

А теперь давайте поэкспериментируем с гиперссылками на практике. Добавим в HTML-код страницы, сохраненной в файле 2.2.html, строку, выделенную полужирным шрифтом:

```
<H1>Пример Web-страницы</H1>
```

```
<P>Это простейшая Web-страничка, созданная в стандартном
<EM>Блокноте</EM> и отображенная в <EM><STRONG>Microsoft</STRONG>
Internet Explorer</EM>.</P>
<P><A HREF="2.4.html">Сведения об авторе</A></P>
```

Сохраним новую страницу в другом файле с именем 2.3.html.

Теперь создадим страницу 2.4.html. Ее HTML-код очень прост:

```
<H1>Сведения об авторе</H1>
<P>Это моя проба силы!</P>
```

Теперь откроем в Web-обозревателе файл 2.3.html и щелкнем по гиперссылке **Сведения об авторе**. В окне Web-обозревателя появится страница со сведениями об авторе, сохраненная в файле 2.4.html.

А теперь сделаем небольшой фокус. Изменим HTML-код гиперссылки в коде страницы 2.3.html таким образом (изменения выделены полужирным шрифтом):

```
<P><A HREF="2.4.html" TARGET="_blank">Сведения об авторе</A></P>
```

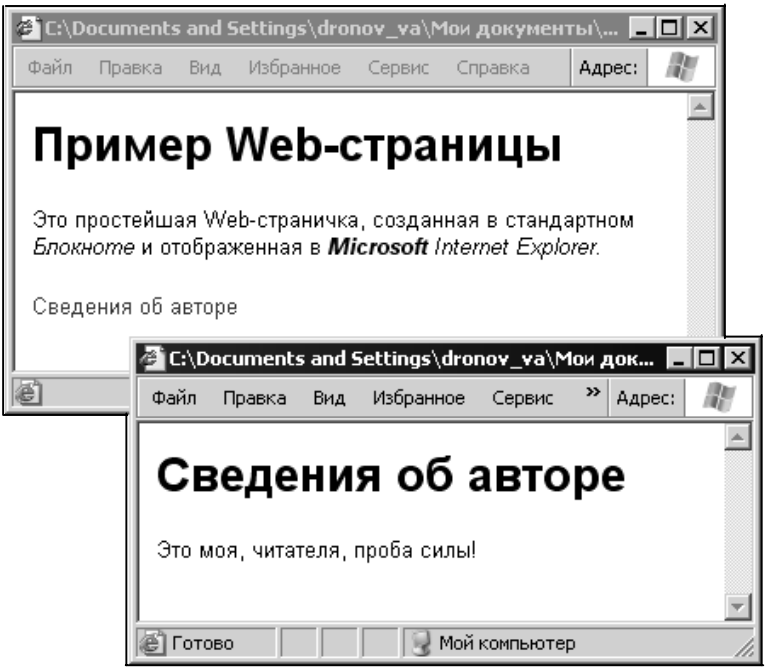


Рис. 2.6. Два окна Web-обозревателя, в которые загружены разные Web-страницы

Мы поместили в тег `<A>` необязательный атрибут `TARGET`, задающий *цель* гиперссылки. Цель задает, куда будет выведена Web-страница, на которую

она указывает. Если этому атрибуту присвоено значение `_blank`, страница будет выведена в отдельное окно Web-обозревателя. Чтобы задать обычное поведение гиперссылки (новая страница выводится в то же окно), достаточно присвоить атрибуту `TARGET` значение по умолчанию `_self` или вообще убрать этот атрибут из кода гиперссылки.

Если теперь сохранить исправленную Web-страницу 2.3.html, повторно открыть ее в Web-обозревателе и щелкнуть по гиперссылке **Сведения об авторе**, Web-обозреватель откроет новое окно и загрузит в него страницу 2.4.html. Оба этих окна показаны на рис. 2.6.

Мы еще вернемся к гиперссылкам в дальнейшем, когда приступим к созданию собственного сайта. А пока давайте выясним кое-что о правилах, согласно которым должны оформляться Web-страницы.

Правильно оформленные Web-страницы

Давайте еще раз посмотрим на HTML-код нашей первой Web-странички.

```
<H1>Пример Web-страницы</H1>
<P>Это простейшая Web-страничка, созданная в стандартном
<EM>Блокноте</EM> и отображенная в <EM><STRONG>Microsoft</STRONG>
Internet Explorer</EM>.</P>
```

Он очень компактен, корректно отображается в Internet Explorer, да и в других Web-обозревателях (автор проверял ее в Firefox — все нормально). Но, несмотря на это, наше первое Web-творение можно назвать разве что только *корректной страницей*. Корректные Web-страницы правильно отображаются в Web-обозревателях, но не удовлетворяют всем требованиям, которые предъявляются к Web-страницам стандартами W^3C.

А что необходимо сделать для того, чтобы наша страница удовлетворяла всем стандартам, иначе говоря, стала *правильно оформленной*? Для этого нужно добавить в ее код несколько новых тегов, называемых невидимыми. Содержимое *невидимых тегов* никак не отображается в окне Web-обозревателя, в отличие от обычных, *видимых*, тегов, но влияет на отображение всей страницы.

Далее приведен HTML-код правильно оформленной Web-страницы (добавленные фрагменты традиционно выделены полужирным шрифтом):

```
<HTML>
  <HEAD>
    <TITLE>Web-страница</TITLE>
  </HEAD>
  <BODY>
```

```
    <H1>Пример Web-страницы</H1>
    <P>Это простейшая Web-страничка, созданная в стандартном
    <EM>Блокноте</EM> и отображенная в <EM><STRONG>Microsoft</STRONG>
    Internet Explorer</EM>.</P>
  </BODY>
</HTML>
```

Давайте сохраним эту страничку в файле под именем 2.5.html для дальнейшего использования. И посмотрим внимательно на ее код HTML.

Видно, что наш изначальный код помещен в какой-то странный парный тег <BODY>. Этот тег задает так называемое *секцию тела* Web-страницы — собственно ее содержимого, выводимого на экран. Все, что мы хотим видеть в окне Web-обозревателя, должно находиться в этой секции, то есть в теге <BODY>.

Кроме секции тела, любая уважающая себя Web-страница также должна содержать задаваемую парным тегом <HEAD> *секцию заголовка*, где помещается некоторая служебная информация. (Не путать с обычным текстовым заголовком, задаваемым одним из тегов <H1>…<H6>!) Эта информация не выводится Web-обозревателем на экран, а используется для его внутренних нужд, в частности, задает некоторые параметры Web-страницы.

В нашем случае из этой служебной информации имеется только *название* Web-страницы, задаваемое парным тегом <TITLE>. Это название Web-обозреватель выводит в заголовке своего окна; кроме того, оно заносится в *историю* Web-обозревателя — список посещенных им ранее Web-страниц.

Кроме названия страницы, секция заголовка в подавляющем большинстве случаев содержит множество других данных, необходимых Web-обозревателю. Мы рассмотрим их позже.

Осталось сказать, что весь HTML-код — и секция заголовка, и секция тела — Web-страницы должен быть вложен внутрь парного тега <HTML>.

Иерархия тегов HTML

О языке HTML осталось рассказать совсем немного. Заодно введем еще несколько терминов, которые помогут нам в дальнейшем.

В начале этой главы мы говорили о вложенности тегов друг в друга. Так вот, приведенный ранее HTML-код — хороший пример такой вложенности. Вложенные друг в друга теги образуют весьма сложную структуру или, как говорят профессиональные программисты, *иерархию*. Так, тег <STRONG> вложен в тег <EM>, тег <EM> — в тег <P>, тег <P> — в тег <BODY>, а тег <BODY> — в тег <HTML>. И это только простейшая Web-страничка — в более сложных страницах иерархия бывает на порядки более замысловатая.

Поэтому, чтобы разобраться в ней, вложенные теги в коде HTML часто пишутся с отступами, как и показано ранее. А иногда даже пишут нечто похожее на это:

```
<HTML>
  <HEAD>
    <TITLE>
  <BODY>
    <H1>
    <P>
      <EM>
        <STRONG>
```

То есть убирают мешающие закрывающие теги и их содержимое.

А теперь — обещанные термины. Величина отступа на показанной ранее схеме обозначает *уровень вложенности* того или иного тега. Так, тег `<HTML>` имеет нулевой уровень вложенности, поскольку вообще никуда не вложен, тег `<BODY>` — первый уровень вложенности, а тег `<H1>` — второй и т. д.

Теги предыдущих уровней вложенности называются *родительскими тегами* или *родителями*, а теги последующих уровней — *дочерними тегами* или *потомками*. Например, для тега `<HEAD>` родительским тегом будет `<HTML>` и дочерним — `<TITLE>`. А для тега `<BODY>` родителем будет тег `<HTML>`, потомками — теги `<H1>`, `<P>`, `<EM>` и `<STRONG>`.

Вот и все. С началами языка HTML мы ознакомились. Давайте перейдем к более сложным вопросам, а именно к средствам оформления Web-страниц. Для этого мы используем другой язык и другую технологию, называемую CSS.

Средства оформления Web-страниц

Хорошо! Мы сделали нашу первую Web-страничку и вдоволь на нее налюбовались. Теперь нам хочется немного разукрасить ее, добавить цветов, изменить шрифты. Какие средства мы можем использовать для этого?

Каскадные таблицы стилей (CSS)

Самый современный на данный момент способ оформить Web-страницу по своему желанию — это использовать так называемые *каскадные таблицы стилей* (или просто *таблицы стилей*). По-английски они называются CSS (Cascading Style Sheets).

Для создания каскадных таблиц стилей используется не HTML, а другой язык, который так и называется — CSS. Сейчас мы с ним познакомимся.

Создание стилей

Предположим, что нам нужно выделить зеленым цветом весь текст, набранный полужирным шрифтом, то есть содержимое всех тегов `<STRONG>`. Для этого нам будет достаточно записать в секции заголовка нашей Web-страницы 2.5.html вот такой фрагмент кода (выделен полужирным шрифтом):

```
<HEAD>
  <STYLE>
    STRONG { color: green; }
  </STYLE>
</HEAD>
```

Если теперь мы откроем эту страницу в Web-обозревателе, то увидим, что слово "Microsoft" позеленело. Так что же мы сделали? И что это за странный текст заключен в парном теге `<STYLE>`?

А это и есть таблица стилей. В данном случае она находится в том же файле, что и сама Web-страница, и поэтому называется *внутренней*.

Наша первая таблица стилей содержит только один *стиль* — описание формата какого-либо элемента страницы. Наш стиль переопределяет внешний вид содержимого тега `<STRONG>` и так и называется — *стиль переопределения тега*.

Каждый стиль состоит из имени и описания. *Имя* однозначно идентифицирует стиль и в нашем случае совпадает с названием тега. *Описание* стиля записывается после имени через пробел в фигурных скобках и содержит набор *атрибутов стиля* и их значений. Между атрибутом и его значением должен стоять знак двоеточия, а пары "атрибут-значение" отделяются друг от друга знаком точки с запятой.

Описание нашего единственного стиля содержит единственный атрибут `color` (цвет текста), которому присвоено значение `green` (зеленый). Мы можем добавить в описание стиля еще один атрибут, скажем, `text-decoration` (дополнительное "украшение" текста) со значением `underline` (подчеркнутый). Вот так:

```
<HEAD>
  <STYLE>
    STRONG { color: green;
      text-decoration: underline; }
  </STYLE>
</HEAD>
```

Посмотрим, что покажет нам Web-обозреватель... Действительно работает!

Внимание

При оформлении Web-страниц нужно избегать подчеркнутого текста. Практически все Web-обозреватели показывают подчеркнутыми гиперссылки, и если текст Web-страницы содержит и другие подчеркнутые фрагменты, гиперссылками не являющиеся, посетитель сайта будет обескуражен.

Да, но что, если нам нужно изменить оформление фрагмента текста, являющегося содержимым какого-либо произвольного тега? Для этого нам нужно создать стиль другого типа, называемый *стилевым классом*. Стилевой класс может быть применен к любому тегу HTML.

Предположим, что нам нужно увеличить размер шрифта, которым набран текст "Microsoft Internet Explorer" (содержимое тега `<EM>`). Для этого мы создадим стилевой класс `bigfont` в таблице стилей нашей страницы 2.5.html:

```
.bigfont { font-size: 14pt; }
```

Имя стилевого класса всегда должно начинаться с точки — это важно. Что касается атрибута `font-size`, то он задает размер шрифта. А значение `14pt` обозначает размер шрифта, равный 14 пунктам.

Теперь нам нужно как-то привязать только что созданный стилевой класс к тегу `<EM>`, содержащему текст "Microsoft Internet Explorer". Для этого используем поддерживаемый всеми тегами атрибут `CLASS`, в качестве параметра которого указывается имя нужного стилевого класса без точки. Добавим его в тег `<EM>`, как показано далее:

```
<EM CLASS="bigfont"><STRONG>Microsoft</STRONG> Internet Explorer</EM>
```

Сохраним исправленную Web-страницу 2.5.html, откроем ее в Web-обозревателе. И увидим то, что показано на рис. 2.7.

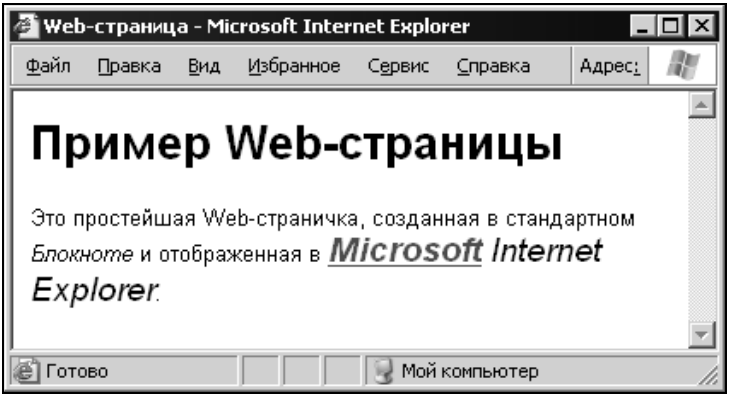


Рис. 2.7. Наша Web-страница после применения стилей

Видно, что текст "Microsoft Internet Explorer" увеличился в размерах, то есть стилевой класс `bigfont` действует. При этом также продолжает действовать стиль переопределения тега `<STRONG>`, созданный нами ранее — слово "Microsoft", помимо того, что "выросло" в размерах, осталось зеленым и подчеркнутым. Выходит, стили как бы наложились друг на друга. (Забегая вперед, нужно заметить, что это действует правило каскадности, о котором мы расскажем позже.)

Продолжим свои эксперименты со стилями. Давайте немного изменим стиль переопределения тега `<STRONG>` таким вот образом:

```
STRONG { color: green;
  text-decoration: underline;
  font-size: 9pt; }
```

Здесь мы задали для полужирного текста размер шрифта, равный 9 пунктам. Теперь, если мы откроем нашу многострадальную страницу 2.5.html в Web-обозревателе, то увидим то, что показано на рис. 2.8.

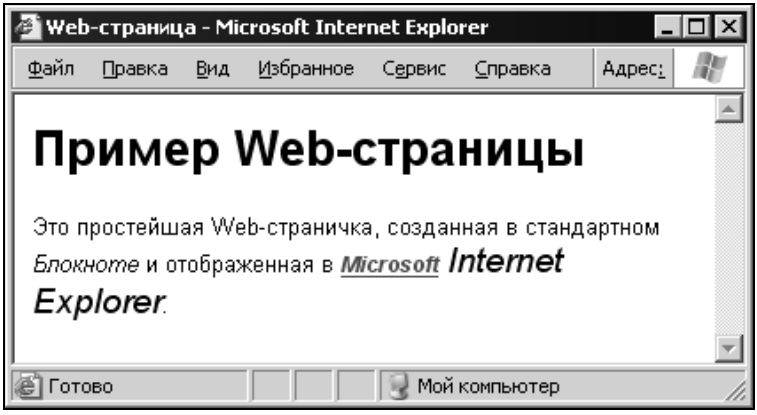


Рис. 2.8. Демонстрация "каскадности" стилей

Теперь слово "Microsoft", помимо зеленого цвета и подчеркивания, отличается еще и уменьшенным размером шрифта. Выходит, атрибут `font-size`, примененный нами в стиле переопределения тега `<STRONG>`, отменил действие того атрибута в стилевом классе `bigfont`, привязанному к родительскому тегу `<EM>`. Но при этом остальные установки стилевого класса `bigfont` перешли в стиль переопределения тега `<STRONG>`. (Это и есть каскадность таблиц стилей, о которой мы поговорим чуть позже.)

Стандарт CSS допускает создание и таких стилей переопределения тега:

```
EM STRONG { color: green; }
```

Подобный стиль будет применен к тегу <STRONG>, вложенному в тег <EM>. К другим тегам он применен не будет. Так, в приведенном далее примере слово "зеленый" станет зеленым, а остальной текст останется черным.

```
<EM>Курсив <STRONG>зеленый</STRONG></EM>
```

Можно скомбинировать стиль переопределения тега и стилевой класс, создав *гибридный стиль*. Вот так:

```
EM.bigfont { font-size: 14pt; }
```

Подобный стиль может быть применен только к тегу <EM>, имеющему атрибут CLASS, значение которого равно bigfont. К другим тегам этот стиль применен не будет. Так, в приведенном далее примере слова "Большой текст" будут набраны увеличенным шрифтом (14 пунктов), а слова "Обычный текст" — обычным шрифтом, заданным по умолчанию.

```
<EM CLASS="bigfont">Большой текст</EM>
<STRONG CLASS="bigfont">Обычный текст</STRONG>
```

Три способа задания стилей

Стандарт CSS допускает задавать стили тремя различными способами. Сейчас мы их рассмотрим.

Первый способ — поместить стили во внутреннюю таблицу стилей. Мы уже знаем, что внутренняя таблица стилей находится в секции заголовка Web-страницы внутри парного тега <STYLE>. Внутренние таблицы стилей используются, если нужно применить какой-то набор стилей только внутри этой страницы.

Второй способ — поместить нужные стили во внешнюю таблицу стилей. *Внешняя таблица стилей* хранится в отдельном файле с расширением css и подключается к Web-странице с помощью особого одинарного тега <LINK>. Вот так:

```
<LINK REL="stylesheet" HREF="styles.css" TYPE="text/css">
```

Из трех атрибутов этого тега нас интересует только атрибут HREF. Он задает файл, в котором хранится внешняя таблица стилей. Остальные атрибуты несут служебную информацию и предусмотрены для особых случаев.

Содержимое файла таблицы стилей — то же самое, что и тега <STYLE>. Единственное: сам тег при этом указывать в файле не нужно.

Внешние таблицы стилей следует использовать, если нужно применить один и тот же набор стилей сразу к нескольким Web-страницам.

И последний способ — самый интересный. Это *встроенные стили*, описания которых помещаются прямо в нужный тег, точнее, в особый атрибут STYLE, поддерживаемый практически всеми видимыми тегами HTML.

```
<P STYLE="font-size: 9pt; color: green;">Маленький и зеленый.</P>
```

Этот пример показывает, как задается встроенный стиль. Фактически он содержит только описание стиля. Имя здесь не указывается, так как этот стиль не нужно однозначно идентифицировать — и так ясно, к какому тегу он должен быть применен.

Если же нужно применить какой-либо стиль к произвольному фрагменту текста, не являющемуся содержимым никакого тега, то нужно использовать особый парный тег `<SPAN>`. Этот тег занимается только тем, что помечает фрагмент текста для привязки к нему стиля.

```
<P>Это <SPAN STYLE="text-decoration: underline;">подчеркнутый</SPAN>
текст.</P>
```

Оформляемый фрагмент текста просто помещается внутрь тега `<SPAN>`, после чего мы можем делать с ним все что угодно.

В одной и той же Web-странице могут быть использованы все три способа задания стилей одновременно. То есть к странице может быть подключена внешняя таблица стилей, в секции заголовка — содержаться внутренняя, а к некоторым тегам — применены встроенные стили. Это позволяет очень гибко управлять оформлением Web-страниц.

Почему "каскадные"?

Но как Web-обозреватель разбирается со всем этим "зоопарком" стилей? Давайте выясним это. И заодно узнаем, почему таблицы стилей называются еще и каскадными.

Предположим, что у нас имеется вот такая внешняя таблица стилей:

```
P          { font-size: 10pt; }
H1         { font-size: 20pt;
             text-align: center; }
.copyright { font-style: italic; }
```

Атрибут `font-style` задает начертание шрифта, а его значение `italic` делает шрифт курсивным без использования тега `<EM>`. А атрибут `text-align` задает выравнивание текстового абзаца, в нашем случае — по центру (значение `center`).

Сохраним созданную таблицу стилей в файле под именем 2.1.css. И создадим небольшую Web-страницу с таким HTML-кодом:

```
<HTML>
  <HEAD>
    <TITLE>Эксперименты со стилями</TITLE>
    <LINK REL="stylesheet" HREF="2.1.css" TYPE="text/css">
    <STYLE>
      H1 { font-size: 14pt;
```

```
            color: red; }
      </STYLE>
    </HEAD>
    <BODY>
      <H1>Стили</H1>
      <P>Давайте немного поэкспериментируем со стилями.</P>
      <P CLASS="copyright">
        Авторские <SPAN STYLE="font-style: normal">права</SPAN>
          принадлежат нам.
      </P>
    </BODY>
</HTML>
```

Все это нам уже знакомо. Единственное — значение `normal` атрибута стиля `font-style` убирает у шрифта курсивное начертание.

Итак, самым первым делом Web-обозреватель загружает внешнюю таблицу стилей 2.1.css, подключенную к Web-странице с помощью тега `<LINK>`. Таким образом, стили переопределения тегов `<P>` и `<H1>` и стилевой класс `copyright` будут применены к Web-странице сразу же.

Далее Web-обозреватель считывает внутреннюю таблицу стилей и выясняет, что в ней также содержится стиль переопределения тега `<H1>`. То есть возникает *конфликт стилей* и вступает в силу правило каскадности, разрешающее его. И разрешающее столь изящно, что нельзя не восторгаться. (Вот если бы так решались все конфликты на свете!..)

К определению стиля, сделанному во внешней таблице, Web-обозреватель добавляет определение, сделанное во внутренней таблице. А если определение затрагивает один и тот же атрибут (в нашем случае — `font-size`, задающий размер шрифта), берется определение, сделанное во внутренней таблице. Правило "своя рубашка ближе к телу" работает и в случае таблиц стилей.

Результирующий стиль тега `<H1>`, сформированный Web-обозревателем для "внутреннего потребления", будет иметь такой вид:

```
H1 { font-size: 14pt;
     text-align: center;
     color: red }
```

Если мы присмотримся к коду Web-страницы и таблицы стилей, то заметим еще один конфликт стилей. Разрешается он таким же образом, как и предыдущий.

Сначала Web-обозреватель берет определение стиля `P` (переопределение одноименного тега) и добавляет к нему определение стилевого класса `copyright`.

При этом атрибуты, определенные в стилевом классе, имеют приоритет перед определенными в стиле переопределения тега. Здесь действует еще одно правило, которое стоит запомнить: "более частные определения имеют приоритет перед более общими". А стилевые классы — как раз более частные, так как могут быть привязаны к любому тегу.

Далее полученный стиль привязывается к тегу `<P>` с атрибутом `CLASS`, равным `copyright`, и содержимое этого тега будет набрано курсивным шрифтом размером 10 пунктов. Но в содержимом этого тега мы видим вложенный тег `<SPAN>`, для которого был задан встроенный стиль, убирающий курсивное начертание. Поскольку встроенный стиль — еще более частный, чем даже стилевой класс, то его определения имеют наивысший приоритет. И слово "права" (содержимое тега `<SPAN>`) будет уже набрано обычным шрифтом — не курсивом.

Все это может показаться сложным. И действительно, для начинающего (да и зачастую для опытного) Web-дизайнера каскадность — бич божий. Но, разобравшись в правилах разрешения конфликтов стилей, можно сократить свои таблицы стилей до минимума. Так, чтобы изменить оформление какого-либо элемента одной из Web-страниц, достаточно будет переопределить только нужные атрибуты стиля во внутренней таблице или встроенном стиле. Разумеется, это намного проще и быстрее, чем писать отдельную таблицу стилей специально для этой страницы.

Мы еще вернемся к таблицам стилей в дальнейшем, когда начнем-таки создавать свой сайт. А сейчас давайте вернемся к уже знакомому HTML и совершим путешествие в историю этого языка.

Теги физического форматирования HTML

Таблицы стилей — это довольно позднее нововведение в Web-дизайне. Если язык HTML появился в 1989 году, то таблицы стилей — только в 1997. Но более-менее широкую популярность они получили только в последние три-четыре года. А что же было до этого?

А в "достилевую" эпоху для оформления текста на Web-страницах использовались так называемые теги физического форматирования HTML. С их помощью Web-дизайнер задавал шрифт и цвет текста, делал текст полужирным или курсивным. В принципе, эти теги до сих пор существуют — их никто не отменял, просто они объявлены не рекомендуемыми к использованию. Вместо них рекомендуется использовать таблицы стилей, более мощные и удобные в работе.

Да, но что такое теги физического форматирования? И как их использовать? (Мало ли что — вдруг мы будем править чужой код HTML!)

Вспомним два уже известных нам тега, позволяющих форматировать текст. Это тег `<STRONG>`, делающий шрифт текста полужирным, и тег `<EM>`, задающий

курсивное начертание шрифта. В принципе, с их помощью тоже можно форматировать текст, чем мы с успехом и занимались ранее.

Но все дело в том, что теги <STRONG> и <EM> не просто форматируют текст. Они еще его и выделяют особым образом: тег <STRONG> — сильнее, тег <EM> — слабее. Собственно, это не теги форматирования, а теги выделения текста; Web-обозреватель просто пытается показать эту "выделенность" своими средствами, изменяя шрифт. Разные программы Web-обозревателей могут, в принципе, показать эту "выделенность" по-разному: цветом, положением на Web-странице, интонацией голоса (уже сейчас есть технологии чтения текста с экрана) и др. Поэтому говорят, что <STRONG> и <EM> — это *теги логического форматирования* текста.

В противоположность им *теги физического форматирования* просто делают текст полужирным, курсивным или зеленым без всякой "задней мысли". Встретив такой тег, Web-обозреватель просто отформатирует текст, как предписывается таким тегом, но не будет выделять его другими способами. (Собственно, так же обрабатываются и таблицы стилей — Web-обозреватель просто форматирует элемент страницы, как предписывает стиль, и оставляет его в покое.)

Тегам логического форматирования <STRONG> и <EM> соответствуют парные теги физического форматирования <B> и <I>; первый делает шрифт текста полужирным, второй — курсивом. Также существует парный тег <U>, делающий текст подчеркнутым. А парный тег <FONT> позволяет задать имя и размер шрифта, которым набран текст, и цвет текста.

Да, но почему бы не использовать теги физического форматирования вместо таблиц стилей? В принципе, использовать можно, но не нужно. И вот почему.

- Теги физического форматирования объявлены комитетом W^3C не рекомендованными к использованию. Это значит, что может наступить такой момент, что новейший Web-обозреватель (или иная программа, обрабатывающая код HTML) просто не будет их поддерживать. Конечно, такое если и наступит, то очень нескоро, но ведь правду говорят: "готовь сани летом".

Примечание

Стандарт языка XHTML уже не включает теги физического форматирования как устаревшие.

- С таблицами стилей работать значительно удобнее. Сложный HTML-код читать очень тяжело, а если он еще и под завязку набит тегами физического форматирования, — так и вообще почти невозможно. А если нам вдруг понадобится весь текст, набранный полужирным шрифтом, еще и раскрасить в темно-синий цвет? Ведь намного удобнее написать одну строчку кода CSS — стиль переопределения тега <STRONG>, чем выиски-

вать по всему коду теги <B> и помещать их содержимое еще и в теги <FONT>.

- Таблицы стилей предлагают больше возможностей. Например, написав пару строчек CSS-кода, мы можем заключить текстовый абзац в симпатичную рамочку. А средствами HTML напрямую это сделать невозможно — только довольно корявыми ухищрениями и уймой дополнительного HTML-кода.
- Таблицы стилей развиваются, а теги физического форматирования — нет. Сейчас уже принят стандарт на вторую версию каскадных таблиц стилей — CSS2. Новых возможностей в ней столько, что пока ни один Web-обозреватель не поддерживает CSS2 полностью.

Что ж, на этом рассказ об оформлении Web-страниц можно считать законченным. Поговорим еще об одной важной вещи — о кодировках русского текста.

Кодирование текста. Проблема русских кодировок

Сначала выясним, как текст представляется в памяти компьютера. Дело в том, что каждому символу, вводимому с клавиатуры, выводимому на экран и хранящемуся в файле, сопоставляется уникальный номер, называемый *кодом символа*. Так что в памяти компьютера текст фактически представляет собой набор кодов. *Служебные символы*, не видимые на экране, например, возврат каретки и перевод строки, обозначающие конец каждой строки текста, также имеют свои коды.

Совокупность кодов символов вместе с описанием, какой код какому символу соответствует, образует *кодировку* или *кодовую таблицу*. Каждая кодировка имеет свое имя, например, 1251 или КОИ-8.

Поскольку любой язык использует свой набор символов, для каждого языка кодировки, как правило, различны. (Исключение — некоторые западноевропейские языки, использующие одну кодировку для всех.) Русский язык использует сразу несколько различных кодировок. Вот с этими кодировками и связана одна из главнейших проблем *Рунета* — русской части Интернета.

А все потому, что русские версии разных операционных систем используют разные кодировки. Так, русская версия Windows использует кодировку 1251, а русская версия MS-DOS — 866 (она же ISO-8859-5). А если добавить сюда еще кодировку, используемую русской версией операционной системы UNIX, — КОИ-8 и русской версией компьютеров Macintosh — MacCyrillic, кодировок станет уже четыре. И это только главные — на памяти автора этой книги существовало еще несколько менее распространенных кириллических кодировок ("основная" кодировка ГОСТ, "болгарская", "американская", "югославская" и т. п.).

Примечание

Лет десять тому назад была разработана универсальная кодировка Unicode, поддерживающая ВСЕ имеющиеся на Земле языки. Но, хотя в настоящее время эта кодировка стала весьма популярной, проблема разных кодировок остается.

Чем все это грозит? Дело в том, что в разных кодировках одни и те же символы (это относится только к символам русского языка — с латинскими все в порядке) имеют разные коды. В результате текст, набранный в одной кодировке, при просмотре в другой становится абсолютно нечитаемым.

Все мы пытались открыть текстовый документ, созданный в Блокноте, в Norton Commander и видели, что при этом получается — текст превращается в набор непонятных закорючек. А все потому, что русские кодировки 866 (MS-DOS), используемая Norton Commander, и 1251 (Windows), используемая Блокнотом, не совпадают! В них один код соответствует разным символам.

Каков же выход?

Выхода нет. Можно надеяться только на то, что какая-то из кодировок станет стандартом и постепенно вытеснит конкурентов. Пока что на роль такого (негласного) стандарта претендует 1251, хотя интернетчики старого поколения, пользующиеся UNIX-совместимыми системами, продвигают на эту роль КОИ-8. Во всяком случае, сейчас большинство Web-страниц, имеющихся в русском сегменте Сети, написано в кодировке 1251. Хотя сейчас практически все Web-обозреватели поддерживают все имеющиеся в наличии кодировки.

Специально для указания кодировки, в которой была набрана Web-страница, комитет W^{3}C предусмотрел в языке HTML особый тег `<META>`. Так, чтобы дать Web-обозревателю понять, что страница набрана в кодировке 1251, нужно вставить в секцию заголовка такой код:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
↳charset=windows-1251">
```

Если же мы создадим Web-страницу в кодировке КОИ-8 (правда, в Блокноте это сделать не получится — нужен текстовый редактор, который поддерживает эту кодировку), то должны будем использовать для указания кодировки такой код:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=koi8-r">
```

Если Web-страница набрана на одном из европейских языков, использующих кодировку 1250, тег `<META>` будет таким:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;
↳charset=windows-1250">
```

Для англоязычных Web-страниц кодировку указывать, в принципе, необязательно. Но если есть желание, то можно вставить в секцию заголовка страницы вот эту строку:

```
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=iso-8859-1">
```

Для русскоязычных же Web-страниц кодировку лучше указывать всегда. Это избавит от возможных проблем и нас — Web-дизайнеров — и посетителей наших будущих сайтов.

Начала сайтостроения

Как мы уже знаем, Web-сайт — это набор множества Web-страниц, объединенных общей тематикой и связанных друг с другом гиперссылками. То есть для создания настоящего сайта достаточно знаний о языках HTML и CSS. Ну и, конечно, нужно придумать тематику сайта и спланировать его структуру.

Планирование сайта

Этап *планирования* — первейший и важнейший в разработке любого сайта. Это справедливо и для простейшей домашней странички, и для гигантского сайта транснациональной суперкорпорации, такой как Intel или Coca-Cola. В самом деле, прежде чем что-то делать, необходимо твердо уяснить, что же мы хотим получить в результате. И уяснить это нужно в самом начале, перед тем как приниматься за дело. Ведь когда работа сделана, менять что-либо значительно труднее, чем сразу делать все как надо.

Поэтому, начиная планирование нашего будущего сайта, давайте лучше вообще выключим компьютер, чтобы не возникло соблазна сразу же засесть за "ваяние" Web-страниц. На этом этапе нам понадобятся только карандаш и бумага или, если мы все-таки предпочтем компьютер, программа текстового редактора или специальный пакет проектирования сайта, наподобие Microsoft Visio.

Итак, что же нужно решить для себя перед началом работы над сайтом?

1. Прежде всего, четко определить задачи сайта. Что он должен делать: рассказывать о ком-либо или о чем-либо, привлекать клиентов, помогать решать какие-то проблемы, просвещать или развлекать. В зависимости от задач структура сайта может сильно различаться.
2. Определить, какая конкретно информация должна присутствовать на сайте, а какая — не должна. Здесь главный принцип — ничего лишнего, только то, что действительно нужно потенциальным посетителям.
3. Собрать всю необходимую информацию. И сделать это прямо сейчас, чтобы не заниматься ее поисками во время работы над сайтом — у нас

тогда и без этого будет чем заняться. Все тексты, изображения, файлы, которые мы намерены выложить в Сеть, должны быть на нашем компьютере.

4. Решить, в каком ключе будет выполнен дизайн сайта. Будет ли он консервативным, строгим или затейливым. Соответственно, домашняя страничка должна отражать эстетические наклонности автора, развлекательный сайт лучше сделать повеселее, а новостной — поскромнее, чтобы пестрота дизайна не заслоняла главное — информацию. На этом этапе лучше всего будет набросать на бумаге, как должна выглядеть та или иная страница.
5. Продумать логическую структуру сайта и — желательно — нарисовать ее. Здесь лучше не изобретать самому велосипед, а посетить какой-нибудь уже существующий и популярный Web-сайт сходной тематики и посмотреть, как он организован. Например, для домашнего сайта идеальна такая структура: начальная ("домашняя") страница с краткими сведениями о хозяине, приглашением посетить другие страницы сайта и набором ссылок на них; а на других страницах размещаются информация об увлечениях, проектах, разработках (если это программист, музыкант, художник, то список ссылок на файлы программ, аудиоклипов или картин), фотогалерея и странички с набором "дружественных" ссылок и более подробными сведениями об авторе, с почтовым адресом и фотографией.
6. Проверить, ничего ли мы не забыли. Это последний этап планирования сайта, но не менее важный, чем остальные.

Теперь самое время поговорить о логической структуре сайта, то есть о том, из каких разделов он будет состоять и как эти разделы будут связаны друг с другом. Тема это непростая, и разговор будет долгим.

Логическая структура Web-сайта

Итак, *логическая структура сайта* описывает назначение и взаимосвязь различных Web-страниц сайта. Она определяется, прежде всего, тем, как организована на сайте информация. То есть для каждого разрабатываемого сайта нужно придумывать свою собственную структуру.

Конечно, существуют некие общие принципы структуризации сайта, которым нужно следовать всегда. Сейчас мы их и рассмотрим. Вот примерный план хорошо продуманного сайта:

```
Главная страница
  Новости сайта
    Архив новостей
  Раздел 1
    Страница 1
    Страница 2
```

```
  . . .
Раздел 2
  Страница 1
  Страница 2
  . . .
. . .
Сведения о разработчиках
Контактные данные
Карта сайта
```

Теперь поговорим о каждой странице сайта более подробно.

Главная или *начальная* страница, как правило, задается в качестве страницы по умолчанию, то есть посетитель после набора интернет-адреса сайта попадает именно на нее. Она содержит краткое название сайта, краткую вводную информацию о сайте, новости (необязательно) и набор гиперссылок, ведущих на другие страницы сайта (так называемую *полосу навигации*). Иногда на главной странице помещаются также сведения о разработчиках и их авторских правах, а также сведения о контакте с разработчиками и другими лицами и организациями, упомянутыми на сайте.

Можно сказать, что главная страница — это "лицо" всего сайта. Поэтому ее стремятся делать не слишком большой, чтобы посетитель не ушел с сайта, не дождавшись окончания ее загрузки. Но не следует впадать в другую крайность — делать главную страницу настолько "спартанской" по содержанию, что посетитель не сможет даже понять, куда он попал. Главная страница должна давать посетителю достаточно информации о сайте, но при этом не перегружать его излишними сведениями и не выводить из себя ожиданием окончания загрузки — это очень важно.

Новости сайта часто помещают на главной странице. Они представляют собой хронологический список всех дополнений и обновлений, сделанных на сайте. Как правило, выводятся только новости за некоторый период (месяц, квартал, год, в зависимости от того, насколько часто обновляется сайт). Для доступа к более старым новостям предусматривается страница так называемого *архива новостей*.

Полезное содержимое сайта — это та информация, ради которой он был создан. Структурируется она так же, как в книге: отдельные абзацы, посвященные какой-либо теме, объединяются в главы, а главы в свою очередь — в разделы. Таким образом, посетитель сайта сразу сможет найти нужную информацию, двигаясь от разделов к главам, а от глав — к абзацам, пока не найдет то, ради чего сюда пришел.

Сведения о разработчиках могут помещаться как на отдельной странице, так и на главной. Если разработчиков немного (или вообще один), более предпочтителен второй вариант. В таком случае сведения о них помещаются

в самом низу главной страницы, рядом со сведениями об авторских правах. Если же разработчиков много или сведения о них достаточно объемные, лучше поместить их на отдельную страницу. При этом обязательно указывается адрес электронной почты, по которому посетитель сайта сможет написать о проблемах, с которыми он столкнулся (незагружающиеся файлы, "пустые" изображения, ссылки, ведущие "в никуда", ошибки в тексте и т. п.).

Сведения о контакте с владельцем сайта нужны, если данный сайт преследует рекламные цели. Например, если это торговый сайт, необходимо указать контактные данные, иначе никто из потенциальных покупателей не сможет связаться с продавцом. В этом случае необходимы адреса как обычной ("бумажной"), так и электронной почты, а также телефон, факс и пейджер — в общем, все данные, по которым могли бы обратиться потенциальные клиенты.

Карта сайта — это страница, на которой находятся гиперссылки, ведущие на все страницы сайта, организованные в виде его логической структуры. Карта сайта служит для того, чтобы посетитель, точно знающий, что ему нужно, мог сразу добраться до необходимой информации. Карта обычно имеется на всех достаточно больших и запутанных сайтах; на простых сайтах она будет лишней.

Проектируем наш первый Web-сайт

Теперь, пока карандаш и бумага или программа текстового редактора еще у нас под рукой, набросаем структуру нашего первого Web-сайта. К его созданию мы приступим только в *главе 3*, но спланировать его лучше заранее.

Пусть наш сайт будет представлять собой архив файлов и статей по компьютерной тематике. Сайт будет содержать Web-страницы с набором ссылок на файлы (программы, документы, архивы и др.) и статьи, но самих файлов и статей содержать не будет (так мы избегнем проблем с авторскими правами).

На главной странице, помимо полосы навигации и вводного текста, будут также приведены новости сайта и сведения о разработчиках. Карту сайта мы делать не будем — она будет излишней в нашем простеньком сайте.

Напишем на бумаге или в текстовом редакторе логическую структуру нашего сайта:

```
Главная страница (новости сайта, сведения о разработчиках)
  Файлы
    Интернет
    Офис
    Мультимедиа
    Система
    Программирование
```

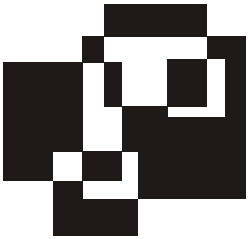
```
    Игры
    Украшения
    Прочее
  Статьи
    Интернет
    Офис
    Мультимедиа
    Система
    Программирование
    Развлечения
    Прочее
```

Каждая строка этого списка будет представлять собой отдельную Web-страницу. Страницы `Файлы` и `Статьи` будут содержать списки категорий (`Интернет`, `Офис` и т. д.); каждая позиция этого списка будет представлять собой гиперссылку на страницу, содержащую сами ссылки на собственно файлы и статьи, относящиеся к этой категории. Каждая статья пусть открывается в отдельном окне Web-обозревателя — так будет удобнее для пользователя.

Что дальше?

Ну вот, кажется, со структурой сайта все ясно. Можно приступать к работе...

Стоп! Мы что, собираемся создавать все Web-страницы нашего сайта вручную? Зачем? Ведь есть замечательные программы, которые помогут нам в этом деле. И называются они Web-редакторами. Один из Web-редакторов — Macromedia Dreamweaver MX 2004 — мы и используем в нашей работе. За дело!



Глава 3

Работа с Macromedia Dreamweaver MX 2004

Человечество постоянно стремится облегчить себе жизнь, придумывая все более и более изощренные инструменты для выполнения тяжелой, монотонной и неинтересной работы. И это понятно. Ведь высвободившееся время можно потратить с гораздо большей для себя пользой, скажем, заняться самообразованием, спортом, искусством или просто выспаться. А всей рутиной пусть занимаются машины.

Так и в Web-дизайне. Вместо того чтобы прилежно выписывать теги HTML, следить за вложенностью, исправлять ошибки, по нескольку раз в минуту проверять полученный результат в Web-обозревателе, человечество хочет просто написать нужный текст и получить готовую Web-страницу. Для этого программисты и пишут особые программы, предназначенные для создания Web-страниц, — так называемые *Web-редакторы*.

Одна из таких программ написана разработчиками из фирмы Macromedia и называется Macromedia Dreamweaver. Первая ее версия вышла еще в далеком 1998 году; в настоящее же время доступна версия MX 2004, шестая по счету. Dreamweaver — исключительно мощная и простая в использовании программа; с ней может работать даже начинающий пользователь, ничего не понимающий в HTML, но и профессионал останется ей доволен. Автор этой книги работает с Dreamweaver, начиная с самой первой его версии, и пока что не собирается менять его на что-либо другое.

Dreamweaver — типичнейший представитель *визуальных Web-редакторов*, работающих по принципу WYSIWYG (What You See Is What You Get, "что ты видишь, то ты и получишь"). При этом пользователь форматирует текст и в окне редактора сразу же видит результаты своих трудов. По такому же принципу работает известный текстовый редактор Microsoft Word.

Невизуальные Web-редакторы (они же — *HTML-редакторы*) основаны на другом принципе. Они работают непосредственно с самим HTML-кодом, предоставляя при этом пользователю различные дополнительные возможности: быстрая вставка тегов, удобное задание атрибутов и их значений, набор

предопределенных шаблонов для создания стандартных элементов Web-страницы, специальные редакторы для таблиц стилей, быстрый просмотр результатов и пр. В этом смысле они похожи на Блокнот, но значительно расширенный.

Напрашивается вывод, что визуальные Web-редакторы хорошо подходят для начинающих Web-дизайнеров, а невизуальными заинтересуются их более опытные коллеги. Но это не совсем верно. Профессионал, как правило, пользуется и теми, и другими; он может создать заготовку Web-страницы в визуальном Web-редакторе, а доделать — в невизуальном. Так намного проще, особенно если Web-страница достаточно сложна.

Необходимо, правда, сказать, что практически все серьезные Web-редакторы имеют режим правки непосредственно самого кода HTML (то есть фактически являются *гибридными Web-редакторами*). Поэтому сейчас практически всегда, когда говорят "визуальный Web-редактор", подразумевают как раз гибридные программы. Разумеется, к их числу относится и Dreamweaver, с которым нам пора познакомиться поближе.

Предварительная настройка Dreamweaver

Dreamweaver — замечательная программа, понимающая пользователя буквально с полуслова. Но, несмотря на это, перед началом работы над Web-сайтом нам придется его настроить. К счастью, таких обязательных настроек не очень много, но выполнить их нужно, иначе мы можем получить не тот результат, который нам нужен.

Все настройки Dreamweaver выполняются в многофункциональном диалоговом окне **Preferences**, состоящем из множества вкладок с разными элементами управления. Чтобы вызвать его, выберем в меню **Edit** пункт **Preferences** или нажмем комбинацию клавиш <Ctrl>+<U>.

Первым же делом нам нужно задать кодировку для новых Web-страниц. (О кодировках см. *главу 2.*) Для этого выберем в списке **Category**, находящемся в левой части окна, пункт **New Document**. После этого в правой части окна появятся элементы управления, с помощью которых задаются параметры вновь создаваемых Web-страниц (рис. 3.1).

Кодировка выбирается в раскрывающемся списке **Default encoding**. Здесь правило очень простое: если нет специальных требований к размещению Web-сайта на Web-сервере, выбираем пункт **Кириллица (Windows)** этого списка, соответствующий кодировке 1251. Для англоязычных Web-страниц нужно выбрать пункт **Западноевропейская** (кодировка iso-8859-1, используемая для английского языка) или **Центральноевропейская** (кодировка 1250).

Иногда бывает так, что администратор Web-сервера требует, чтобы Web-страницы, публикуемые на этом сервере, были сохранены в кодировке

КОИ-8. В таком случае в списке **Default encoding** нужно выбрать пункт **Кириллица (KOI8-R)**, задающий русскую версию КОИ-8, или **Кириллица (KOI8-U)**, задающий ее украинскую версию.

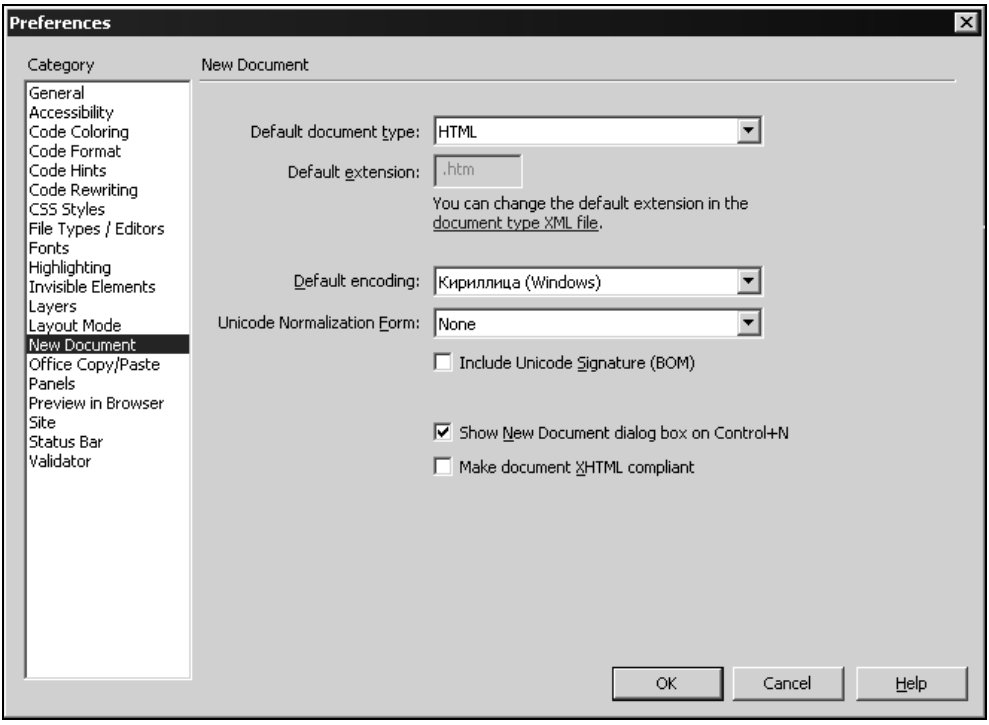


Рис. 3.1. Категория **New Document** диалогового окна **Preferences**

Примечание

Dreamweaver позволяет задать кодировку отдельно для каждой Web-страницы. Как это сделать, мы выясним позже.

Теперь нам нужно обязательно включить флажок **Use when opening existing files that don't specify an encoding**, если он не включен по умолчанию. После этого Dreamweaver будет автоматически применять выбранную в списке **Default encoding** кодировку ко всем открываемым Web-страницам, если в их HTML-коде нет тега `<META>`, задающего кодировку.

Задав кодировку, выберем в списке **Category** пункт **General**. Окно **Preferences** примет вид, показанный на рис. 3.2.

Флажок **Use and in place of <b> and <i>** включает или отключает использование вместо тегов физического форматирования текста

<B> и <I> тегов логического форматирования <STRONG> и <EM>. По умолчанию он включен; если же нет, то его нужно включить. Мы же хотим создавать Web-страницы в соответствии с самыми последними стандартами.

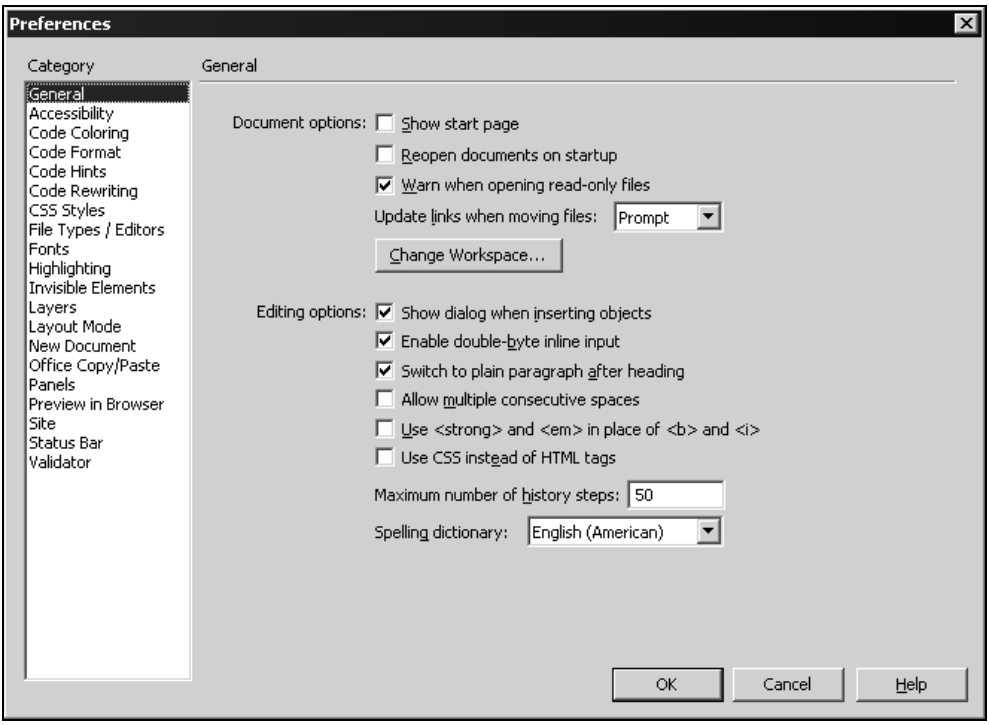


Рис. 3.2. Категория **General** диалогового окна **Preferences**

Флажок **Use CSS instead of HTML tags** включает или отключает использование для форматирования текста стилей CSS вместо тегов HTML. По умолчанию он также включен, и опять же, если это не так, его нужно включить.

После окончания настройки Dreamweaver нажмем кнопку **OK**, чтобы сохранить сделанные установки. Теперь можно приступать к работе.

Основы работы в Dreamweaver

Сейчас мы начнем работу над нашим первым Web-сайтом. Создадим для всех файлов, которые войдут в его состав (Web-страниц, таблиц стилей, графических изображений), отдельную папку и назовем ее, скажем, Site1. И начнем.

Как запустить программу в системе Windows, знает всякий. Нажмем хорошо знакомую нам кнопку **Start** (Пуск), выберем в меню пункт **Programs** (Про-

граммы), далее — пункт **Macromedia** и в появившемся подменю — пункт **Macromedia Dreamweaver MX 2004**. Через некоторое время (Dreamweaver запускается довольно долго, так что наберемся терпения) на экране появится его *главное окно*, в котором мы и будем работать.

Создание новой Web-страницы

Работа над новой Web-страницей начинается с ее создания. Чтобы создать новую "пустую" страницу, выберем пункт **New** в меню **File** или нажмем комбинацию клавиш <Ctrl>+<N>. На экране появится диалоговое окно **New Document**, показанное на рис. 3.3.

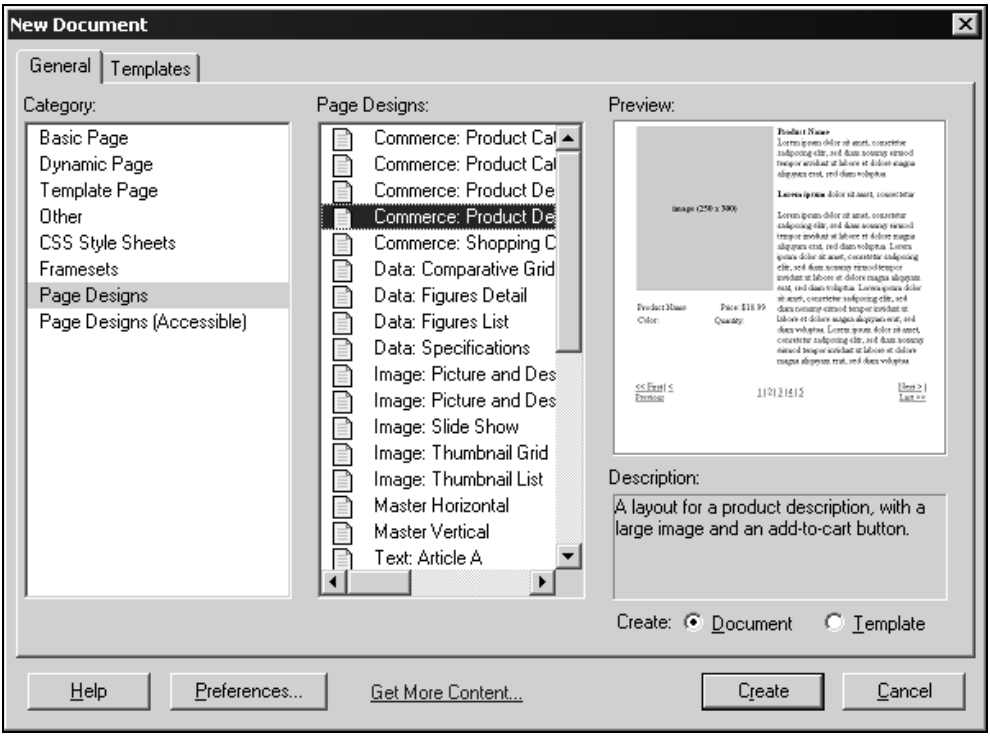


Рис. 3.3. Диалоговое окно **New Document**

Чтобы получить "чистый лист" для своего Web-творчества, выберем в списке **Category** пункт **Basic page**, в списке, расположенном правее, — пункт **HTML** и нажмем кнопку **Create**. На экране появится *окно документа*, в котором и будет открыта только что созданная Dreamweaver для нас Web-страница.

Здесь нужно немного рассказать об окнах документов. Dreamweaver позволяет открывать одновременно несколько Web-страниц; при этом каждая

страница открывается в своем отдельном окне. Это очень удобно при создании сайта: работая с одной Web-страницей, можно держать перед глазами другие.

Примечание

Dreamweaver предоставляет возможность создавать Web-страницы на основе *шаблонов* — своего рода заготовок, уже содержащих некоторые элементы. В списке **Category** окна **New Document** как раз выбирается категория шаблонов, а в списке, расположенном правее, — сам шаблон, относящийся к выбранной категории. В области предварительного просмотра при этом мы можем увидеть, что представляет собой выбранный шаблон.

Набор текста

Вот и наступила торжественная минута! Сейчас мы наберем наш первый текст в окне документа Dreamweaver (рис. 3.4) и создадим таким образом главную страницу нашего сайта.

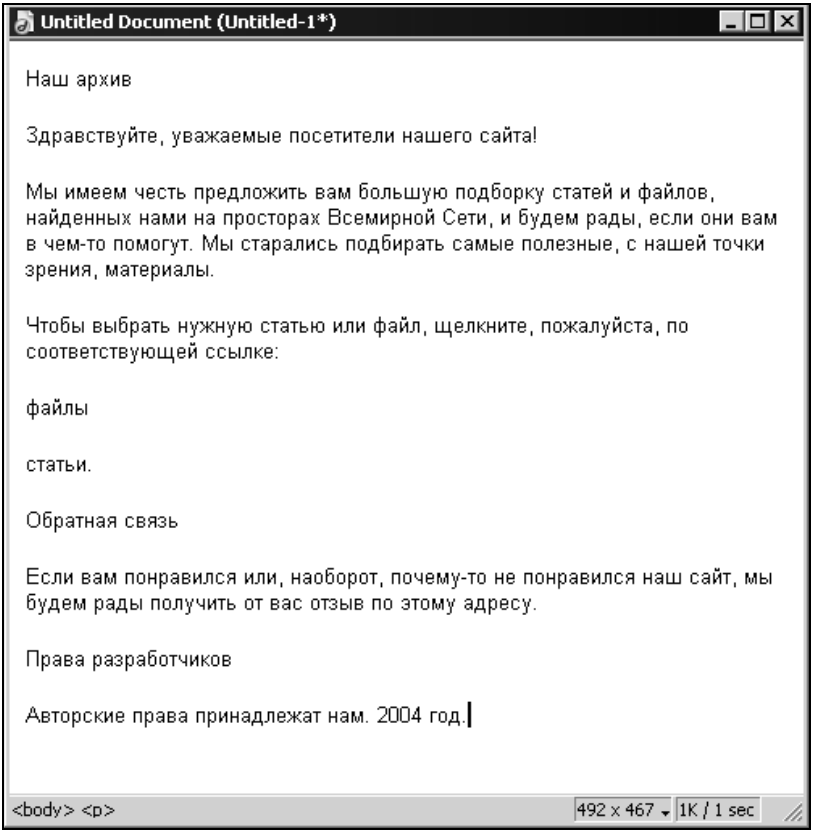


Рис. 3.4. Наш первый текст

Не будем сейчас заниматься его форматированием — пока просто наберем. Это выполняется точно так же, как и в любом текстовом редакторе — "стучим" по клавиатуре, смотрим на результат и пользуется клавишами-стрелками, <Backspace> и <Del> для правки. А в конце сохраним получившуюся Web-страницу.

Для сохранения Web-страницы нужно выбрать пункт **Save** в меню **File** или нажать комбинацию клавиш <Ctrl>+<S>. Если до этого мы не сохраняли нашу Web-страницу (а мы ее еще не сохраняли), на экране появится стандартное диалоговое окно сохранения файла Windows. Выберем в нем созданную ранее папку Site1 и подумаем, как бы назвать файл нашей первой Web-страницы.

Нам уже известно, что одна из Web-страниц на Web-сервере задается в качестве страницы по умолчанию. Также мы знаем, что такая страница обычно носит имя default или index (и расширение htm или html). Так давайте назовем ее default.htm. Введем это имя в поле ввода имени файла диалогового окна и нажмем кнопку сохранения.

Итак, мы создали нашу первую Web-страницу... Но мы же совсем забыли о названии! Том самом названии, задаваемом тегом `<TITLE>`, которое не отображается в окне Web-обозревателя, но выводится в его заголовке. Давайте зададим его прямо сейчас, чтобы потом не забыть. Тем более что это делается очень просто: поле ввода, где оно набирается, находится на небольшой серой панели, тянущейся вдоль верхнего края окна документа (рис. 3.5). (Эта панель еще называется *инструментарием документа*, так как в ней находятся некоторые элементы управления, позволяющие выполнять различные действия над Web-страницей.)

Рис. 3.5. Поле ввода названия Web-страницы в инструментарии документа

Введем в это поле текст `Наш архив` — название главной страницы нашего сайта. И сохраним страницу еще раз.

Все это время мы работали в режиме отображения Web-страницы Dreamweaver, то есть в режиме WYSIWYG. Ранее говорилось, что Dreamweaver позволяет пользователю редактировать также и HTML-код. Чтобы получить доступ к HTML-коду, нам будет нужно переключиться в режим отображения кода. Это выполняется с помощью особых кнопок, расположенных в левой части инструментария документа и показанных на рис. 3.6.

Рис. 3.6. Кнопки переключения режимов отображения Web-страницы

Давайте перечислим эти кнопки в порядке слева направо:

- **Code** — режим отображения HTML-кода;
- **Split** — режим отображения HTML-кода и Web-страницы. При этом окно документа будет разделено на две части; в верхней части будет показываться код HTML, в нижней — сама страница;
- **Design** — режим отображения Web-страницы.

На рис. 3.6 видно, что одна из этих кнопок, а именно **Design**, нажата, и одновременно включен режим отображения Web-страницы. Если теперь нажать кнопку **Code**, то включится режим отображения HTML-кода; одновременно кнопка **Code** станет нажатой, а кнопка **Design** "отожмется". Такие кнопки, объединенные в группы и работающие как переключатели, из которых может быть включен только один, называются *кнопками-переключателями*.

В режиме отображения HTML-кода хорошо видно, что Dreamweaver поместил каждый абзац введенного нами текста внутрь тега `<P>`. Также он автоматически сформировал секцию заголовка и поместил в нее название страницы и тег `<META>`, указывающий кодировку. Так что, не написав ни строчки HTML-кода, мы получили правильно оформленную Web-страницу.

А теперь давайте займемся ее оформлением.

Форматирование фрагментов текста

Итак, давайте выделим некоторые фрагменты набранного нами текста полужирным шрифтом и курсивом. И заодно изучим средства Dreamweaver по заданию параметров различных элементов страницы.

Начнем с того, что сделаем шрифт строки `Здравствуйте, уважаемые посетители нашего сайта!` полужирным. Для этого выделим ее так, как это делается в любом текстовом редакторе. И после этого посмотрим в самый низ главного окна программы. Там должно находиться небольшой окно, показанное на рис. 3.7.

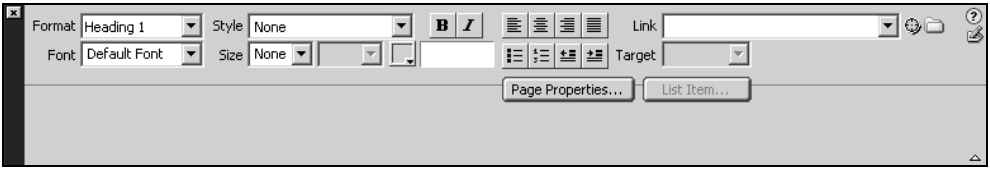


Рис. 3.7. Редактор свойств

Это окно (точнее, *панель* — небольшое окно Dreamweaver, содержащее только элементы управления и предназначенное для задания различных параметров) называется *редактором свойств*. С его помощью и выполняется работа над содержимым Web-страницы.

Внимание

Если в низу главного окна вместо редактора свойств видна только серая полоса с надписью **Properties**, нужно щелкнуть мышью точно по этой надписи — и редактор свойств появится на экране. Если же там видна только тонкая серая линия с плоской кнопкой, нужно щелкнуть по этой кнопке.

Содержимое редактора свойств зависит от того, что мы выделим в окне документа. Если, как в данном случае, мы выделим текст, то в нем появятся элементы управления для форматирования текста.

Чтобы сделать шрифт выделенного фрагмента текста полужирным, нужно нажать находящуюся в редакторе свойств кнопку **B**, после чего она останется нажатой. Если теперь нужно сделать полужирный шрифт, наоборот, обычным, достаточно "отжать" эту кнопку. Такие кнопки (как говорят специалисты, имеющие два состояния) называются *кнопками-выключателями*. Еще можно воспользоваться комбинацией клавиш <Ctrl>+<B> — она работает так же, как описанная ранее кнопка.

Если мы теперь переключимся в режим отображения HTML-кода, то увидим, что Dreamweaver поместил строку `Здравствуйте, уважаемые посетители нашего сайта!` внутрь тега `<STRONG>`. В принципе, мы этого от него и ждали.

Теперь давайте выделим целиком весь последний абзац, описывающий наши авторские права, и сделаем его шрифт курсивным. И заодно разучим весьма интересный способ выделить фрагмент текста, имеющийся только в Dreamweaver.

Давайте посмотрим на нижний край окна документа (не главного окна!). Там находится узкая серая полоса, содержащая различные сведения о Web-странице, — *строка статуса*. Мы можем встретить ее в окнах очень многих программ. И везде она ведет себя по-разному.

Так вот, в бóльшей части строки статуса окна документа Dreamweaver расположена *секция тегов* (рис. 3.8). Она содержит набор небольших кнопок, соответствующих тегам, в которые вложен фрагмент текста, где в данный момент находится текстовый курсор. Самая правая кнопка (**** — на рис. 3.8) соответствует тегу, содержимым которого является этот фрагмент, кнопка, расположенная левее (**<p>**), соответствует тегу-родителю и т. д. Самая левая кнопка — **<body>** — соответствует секции тела Web-страницы, то есть всему содержимому окна документа.

Рис. 3.8. Секция тегов, расположенная в строке статуса

Предположим, что нам нужно выделить текст, помеченный тегом `<STRONG>` (строку `Здравствуйте, уважаемые посетители нашего сайта!`, шрифт кото-

рой мы только что сделали полужирным). Для этого поместим текстовый курсор куда-либо на эту строку (после чего секция тегов примет вид, показанный на рис. 3.8) и нажмем кнопку **** этой секции, после чего вся строка будет выделена. (Хотя с тем же успехом мы могли бы щелкнуть кнопку **<p>**.) А кнопка **<body>** выделяет всю Web-страницу (то есть все содержимое тега `<BODY>`).

Давайте поставим текстовый курсор на последний абзац и щелкнем в секции тегов строки статуса кнопку **<p>**. После этого весь последний абзац будет выделен.

Чтобы превратить шрифт, которым набран выделенный фрагмент текста, в курсивный, нужно нажать кнопку-выключатель **I**, находящуюся в редакторе свойств. Также можно воспользоваться комбинацией клавиш <Ctrl>+<I>.

Кстати, секция тегов поможет нам в том случае, если нужно удалить какой-либо элемент страницы. Достаточно щелкнуть по кнопке, соответствующей нужному тегу, и нажать клавишу <Del>.

Аппетит приходит во время еды. Давайте еще что-нибудь сотворим с нашим текстом. Например, изменим шрифт, которым набрана строка `Здравствуйте, уважаемые посетители нашего сайта!`, немного увеличим его размер и сделаем эту строку красной. Разумеется, сначала мы должны ее выделить, для чего воспользуемся секцией тегов.

Для смены шрифта используется раскрывающийся список **Font**, показанный на рис. 3.9, а для смены размера шрифта — список **Size**, показанный на рис. 3.10. Оба этих списка находятся все в том же редакторе свойств. Давайте посмотрим, какой выбор они нам предлагают.

Рис. 3.9. Раскрывающийся список **Font**, находящийся в редакторе свойств

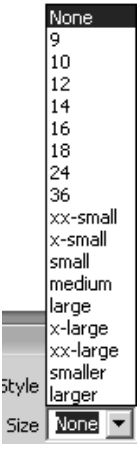


Рис. 3.10. Раскрывающийся список **Size**, расположенный в редакторе свойств

Раскрывающийся список **Font** содержит несколько пунктов, представляющих так называемые *стандартные шрифты* HTML, которые могут быть использованы на Web-страницах. Предполагается, что эти шрифты установлены на любом клиентском компьютере, поэтому никаких проблем с отображением текста на Web-страницах быть априори не должно. Стандартных шрифтов немного, но в Web-дизайне больше и не нужно.

Замечание

В принципе, стандарты HTML и CSS не запрещают нам задать для текста любой шрифт. Но при этом нам самим придется позаботиться, чтобы у посетителей нашего сайта этот шрифт имелся.

Пункт **Default Font** этого списка задает для выделенного текста тот же шрифт, что и у родительского тега.

Также стандарты HTML и CSS определяют несколько стандартных величин размеров для шрифта. Им соответствуют пункты **xx-small**, **x-small**, **small**, **medium**, **large**, **x-large** и **xx-large** раскрывающегося списка **Size**. Они перечисляют все стандартные размеры в порядке от самого мелкого до самого крупного. Пункты **larger** и **smaller** задают размер шрифта соответственно на одну ступень больше и меньше размера шрифта родительского тега. Числовые пункты задают размер шрифта в пунктах; также можно ввести нужный нам размер вручную прямо в список. А пункт **None** делает размер шрифта таким же, как у родительского тега.

Давайте зададим для выделенной нами строки текста шрифт **Verdana, Arial, Helvetica, sans-serif**, для чего выберем соответствующий пункт списка **Font**. И увеличим на одну ступень размер этого шрифта, выбрав в списке **Size** пункт **larger**. А после этого давайте...

Стоп! Что это такое? В раскрывающемся списке **Style**, находящемся все в том же незаменимом редакторе свойств, появился пункт **style1**. И шрифт, которым набран этот пункт, в точности совпадает со шрифтом выделенной нами строки. Список **Style**, если следовать логике, предназначен для выбора стилевого класса (стили переопределения тегов, как мы знаем, привязываются автоматически), который будет привязан к выделенному тексту. Выходит, Dreamweaver автоматически создал стилевой класс и привязал его к этой строке? Давайте посмотрим!

Переключимся в режим отображения кода HTML и посмотрим на секцию заголовка Web-страницы. Так и есть — там находится свежесозданная таблица стилей с единственным стилем `style1` (цифра в конце имени может быть другой и даже может измениться впоследствии — это нормально). Его определение выглядит так:

```
.style1 { font-family: Verdana, Arial, Helvetica, sans-serif;
          font-weight: bold;
          font-size: larger; }
```

То есть он задает шрифт (атрибут `font-family`), размер (уже знакомый нам атрибут `font-size`) и полужирность шрифта! Dreamweaver убрал тег `<STRONG>` и внес в определение стиля атрибут `font-weight`, задающий эту самую полужирность (значение `bold`). Пожалуй, эта программа даже лучше, чем мы ожидали!

Осталось заставить выделенную нами строку "покраснеть". Для этого мы переключимся обратно в режим отображения Web-страницы и обратимся к особому инструменту Dreamweaver, предназначенному для выбора цвета, — *селектору цвета*. Он выглядит так — .

Мы уже знаем, как задать цвет в CSS — ввести его английское название: green (зеленый), red (красный) и т. д. Но таким образом можно задать далеко не все цвета, а только немногие. Большинство цветов задаются так называемым кодом RGB (Red, Green, Blue — красный, зеленый, синий) вот в таком формате: `#RRGGBB`. Здесь `RR` — это шестнадцатеричное число от 0 до FF, задающее долю в окончательном цвете красной составляющей, `GG` — зеленой, а `BB` — синей. Пример задания цвета в формате RGB — `#336699` (это тускло-голубой цвет).

Этот самый код RGB мы можем ввести в небольшое поле ввода, находящееся в правой части селектора цвета, и нажать клавишу <Enter>. Но ведь мы не знаем код нужного нам цвета! Поэтому лучше щелкнем по небольшой квадратной кнопке, находящейся в левой части селектора цвета. На экране появится небольшое *окно выбора цвета*, показанное на рис. 3.11.

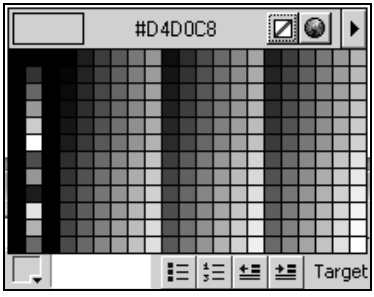


Рис. 3.11. Открытое окно выбора цвета

Большую часть этого окна занимает *палитра* — набор разноцветных квадратиков. Чтобы выбрать нужный цвет, достаточно щелкнуть по соответствующему квадратику. После этого окно выбора цвета закроется, а выбранный цвет будет тотчас применен. Если же нужно закрыть данное окно без выбора цвета, достаточно нажать клавишу <Esc>.

Давайте выберем в окне красный цвет. И сразу же переключимся в режим отображения HTML-кода, чтобы взглянуть на таблицу стилей. Dreamweaver

добавил в единственный стиль `style1` атрибут задания цвета `color` со значением, соответствующим коду RGB выбранного нами цвета.

Здесь нужно сказать еще несколько слов о цветах, применяемых в Web-дизайне. Дело в том, что разные компьютерные платформы — даже разные компьютеры — имеют разные параметры видеосистемы. Одни могут отображать всего шестнадцать цветов, а другие — все 16,7 миллионов, что с лихвой перекрывает цветовую разрешающую способность человеческого глаза. Разумеется, при таком богатом множестве компьютерных платформ Web-дизайнеру не стоит и рассчитывать, что все его цвета и оттенки будут везде отображены правильно.

Поэтому стандарт HTML определяет так называемую *безопасную палитру* цветов, которая гарантированно должна отображаться правильно всеми программами на всех компьютерах. Web-дизайнерам рекомендуется придерживаться этой безопасной палитры (хотя никто им не запрещает ее игнорировать). И палитра цветов, выводимая в окне выбора цвета Dreamweaver, также соответствует этой палитре.

Теперь сохраним нашу единственную Web-страницу. И приступим к работе над целыми абзацами текста.

Форматирование абзацев

Когда мы вводили текст Web-страницы, то предусмотрели три строки, которые собирались превратить в заголовки. Давайте же это сделаем.

Начнем с самой первой строки — `Наш архив`. Это будет название нашего сайта, давайте превратим ее в заголовок первого уровня. Поставим на нее текстовый курсор (выделять весь абзац целиком в этом случае необязательно) и посмотрим на редактор свойств. Чем он может нам помочь?

Там есть раскрывающийся список **Format**. Если его раскрыть, мы увидим то, что показано на рис. 3.12.

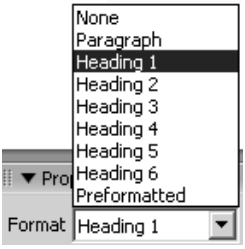


Рис. 3.12. Раскрывающийся список **Format**, находящийся в редакторе свойств

Здесь все просто. Пункт **Paragraph** этого списка форматирует текст как обычный абзац (отмечаемый тегом `<P>`). Пункты **Heading 1**, ..., **Heading 6**

позволяют превратить его в заголовок, соответственно, первого, ..., шестого уровня. Поэтому без лишних разговоров выберем пункт **Heading 1** — и название сайта создано.

Теперь создадим заголовки более низких уровней. Поставим текстовый курсор на строку `Обратная связь` и выберем в списке **Format** пункт **Heading 2**. То же самое проделаем со строкой `Права разработчиков`. Вот теперь все заголовки созданы.

Все это хорошо, но название нашего сайта выглядит как-то не очень аккуратно. Давайте выровняем его по центру. Редактор свойств припас нам специально для этого набор кнопок — . Эти кнопки задают выравнивание текста соответственно (порядок перечисления слева направо):

- по левому краю;
- по центру;
- по правому краю;
- выравнивание по ширине.

Снова поставим текстовый курсор на первую строку, которую мы уже превратили в название сайта, и щелкнем вторую слева кнопку в описанном ранее наборе. Все, название сайта мы отцентрировали.

Интересная особенность: когда мы щелкнули по второй кнопке упомянутого ранее набора, она остается нажатой. Если мы после этого щелкнем по другой кнопке, например, третьей (задающей выравнивание по правому краю), то третья кнопка нажмется, а вторая "отожмется". Такие кнопки, составляющие наборы, в которых только одна из них может быть нажата, называются *кнопками-переключателями*.

Переключимся в режим отображения HTML-кода и посмотрим, что поставил там Dreamweaver. Так, теги `<H1>` и `<H2>` расставлены там где нужно. Только вместо того, чтобы создать новый стиль, он вставил в тег `<H1>` атрибут `ALIGN` со значением `center`, задающий выравнивание по центру. Это, в принципе, не совсем то, чего мы хотели — используется атрибут физического форматирования — но пусть пока останется. Мы исправим это потом.

Теперь посмотрим на строки, перечисляющие разделы нашего сайта: `файлы` и `статьи`. Очень уж они нехорошо выглядят... Давайте превратим их в список HTML, для чего воспользуемся набором кнопок-переключателей , находящимся в редакторе свойств.

Список HTML выглядит как набор строк — пунктов списка, каждая из которых либо помечена особым значком, либо пронумерована. Списки с пунктами, помеченными значками (*маркерами*), называются *маркированными*, а списки с пронумерованными пунктами — *нумерованными*. Левая кнопка описанного ранее набора превращает выделенные строки в пункты маркированного списка, правая — в пункты нумерованного списка.

- файлы
- статьи.

Рис. 3.13. Маркированный список

1. файлы
2. статьи.

Рис. 3.14. Нумерованный список

Выделим строки `файлы` и `статьи` и нажмем левую кнопку набора. Созданный нами маркированный список будет выглядеть так, как показано на рис. 3.13.

Хотя, возможно, будет удобнее создать нумерованный список. Для этого достаточно выделить эти строки и нажать правую кнопку набора. Получится то, что показано на рис. 3.14. Что ж, пусть так и остается.

Если требуется превратить пункты какого-либо списка в обычные абзацы, то нужно выделить их и щелкнуть еще раз по нажатой в данный момент кнопке набора, чтобы "отжать" ее.

Переключимся в режим отображения HTML-кода и посмотрим, из чего состоит список HTML.

```
<OL>
  <LI>файлы</LI>
  <LI>статьи.</LI>
</OL>
```

Видно, что нумерованный список представляет собой парный тег `<OL>`, внутри которого находятся пункты этого списка. Каждый пункт, в свою очередь, представляет собой содержимое другого тега — `<LI>`. Ничего сложного.

Для создания маркированного списка вместо тега `<OL>` используется тег `<UL>`. Пункты списка и в этом случае создаются с помощью все того же тега `<LI>`.

Вот, собственно, и все, что можно рассказать о форматировании абзацев текста. Разумеется, если нам понадобится что-то большее, мы всегда можем заглянуть в интерактивную справку Dreamweaver и справочники по HTML и CSS, которые поставляются в его составе (о них будет рассказано в конце этой главы).

А сейчас давайте выясним, как вставить в текст Web-страницы некоторые специальные символы, в частности, знак "копирайта" (авторского права).

Специальные символы и нетекстовые элементы

Посмотрим на последнюю строку нашего текста, содержащую сведения об авторских правах. Общепринятым стандартом стало использование значка © ("копирайт"), у нас же — длинный некрасивый текст. Давайте заменим его коротким и приметным значком.

Сначала выделим слова `Авторские права принадлежат нам.` (не забыв и точку), которые мы заменим значком ©, и удалим. Далее поставим текстовый курсор в то место текста, где мы хотим поместить этот знак. И выберем

пункт **Copyright** подменю **Special Characters**, находящегося, в свою очередь, в подменю **HTML** меню **Insert**.

Dreamweaver выведет небольшое предупреждение (рис. 3.15), говорящее о том, что данный символ может не отобразиться корректно в случае использования заданной нами в настройках Dreamweaver кодировки. Закроем его, нажав кнопку **OK**, а чтобы оно не выводилось на экран в дальнейшем, перед закрытием включим флажок **Don't show me again**.

На рис. 3.16 показан символ ©, только что вставленный нами в текст. (Не забудем также исправить последнюю строку.)

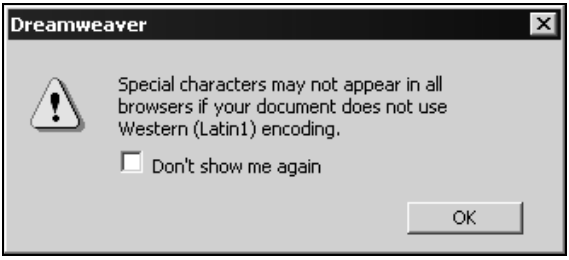


Рис. 3.15. Предупреждение о возможном некорректном отображении специального символа

© мы, разработчики, 2004 год.

Рис. 3.16. Символ © в тексте страницы

Здесь мы столкнулись с так называемыми *специальными символами* HTML. Эти символы из соображений все той же совместимости не могут быть просто так вставлены в код HTML, а заменяются либо числовыми кодами, либо мнемоническими обозначениями. В частности, символ © обозначается в HTML-коде как `©`. Вот так:

```
<P><EM>&copy; мы, разработчики, 2004 год.</EM></P>
```

Кстати, символ двойных кавычек (") обозначается как `"`, символ "меньше" (<) — как `<`, а символ "больше" (>) — как `>`. (Знак точки с запятой в конце обязателен — нужно иметь это в виду при правке HTML-кода!)

Еще один полезный специальный символ — *неразрывный пробел*. Что бы ни случилось, Web-обозреватель никогда не будет переносить строку по этому пробелу. Чтобы вставить такой пробел в текст, необходимо поставить в нужно место текстовый курсор, удалить обычный пробел, если он там есть, и нажать комбинацию клавиш <Ctrl>+<Shift>+<Пробел>. В коде HTML неразрывный пробел обозначается так — ` `.

Иногда нужно, наоборот, разорвать строку абзаца на две, да так, чтобы этот разрыв сохранялся всегда. Для этого нужно поставить текстовый курсор на нужное место и нажать комбинацию клавиш <Shift>+<Enter>. При этом в код HTML будет вставлен особый одинарный тег `<BR>` — тег *разрыва строк*.

Вдоволь наэкспериментировавшись с разрывами строк, неразрывными пробелами и прочими специальными символами, давайте займемся делом. Не кажется ли, что третий и четвертый абзацы нашей страницы расположены слишком близко друг к другу? Как бы их немного разнести?

Можно, конечно, вставить пустой абзац, поместив текстовый курсор в конец третьего абзаца и нажав клавишу <Enter>, но получится некрасиво. Давайте сделаем по-другому, а именно разделим этим абзацы *горизонтальной линией*.

Поместим текстовый курсор в начале четвертого абзаца и выберем пункт **Horizontal Rule** подменю **HTML** меню **Insert**. Созданная нами горизонтальная линия показана на рис. 3.17.

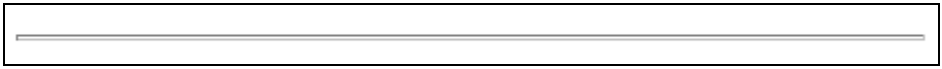


Рис. 3.17. Горизонтальная линия

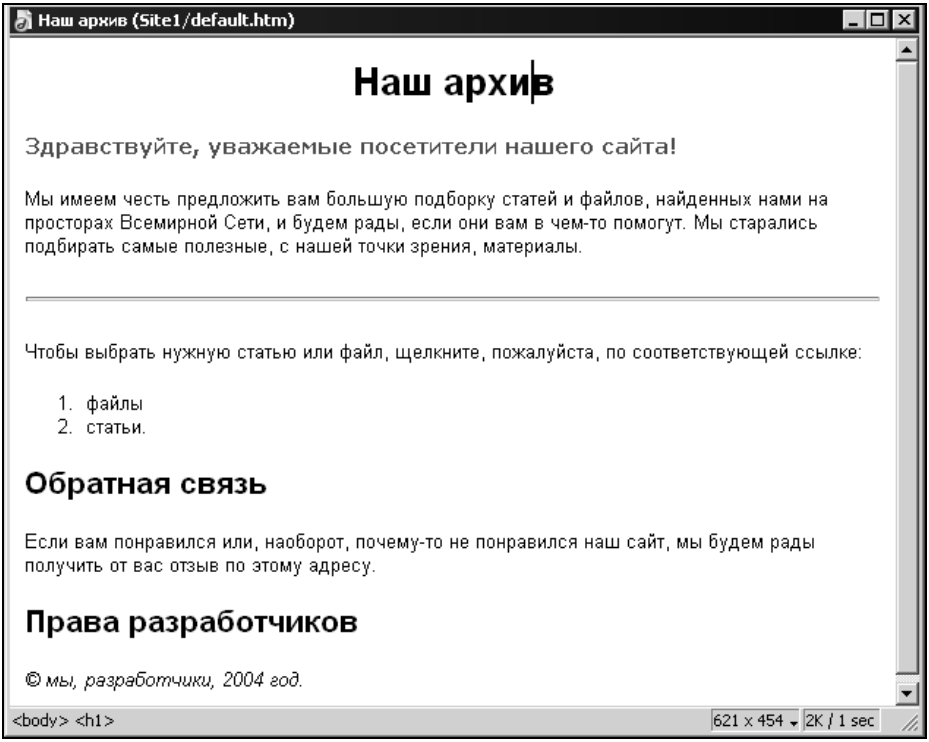


Рис. 3.18. Готовая Web-страница default.htm сайта Site1

Чтобы удалить вставленную не в то место горизонтальную линию, нужно выделить ее щелчком мыши и нажать клавишу <Del>.

Горизонтальная линия создается одинарным тегом `<HR>` и относится к так называемым *нетекстовым элементам*. Нетекстовые элементы определяются в самом коде HTML, но не относятся к тексту. Кроме горизонтальных линий, к ним относятся еще и таблицы, с которыми мы сейчас начнем работать.

А пока что сохраним созданную Web-страницу default.htm и проверим, все ли мы сделали правильно. Готовая страница должна выглядеть так, как показано на рис 3.18. После этого закроем готовую страницу, щелкнув кнопку закрытия окна документа, в котором она открыта, выбрав пункт **Close** меню **File** или нажав комбинацию клавиш <Ctrl>+<W>.

Главная страница нашего сайта, в принципе, готова. Приступим к созданию остальных страниц.

Работа с таблицами

Если нужно поместить на ограниченном пространстве Web-страницы множество числовых (и не только числовых) данных, нет лучшего средства, чем таблица. Если необходимо создать красивый список, снова на помощь приходит таблица. Таблицы заполонили Web-документы. И немудрено: при нескольких не слишком значительных недостатках они обладают массой достоинств.

Поэтому для представления на своих Web-страницах списков категорий и самих файлов и статей мы используем именно таблицы. Это действительно удобная штука — уж поверьте автору!

Но сначала давайте создадим новую Web-страницу. Пусть это будет страница категорий статей. Дадим ей название `Статьи`, напишем какой-либо вводный текст, сохраним под именем Articles.htm в папке Site1 и приготовимся к знакомству с таблицами.

Создание таблиц

Поставим текстовый курсор в конце самого последнего абзаца и нажмем клавишу <Enter>, чтобы создать пустой абзац. (Если он уже есть, то его создавать не нужно.) Именно здесь мы и создадим нашу первую таблицу.

Пустая таблица создается выбором пункта **Table** меню **Insert** или нажатием комбинации клавиш <Ctrl>+<Alt>+<T>. На экране появится диалоговое окно **Table**, показанное на рис. 3.19.

В полях ввода **Rows** и **Columns** этого окна вводятся, соответственно, количество строк и столбцов создаваемой таблицы. Введем в поле **Rows** число 5 — пока наша таблица будет содержать пять строк. А в поле **Columns** введем число 2.

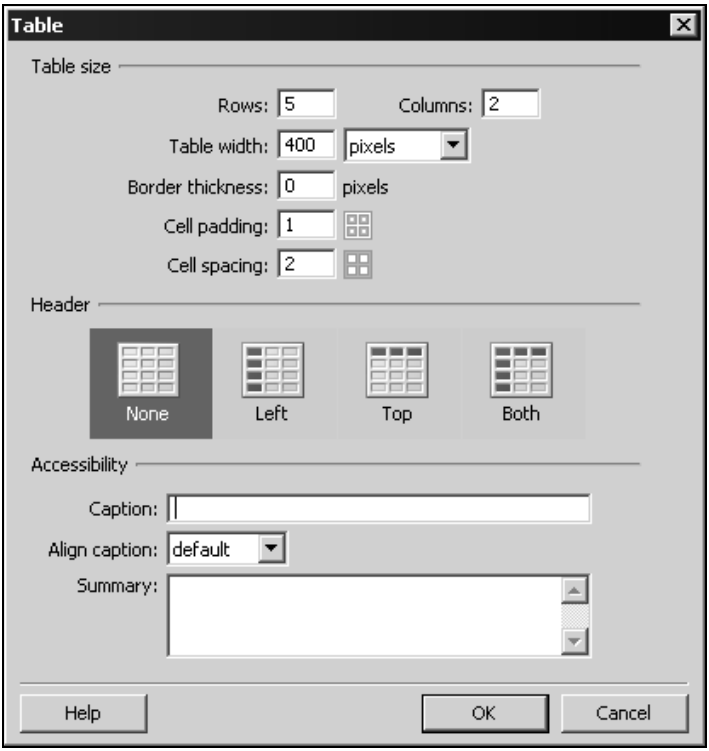


Рис. 3.19. Диалоговое окно **Table**

В поле ввода **Table width** задается ширина таблицы в пикселах или процентах от ширины родителя. В раскрывающемся списке, расположенном справа от этого поля ввода, нужно будет выбрать соответственно пункт **pixels** или **percent**. Давайте зададим ширину таблицы, равную 400 пикселов — этого хватит для вывода списка категорий.

В поле ввода **Border thickness** задается толщина границ таблицы в пикселах. По умолчанию она равна 1. Мы можем ввести 0, чтобы убрать границы совсем; давайте так и сделаем.

В поле ввода **Cell padding** задается расстояние между границей ячейки таблицы и ее содержимым в пикселах. По умолчанию оно равно 1, пусть таким и остается.

Аналогично, поле ввода **Cell spacing** служит для задания расстояния между границами отдельных ячеек. По умолчанию оно равно 2 — пусть таким и остается.

Остальные элементы управления нам пока не пригодятся. Поэтому сразу нажмем кнопку **OK**. В результате у нас должно получиться что-то похожее на рис. 3.20.

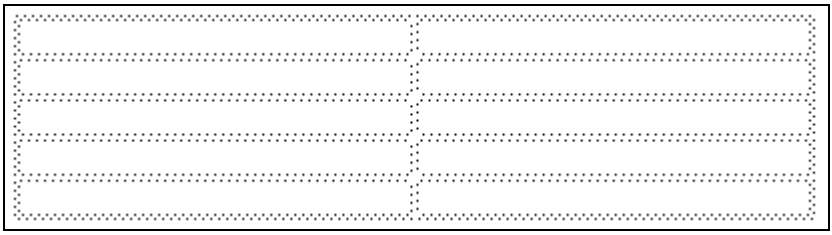


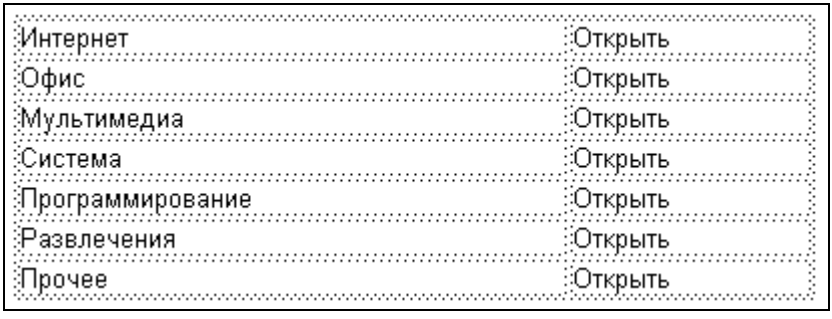
Рис. 3.20. Наша первая таблица

Интересно, что, хотя ранее мы задали нулевую толщину границы таблицы (в поле ввода **Border thickness** диалогового окна **Table**), Dreamweaver все равно отобразит так называемую *невидимую границу*, состоящую из тонких точечных линий. В Web-обозревателе эта граница отображаться не будет. Если бы мы задали ненулевую толщину *видимой границы* таблицы, Dreamweaver отобразил бы ее в виде сплошной линии.

Теперь поставим текстовый курсор в первую (верхнюю левую) ячейку таблицы и начнем набирать список категорий. Набрав содержимое первой ячейки, нажмем клавишу <Tab>, чтобы переместить текстовый курсор во вторую ячейку таблицы, и продолжим создание списка.

Dreamweaver поможет нам в этом. Когда мы наберем содержимое последней ячейки таблицы и нажмем клавишу <Tab>, он добавит в таблицу еще одну строку. Так что нам не придется заниматься этим вручную. Но если же нам понадобится вставить в таблицу новую строку, нам будет нужно поместить текстовый курсор в любую ячейку строки, над которой требуется вставить новую строку, и либо выбрать пункт **Insert Row** в подменю **Table** меню **Modify**, либо нажать комбинацию клавиш <Ctrl>+<M>.

В результате у нас должно получиться нечто, похожее на рис. 3.21. Пока мы не будем заниматься оформлением текста списка, а лучше поэкспериментируем с нашей первой таблицей.



Интернет	Открыть
Офис	Открыть
Мультимедиа	Открыть
Система	Открыть
Программирование	Открыть
Развлечения	Открыть
Прочее	Открыть

Рис. 3.21. Таблица — список категорий с заполненными ячейками

Работа с таблицей

Мы уже знаем, как можно вставить в таблицу новую строку. Для этого нужно поместить текстовый курсор в ячейку строки, над которой должна быть добавлена новая, и выбрать пункт **Insert Row** в подменю **Table** меню **Modify** или нажать комбинацию клавиш <Ctrl>+<M>.

Аналогично можно вставить в таблицу и новый столбец. Ставим текстовый курсор в ячейку, справа от которой должен появиться новый столбец, и выберем пункт **Insert Column** в подменю **Table** меню **Modify** или нажмем комбинацию клавиш <Ctrl>+<Shift>+<A>.

Удалить ненужную строку или столбец так же просто. Для удаления строки ставим текстовый курсор в ячейку удаляемой строки и либо выбираем пункт **Delete Row** в подменю **Table** меню **Modify**, либо нажимаем комбинацию клавиш <Ctrl>+<Shift>+<M>. А для удаления столбца, в ячейке которого находится текстовый курсор, нужно либо выбрать пункт **Delete Column** в подменю **Table** меню **Modify** или контекстного меню, либо нажать комбинацию клавиш <Ctrl>+<Shift>+<->.

Удалить же саму таблицу проще всего, воспользовавшись секцией тегов. Ставим текстовый курсор в любую ячейку нужной таблицы, щелкаем по кнопке **<table>** секции тегов и нажимаем клавишу <Del>.

Когда мы вводили в ячейки таблицы текст, размеры ячеек изменялись автоматически, чтобы вместить их содержимое. Конечно, это полезно, но зачастую эти размеры изменяются самым непонятным образом. Чтобы такого не случилось, давайте сами зададим размеры ячеек, чтобы Web-обозреватель не своевольничал.

Поместим курсор мыши на вертикальную границу между ячейками и перетащим ее так, чтобы правый столбец был как можно у́же, чтобы вмещать только слово `Открыть`. После этого ширина обоих столбцов нашей таблицы будет жестко зафиксирована.

Точно так же мы можем менять высоту строк, перетащив мышью границу между нужными ячейками. И последний штрих — изменение ширины и высоты таблицы перетаскиванием ее правой или нижней границы.

Наэкспериментировавшись вдоволь с изменением размеров таблицы, ее строк и столбцов, давайте сохраним ее. И для разнообразия переключимся в режим отображения кода HTML, чтобы посмотреть, с помощью каких тегов формируется созданная нами таблица. И увидим там такое нагромождение тегов, что страшно станет.

Как формируются таблицы

Но на самом деле ничего страшного там нет. Достаточно понять принцип, по которому формируется таблица HTML, чтобы с легкостью ориентиро-

ваться в ее коде. Давайте рассмотрим код HTML, с помощью которого формируется таблица, по частям.

Сначала нам нужно создать саму таблицу. Это выполняется с помощью парного тега `<TABLE>`:

```
<TABLE WIDTH="400" BORDER="0" CELLSPACING="2" CELLPADDING="1">
</TABLE>
```

Атрибут `WIDTH` задает ширину таблицы, атрибут `BORDER` — толщину видимой границы, атрибуты `CELLSPACING` и `CELLPADDING` — соответственно расстояние между границами соседних ячеек и между границей ячейки и ее содержимым. Все эти параметры мы задали в диалоговом окне **Table** (см. рис. 3.19).

Далее с помощью парных тегов `<TR>` мы формируем строки таблицы:

```
<TABLE WIDTH="400" BORDER="0" CELLSPACING="2" CELLPADDING="1">
  <TR>
  </TR>
</TABLE>
```

Здесь мы сформировали только одну строку — для примера этого будет достаточно. Заметим, что тег `<TR>` может находиться только внутри тега `<TABLE>`, в противном случае Web-обозреватель обработает его неправильно.

Следующий шаг — формирование ячеек таблицы с помощью парных тегов `<TD>`:

```
<TABLE WIDTH="400" BORDER="0" CELLSPACING="2" CELLPADDING="1">
  <TR>
    <TD WIDTH="327">Интернет</TD>
    <TD WIDTH="63">Открыть</TD>
  </TR>
</TABLE>
```

Здесь мы поместили в строку две ячейки, содержащие текст `Интернет` и `Открыть` соответственно. Атрибут `WIDTH` и в этом случае задает ширину ячейки. Опять же, тег `<TD>` может находиться только внутри тега `<TR>`, иначе Web-обозреватель не сможет его обработать.

Все это может показаться очень сложным. Но на самом деле ничего сложного здесь нет — наоборот, все очень просто, если понять принцип. Более того, подобный способ формирования таблиц исключительно гибок, и именно в жертву гибкости была принесена компактность HTML-кода таблицы.

Вообще, таблицы HTML — это исключительно мощная вещь! Так, в ячейку таблицы можно поместить все что угодно: сколь угодно большой текст, графическое изображение и даже другую таблицу. Нужный код HTML просто

помещается внутрь соответствующего тега `<TD>` — и вуаля! Единственное: не запутаться в хитросплетениях HTML-кода самому и не свести с ума Web-обозреватель, которому все это придется выводить на экран.

Более сложные таблицы

Давайте сохраним и закроем нашу Web-страницу Articles.htm. Сейчас мы займемся созданием страницы, содержащей список статей на тему `Интернет`. Создадим новую пустую Web-страницу, дадим ее название `Статьи — Интернет`, введем какой-либо вступительный текст и сохраним под именем Articles_internet.htm. Вообще, будем стараться давать файлам наших страниц "говорящие" имена — так нам самим будет проще в дальнейшем.

Список статей мы поместим, опять же, в таблицу. Эта таблица будет содержать три столбца: имя автора, название статьи и гиперссылку вида `Открыть`, как и в списке категорий. Поставим текстовый курсор после вводного текста и выберем пункт **Table** меню **Insert** или нажмем комбинации клавиш <Ctrl>+<Alt>+<T>. На экране появится диалоговое окно **Table** (см. рис. 3.19).

Пусть наша таблица содержит две строки (поле ввода **Rows**) и три, как мы договорились заранее, столбца (поле ввода **Columns**). По ширине она пусть занимает всю Web-страницу, поэтому введем в поле ввода **Table width** значение 100, а в раскрывающемся списке справа выберем пункт **percent**. Дадим нашей второй по счету таблице видимую границу толщиной в 1 пиксел, введя в поле ввода **Border thickness** единицу. А значения полей ввода **Cell padding** и **Cell spacing** пусть останутся равными 1 и 2 соответственно.

И давайте дадим нашей таблице нормальную "шапку". Для этого воспользуемся набором переключателей **Header**, расположенным в нижней половине окна **Table**. Каждый из этих переключателей имеет "говорящий" рисунок, поэтому ясно, что нам нужно включить переключатель **Top**. Что мы и сделаем.

Осталось только нажать кнопку **OK**, чтобы Dreamweaver создал для нас таблицу. Ну и, разумеется, заполнить ячейки созданной таблицы содержимым. У нас должна получиться вот такая симпатичная табличка — см. рис. 3.22.

Автор	Название	
Дуболом, Е.	К вопросу о прочности модемов	Открыть
Криворукий, Ю.	Тяп-ляп Web-дизайнер	Открыть
Сусанин-мл., И.	Путеводитель по Всемирной Паутине	Открыть

Рис. 3.22. Таблица списка статей

Давайте-ка внимательно посмотрим на этот рисунок. Что в нем примечательного?

Во-первых, конечно, наша таблица обзавелась красивой видимой границей в виде тонкой сплошной "трехмерной" линии. Более того, эти линии окружают как саму таблицу, так и ее отдельные ячейки, так что граница фактически получается двойной. Выглядит она действительно эффектно; кажется, будто ячейки как бы вдавлены в фон Web-страницы.

Во-вторых, текст, находящийся в ячейках первой строки таблицы, набран полужирным шрифтом, хотя мы явно не делали его таким. Почему? Дело в том, что Dreamweaver сформировал ячейки первой строки немного по-другому: не тегом `<TD>`, а тегом `<TH>`. А содержимое тега `<TH>` всегда выделяется полужирным шрифтом.

Вот фрагмент HTML-кода таблицы, описывающий ее первую строку:

```
<TABLE WIDTH="100%"  BORDER="1" CELLSPACING="2" CELLPADDING="1">
  <TR>
    <TH SCOPE="col">Автор</TH>
    <TH SCOPE="col">Название</TH>
    <TH SCOPE="col"> </TH>
  </TR>
  . . .
```

Атрибут `SCOPE` со значением `col` определяет ячейку `<TH>` как заголовок для всего столбца таблицы. В принципе, он необязателен и даже толком не поддерживается современными Web-обозревателями, но Dreamweaver решил перестраховаться. Не будем обращать на этот атрибут внимания.

В последнюю, пустую, ячейку Dreamweaver поместил символ неразрывного пробела ` `. Это необходимо, чтобы таблица правильно отображалась в старых программах Web-обозревателей. Лучше этот символ не убирать.

Так, вроде бы, все замечательно. Только вот третья по счету ячейка в первой строке осталась пустой и выглядит не очень красиво. Давайте объединим ее со второй ячейкой, превратив две ячейки в одну.

Сначала нам нужно выделить две объединяемые ячейки. Поместим курсор мыши во вторую ячейку, нажмем левую кнопку мыши и, не отпуская ее, протащим вправо, пока обе объединяемые ячейки не будут выделены толстой черной линией. После этого нажмем кнопку , находящуюся в редакторе свойств. Результат показан на рис. 3.23.

Автор	Название	
Дуболом, Е.	К вопросу о прочности модемов	Открыть
Криворукий, Ю.	Тяп-ляп Web-дизайнер	Открыть
Сусанин-мл., И.	Путеводитель по Всемирной Паутине	Открыть

Рис. 3.23. Результат объединения ячеек

Точно так же можно объединять ячейки и по вертикали. Причем объединить в одну можно сколько угодно ячеек — стандарты HTML не ограничивают их количество. Если же требуется разъединить объединенную ячейку на несколько, то нужно поставить в нее текстовый курсор и нажать кнопку , которая находится также в редакторе свойств. После этого на экране появится диалоговое окно **Split Cell**, показанное на рис. 3.24.

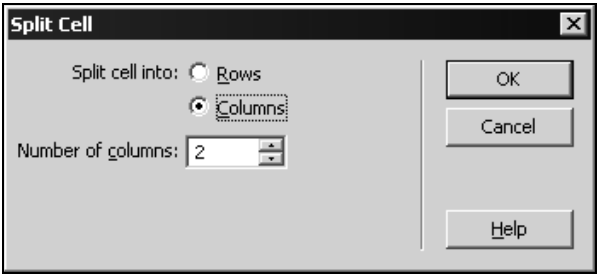


Рис. 3.24. Диалоговое окно **Split Cell**

Группа переключателей **Split cell into** задает, как будет делиться ячейка. Переключатель **Rows** задает разделение ячейки по горизонтали, на несколько строк, количество которых устанавливается в поле счетчика **Number of rows**. Если же выбран переключатель **Columns**, ячейка будет делиться по вертикали, на несколько столбцов, количество которых задается в поле счетчика **Number of columns**. После нажатия кнопки **OK** объединенная ячейка будет разделена.

Теперь давайте посмотрим на HTML-код нашей таблицы. Вот он:

```
<TABLE WIDTH="100%"  BORDER="1" CELLSPACING="2" CELLPADDING="1">
  <TR>
    <TH SCOPE="col">Автор</TH>
    <TH COLSPAN="2" SCOPE="col">Название</TH>
  </TR>
  . . .
```

Здесь добавился атрибут `COLSPAN`, задающий количество объединяемых по горизонтали ячеек. Видно, что он находится в теге `<TH>` самой левой из объединяемых ячеек. Что касается третьей ячейки, то она в HTML-коде не описывается, так как фактически становится частью второй, объединенной ячейки.

Объединение ячеек по вертикали выполняется с помощью аналогичного атрибута `ROWSPAN`. Вот небольшой пример HTML-кода таблицы с объединенными по вертикали ячейками:

```
<TABLE WIDTH="100%" BORDER="1" CELLSPACING="2" CELLPADDING="1">
  <TR>
```

```
    <TD ROWSPAN="2">Дуболом, Е.Криворукий, Ю.</TD>
    <TD>К вопросу о прочности модемов</TD>
    <TD>Открыть</TD>
  </TR>
  <TR>
    <TD>Тяп-ляп Web-дизайнер</TD>
    <TD>Открыть</TD>
  </TR>
  <TR>
    <TD>Сусанин-мл., И.</TD>
    <TD>Путеводитель по Всемирной Паутине</TD>
    <TD>Открыть</TD>
  </TR>
</TABLE>
```

Первые ячейки первой и второй строк этой таблицы объединены в одну — об этом говорит атрибут `ROWSPAN` со значением, равным 2. При этом во второй строке первая ячейка также не определяется, так как она становится частью объединенной ячейки.

Вот мы и закончили с таблицами. Теперь давайте поговорим о том, как в Dreamweaver поместить на Web-страницу графическое изображение.

Вставка графических изображений

Посмотрим еще раз на страницу Articles_internet.htm. Что еще с ней можно сделать? В частности, третий столбец таблицы занимают будущие гиперссылки вида `Открыть`. Выглядят они, честно говоря, не очень аккуратно. Давайте их заменим на что-нибудь более приличное.

Откроем любой графический редактор, позволяющий сохранять изображения в формате GIF (вполне подойдет поставляющийся в составе Windows простейший редактор Paint). Нарисуем в нем небольшую картинку в виде направленной вправо стрелки и сохраним ее в файле под именем Arrow.gif. И заменим ею наши текстовые ссылки.

Прежде всего, удалим из ячейки третьего столбца текст `Открыть`. После этого проверим, установлен ли в эту ячейку текстовый курсор, и выберем пункт **Image** в меню **Insert** или нажмем комбинацию клавиш <Ctrl>+<Alt>+<I>. На экране появится диалоговое окно **Select Image Source**, показанное на рис. 3.25.

Раскрывающийся список папок и список файлов позволят нам выбрать нужную папку и файл. В поле ввода **Имя файла** появится имя выбранного файла (его также можно ввести туда вручную). Все это знакомо вам по стан-

дартным диалоговым окнам открытия и сохранения файлов Windows. Единственное отличие — справа находится панель предварительного просмотра, где в данный момент видна нарисованная нами стрелка. Ее, кстати, можно убрать, просто отключив флажок **Preview images**.

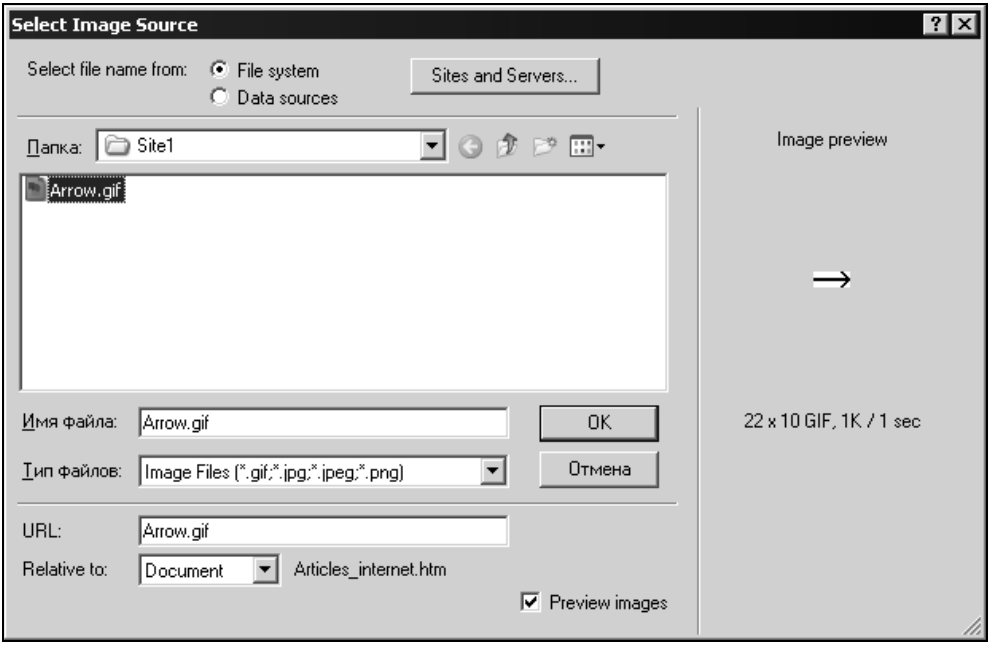


Рис. 3.25. Диалоговое окно **Select Image Source**

Итак, мы выбрали файл, где хранится наша стрелка. Осталось нажать кнопку **OK**. После этого выбранное нами изображение будет помещено на Web-страницу.

Сразу же после вставки Dreamweaver выделит это изображение. (Если же оно не выделено, то нужно щелкнуть по нему мышью.) Выделенное изображение окружено тонкой черной рамкой, на правой и нижней границе которой имеются небольшие черные квадратики. Это так называемые *маркеры изменения размера*. Мы можем "захватить" мышью любой маркер и перетащить его на новое место, изменив тем самым горизонтальный или вертикальный размер изображения. А если нам нужно изменить оба размера пропорционально, перетащим мышью маркер, находящийся в правом нижнем углу рамки, при нажатой клавише <Shift>.

Наша стрелка слишком маленькая — не всякий пользователь попадет по ней с первого раза. Поэтому давайте немного "вытянем" ее по горизонтали, чтобы она заполнила всю третью ячейку таблицы.

Изменить размеры изображения можно и по-другому. Если выделить изображение, редактор свойств отобразит набор элементов управления для задания различных параметров этого изображения (рис. 3.26). В том числе, в нем появятся поля ввода **W** и **H**, в которых задаются соответственно ширина и высота выделенного изображения.

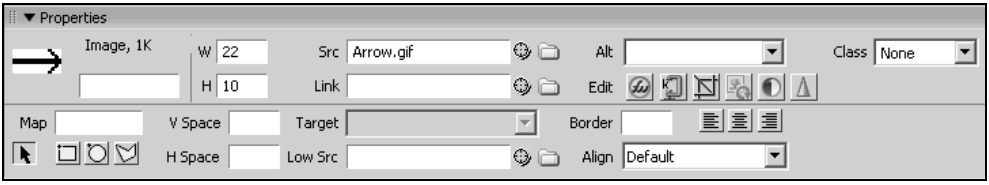


Рис. 3.26. Вид редактора свойств при выделенном графическом изображении

В редакторе свойств также находится раскрывающийся список **Alt**, служащий для задания альтернативного текста — текста, который Web-обозреватель покажет на странице, если почему-то не сможет загрузить файл изображения. Нужный текст вводится прямо в этот список, как в поле ввода.

Чтобы удалить ненужное или неудачно вставленное изображение, достаточно выделить его щелчком мыши и нажать клавишу <Del>.

Теперь давайте поместим такую же стрелку в остальные ячейки третьего столбца таблицы, для чего:

1. Выделим изображение стрелки щелчком мыши, если оно еще не выделено.
2. Скопируем его в буфер обмена Windows, нажав комбинацию клавиш <Ctrl>+<C> (можно также выбрать пункт **Copy** в меню **Edit**, но это долго).
3. Удалим текст `Открыть` из второй ячейки третьего столбца и поставим в нее текстовый курсор.
4. Вставим в ячейку изображение из буфера обмена, нажав комбинацию клавиш <Ctrl>+<P> (или выбрав пункт **Paste** меню **Edit**).
5. Повторим пункты 3 и 4 для третьей ячейки третьего столбца таблицы.

Сохраним страницу Articles_internet.htm и закроем ее. После этого откроем страницу default.htm. Для этого выберем пункт **Open** меню **Edit** или нажмем комбинацию клавиш <Ctrl>+<O>. На экране появится стандартное диалоговое окно открытия файла Windows; выберем в нем нужный нам файл и нажмем кнопку открытия.

Пока отложим создание остальных страниц нашего сайта. Давайте лучше узнаем, как с помощью Dreamweaver создаются гиперссылки.

Создание гиперссылок

Поскольку страница со списком категорий статей Articles.htm у нас уже готова, давайте создадим на странице default.htm гиперссылку, указывающую на нее. Ясно, что гиперссылкой станет надпись `статьи`, поэтому давайте ее выделим.

Способов создать гиперссылку в Dreamweaver существует несколько. Давайте рассмотрим самый простой для начинающих Web-дизайнеров. Выберем пункт **Make Link** меню **Modify** или нажмем комбинацию клавиш <Ctrl>+<L>. На экране появится диалоговое окно **Select File**, показанное на рис. 3.27. Осталось только выбрать нужный нам файл в списке файлов, занимающем бóльшую часть этого окна, и нажать кнопку **OK**.

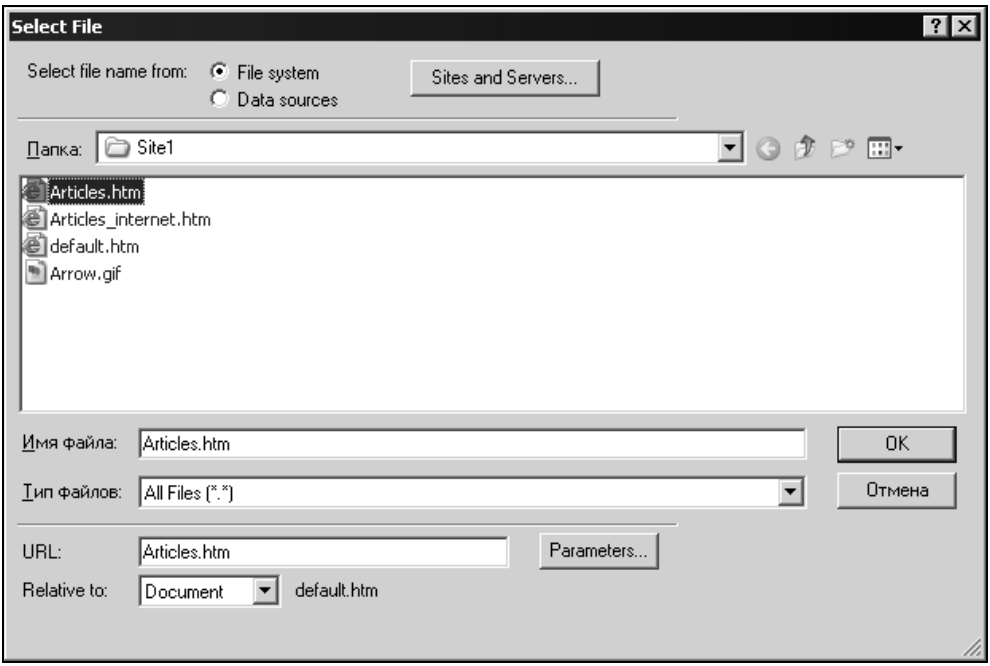


Рис. 3.27. Диалоговое окно **Select File**

Приведенный ранее способ позволяет создать только гиперссылку с относительным интернет-адресом, указывающую на файл относительно текущей Web-страницы. Если же нужно создать гиперссылку с абсолютным адресом, содержащим имя сервера, то нужный адрес достаточно ввести в поле ввода **URL** диалогового окна **Select File** и, разумеется, нажать кнопку **OK**.

Если же нам точно известно имя Web-страницы, ссылку на которую нужно создать, то мы можем поступить еще проще. В редакторе свойств (см. рис. 3.7)

имеется раскрывающийся список **Link**, в котором, как в поле ввода, вводится путь и имя нужного файла. (Если это имя вводилось ранее, мы также можем выбрать его из списка.) После ввода имени файла нужно нажать клавишу <Enter> — и Dreamweaver создаст для нас гиперссылку.

С помощью этого раскрывающегося списка можно и изменить интернет-адрес гиперссылки, если мы ошиблись при его вводе. Удалить же гиперссылку проще всего, поставив в нее текстовый курсор, щелкнув по кнопке **<a>** секции тегов и нажав клавишу <Del>.

Ниже раскрывающегося списка **Link** находится другой раскрывающийся список — **Target**. С помощью этого списка задается цель гиперссылки. Пункт **_blank** предписывает Web-обозревателю открыть Web-страницу, на которую указывает гиперссылка, в новом окне, а пункт **_self** или отсутствие любого значения — в том же окне.

Гиперссылка, указывающая на адрес электронной почты (*почтовая гиперссылка*), создается несколько по-другому. Выделим слова `по этому адресу` и выберем пункт **Email Link** в меню **Insert**. На экране появится небольшое диалоговое окно **Email Link**, показанное на рис. 3.28.

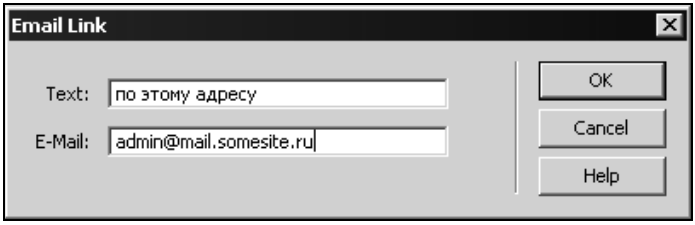


Рис. 3.28. Диалоговое окно **Email Link**

В поле **Text** самим Dreamweaver подставляется текст, выделенный в окне документа (в нашем случае — слова `по этому адресу`). В поле **E-Mail** вводим нужный нам почтовый адрес. После этого остается нажать кнопку **OK**.

Если мы после этого переключимся в режим отображения HTML-кода, то увидим, что интернет-адрес созданной нами почтовой гиперссылки имеет вид:

```
mailto:admin@mail.somesite.ru
```

То есть от гиперссылки, указывающей на файл, она отличается наличием символов `mailto:` перед почтовым адресом, причем между ними не должно быть пробела.

Ну и, разумеется, мы можем ввести почтовый адрес в представленном ранее виде в раскрывающийся список **Link** редактора свойств. Это, пожалуй, будет проще и быстрее.

В гиперссылку можно превратить не только строку текста, но и графическое изображение. Так, мы можем создать гиперссылку, содержимым которой

будет одна из стрелок, помещенных нами в ячейки третьего столбца таблицы на Web-странице Articles_internet.htm. Делается это точно так же, как было описано ранее: выделяем стрелку, вводим нужный интернет-адрес в раскрывающийся список **Link** редактора свойств и нажимаем клавишу <Enter>.

Кстати, будет лучше, если мы также зададим в качестве цели гиперссылки новое окно Web-обозревателя, выбрав в раскрывающемся списке **Target** редактора свойств пункт **_blank**. Пусть все эти статьи открываются в новом окне — так будет удобнее для посетителя.

И еще. На всех страницах, содержащих списки категорий файлов и статей, нужно создать гиперссылки, указывающие на главную страницу. А на страницах, содержащих ссылки на сами файлы и статьи, также должна быть гиперссылка на страницы списка соответствующей категории. Посетители сайта скажут нам за это спасибо.

Работа со стилями CSS

Ну вот, все страницы нашего сайта созданы. (Процесс их создания здесь не описывается, так как он достаточно очевиден.) Теперь можно немного, как сказал поэт, подумать "о красе ногтей". А именно — оформить наши странички с помощью стилей CSS.

Для работы со стилями нам понадобится панель **CSS Styles**. (Панель — это, как мы уже узнали, небольшое окно Dreamweaver, содержащее различные полезные инструменты.) Посмотрим, есть ли она на экране; если же нет, то включим пункт-выключатель **CSS Styles** в меню **Window** или нажмем комбинацию клавиш <Shift>+<F11>. Сама эта панель показана на рис. 3.29.

Рис. 3.29. Панель **CSS Styles**

Бóльшую часть этой панели занимает иерархический список всех действующих на редактируемой в данный момент Web-странице (как говорят опытные пользователи, *активном документе*) стилей с указанием, где какой стиль определен. Так, на рис. 3.29 видно, что из "корня" **<style>** растет чахлое "деревце" с единственной "ветвью" **.style2**. Это значит, что единственный стиль `style2` определен во внутренней таблице стилей (она-то и обозначается пунктом **<style>**) Web-страницы. Как мы помним, этот стиль вместе с таблицей создал сам Dreamweaver в Web-странице default.htm, которая как раз в данный момент открыта у автора.

Хорошо, страницу default.htm мы открыли. Давайте теперь сделаем так, чтобы все заголовки первого уровня (тег `<H1>`) выравнивались по центру без всяких наших усилий. Для этого нам будет нужно создать стиль переопределения тега `<H1>` и поместить его во внешнюю таблицу стилей, а ее потом — привязать ко всем страницам нашего сайта.

Нажмем маленькую кнопочку , находящуюся на нижнем крае панели **CSS Styles**. На экране появится диалоговое окно **New CSS Style**, показанное на рис. 3.30.

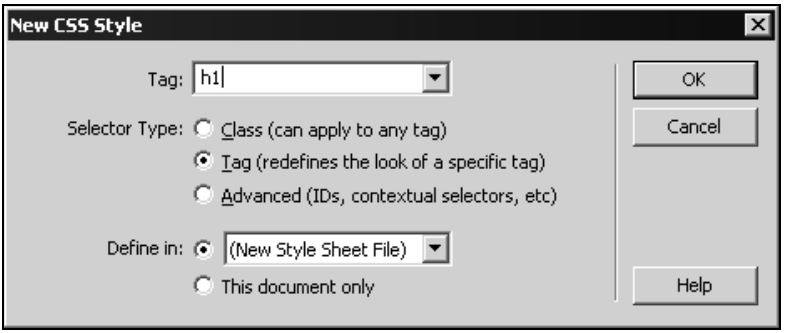


Рис. 3.30. Диалоговое окно **New CSS Style**

Здесь нам нужно включить переключатель **Tag** в группе **Selector Type**, задающий создание стиля переопределения тега. После этого в раскрывающемся списке **Tag** мы сможем выбрать нужный нам тег, а именно `<H1>`. Переключатель **Class** группы **Selector Type** заставляет Dreamweaver создать для нас стилевой класс, а переключатель **Advanced** — комбинированный стиль.

Теперь включим верхний переключатель в группе **Define in** и проверим, чтобы в расположенном правее раскрывающемся списке был выбран пункт **(New Style Sheet File)**. Таким образом мы дадим Dreamweaver знать, что нам нужно поместить создаваемый стиль во внешнюю таблицу стилей, которая также должна быть создана. Переключатель **This document only** группы **Define in** предписывает Dreamweaver поместить создаваемый стиль во внутреннюю таблицу стиля, что нам совсем не нужно.

Нажимаем кнопку **OK**. На экране появляется диалоговое окно сохранения файла Dreamweaver, похожее на аналогичное окно Windows. Вводим в поле имени файла styles.css — именно так, без затей будет называться наша внешняя таблица стилей — и нажимаем кнопку сохранения. Новая, пока еще пустая таблица стилей создана.

Теперь перед нами — диалоговое окно **CSS Style Definition**. В левой его части находится список **Category**, в котором выбирается категория атрибутов стилей, а сами эти атрибуты настраиваются с помощью элементов управления, находящихся в правой части этого окна.

Выберем в списке **Category** пункт **Block** — эта категория содержит атрибуты, затрагивающие формат абзацев текста. Окно **CSS Style Definition** примет такой вид — рис. 3.31.

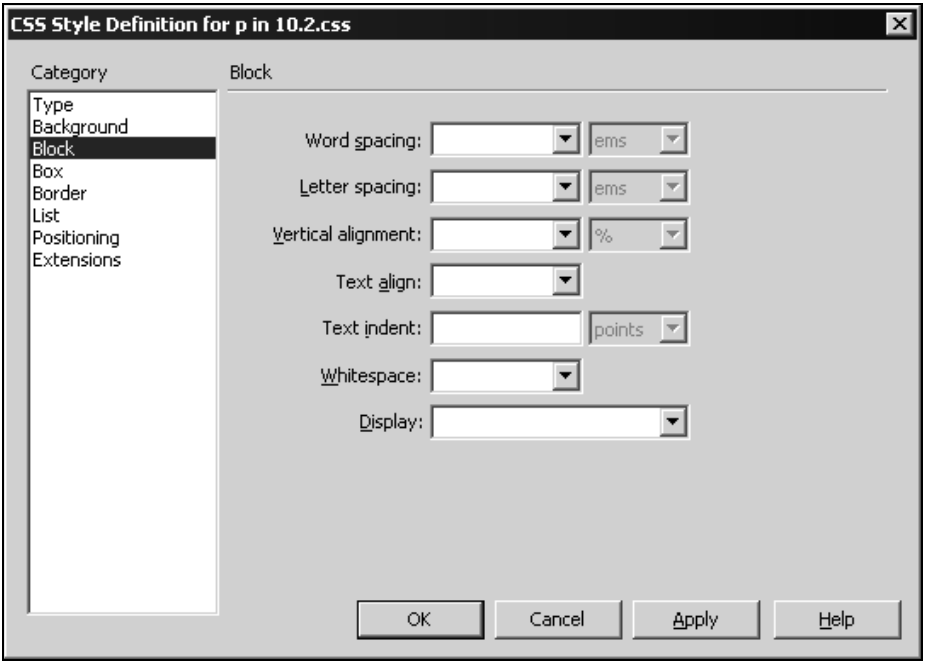


Рис. 3.31. Категория **Block** диалогового окна **CSS Style Definition**

Выравнивание абзаца задается с помощью раскрывающегося списка **Text align**. Выберем в нем пункт **center** и нажмем кнопку **OK**. В списке панели **CSS Styles** появится новое "дерево" с "корнем" **styles.css** (имя файла нашей внешней таблицы стилей) и ветвью **h1** (только что созданный нами стиль переопределения тега `<H1>`).

Теперь осталось выбрать окно документа, в котором открыта страница default.htm, переключиться в режим отображения кода HTML и убрать у тега

<H1> атрибут ALIGN вместе со значением. Ура, созданный нами стиль работает — название сайта все равно выровнено по центру!

Внимание

После того как внешняя таблица стилей будет создана, Dreamweaver откроет ее в отдельном окне документа. Не нужно забывать время от времени сохранять ее, иначе все наши стили могут быть потеряны, например, при внезапном отключении питания. (Впрочем, это относится и к Web-страницам.)

Теперь давайте закроем страницу default.htm и таблицу стилей styles.css, не забыв сохранить их. И откроем страницу Articles.htm. Сейчас мы привяжем к ней созданную нами таблицу стилей.

Список стилей в панели **CSS Styles** сейчас пуст, так как ни одной таблицы стилей к этой странице пока не привязано и внутренней таблицы стилей она не содержит. Нажмем кнопку , расположенную на нижнем крае панели **CSS Styles**. На экране появится небольшое диалоговое окно **Attach External Style Sheet**, показанное на рис. 3.32.

Рис. 3.32. Диалоговое окно **Attach External Style Sheet**

Нажмем кнопку **Browse** этого окна, в появившемся на экране диалоговом окне **Select File** выберем файл нашей таблицы стилей и нажмем кнопку открытия. После этого имя выбранного файла появится в поле ввода **File/URL**. Включим переключатель **Link**, чтобы привязать эту таблицу стилей к Web-странице, и нажмем кнопку **OK**. (Переключатель **Import** заставляет Dreamweaver поместить все определенные во внешней таблице стили во внутреннюю таблицу, а это нам не нужно.)

Опять же, выберем окно документа, в котором открыта страница Articles.htm, переключимся в режим отображения кода HTML и уберем у тега <H1> атрибут ALIGN вместе со значением. Все, действие внешней таблицы стилей распространилось и на эту страницу! Осталось повторить ту же самую процедуру для всех страниц нашего сайта.

Теперь создадим стиль для сведений об авторских правах. Сохраним все открытые Web-страницы, закроем их и откроем страницу default.htm. Опять

щелкнем по кнопке панели **CSS Styles**. Включим переключатель **Class** в группе **Selector Type** окна **New CSS Style**, в раскрывающийся список **Name** введем текст `.copyright`, проверим, включен ли верхний переключатель группы **Define in** и выбран ли в раскрывающемся списке правее него пункт **styles.css**. (Нам нужно поместить стиль `copyright` во внешнюю таблицу стилей styles.css, не так ли?) Нажимаем кнопку **OK** и попадаем в диалоговое окно **CSS Style Definition**.

Чтобы задать параметры шрифта, нам понадобятся элементы управления категории **Type**, так что выберем пункт **Type** в списке **Category** и увидим то, что показано на рис. 3.33. Выберем в раскрывающемся списке **Size** пункт **smaller**, а в раскрывающемся списке **Style** — пункт **italic**. Таким образом мы зададим для стилевого класса `copyright` уменьшенный курсивный шрифт. И нажмем кнопку **OK**.

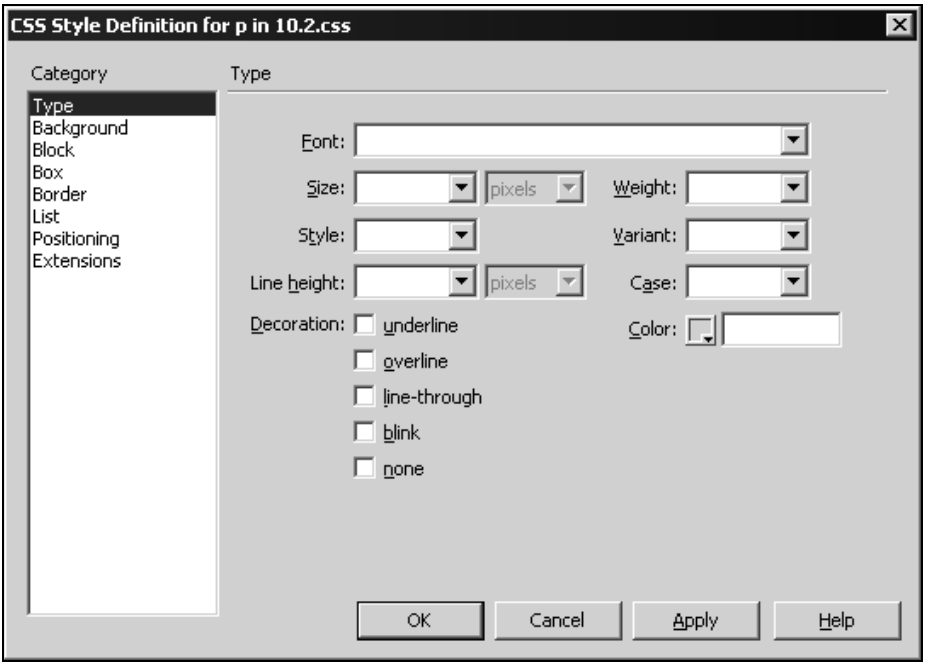


Рис. 3.33. Категория **Type** диалогового окна **CSS Style Definition**

Теперь выберем окно документа, в котором открыта страница default.htm. Нам нужно будет удалить тег `<EM>`, задающий курсивное начертание текста сведений об авторских правах, чтобы он не мешал нам, и привязать только что созданный стилевой класс к этому тексту (точнее, к тегу `<P>`).

Поставим текстовый курсор на текст `мы, разработчики`, щелкнем кнопку **** в секции тегов правой кнопкой мыши и выберем в появившемся

на экране контекстном меню пункт **Remove Tag**. Так мы удалим тег <EM>, но оставим его содержимое. (Если бы мы нажали клавишу <Del>, то был бы удален и тег, и его содержимое.) Теперь останется только щелкнуть по кнопке **<p>** секции тегов, дабы выделить весь абзац, и выбрать в раскрывающемся списке **Style** редактора свойств пункт **copyright**, соответствующий нужному стилевому классу. Все!

Да, с таблицами стилей можно много чего натворить... Но давайте же все-таки остановимся. В конце концов, о Web-дизайне рассказывают совсем другие книги; эта же посвящена больше интернет-программированию. Давайте лучше посмотрим, что еще может нам предложить Dreamweaver. Но сначала сохраним все, что мы успели натворить.

Предварительный просмотр Web-страниц

Хотя Dreamweaver в режиме просмотра страницы и представляет ее почти в таком виде, как она будет показана в Web-обозревателе, все же часто возникает необходимость увидеть ее в Web-обозревателе. Дело в слове "почти": все-таки Dreamweaver не может учесть многие тонкости конкретной программы просмотра Web-страниц. И такая возможность предусмотрена: не закрывая окна документа, мы можем вызвать наш любимый Web-обозреватель и оценить окончательный вид своего творения, так сказать, "в родной обстановке". Для этого достаточно выбрать пункт **iexplore** подменю **Preview in Browser** меню **File** или, что проще и быстрее, нажать клавишу <F12>.

Вызов справки

Данная книга описывает далеко не все возможности Dreamweaver. Поэтому при работе с этой программой нам может понадобиться — и наверняка понадобится — помощь. Как и все серьезные Windows-приложения, Dreamweaver снабжен мощной *справочной системой*. Для ее вызова нужно просто нажать клавишу <F1> или выбрать пункт **Using Dreamweaver** в меню **Help**. После этого на экране появится *окно справки*, показанное на рис. 3.34.

В левой части окна справки, на вкладке **Содержание**, расположен иерархический список тем. Содержимое самих статей отображается справа. Чтобы прочитать статью, достаточно развернуть нужное "дерево" списка и щелкнуть по соответствующей "ветви".

Справочная система Dreamweaver также содержит предметный указатель. Чтобы воспользоваться им, переключимся на вкладку **Указатель**, выберем в списке терминов нужный и дважды щелкнем по нему. Соответствующая статья справки отобразится справа. Чтобы быстро найти нужный термин, мы также можем ввести первые несколько символов его названия в поле ввода, расположенном над списком, и первый подходящий термин будет выделен в списке.

Рис. 3.34. Окно справки Dreamweaver

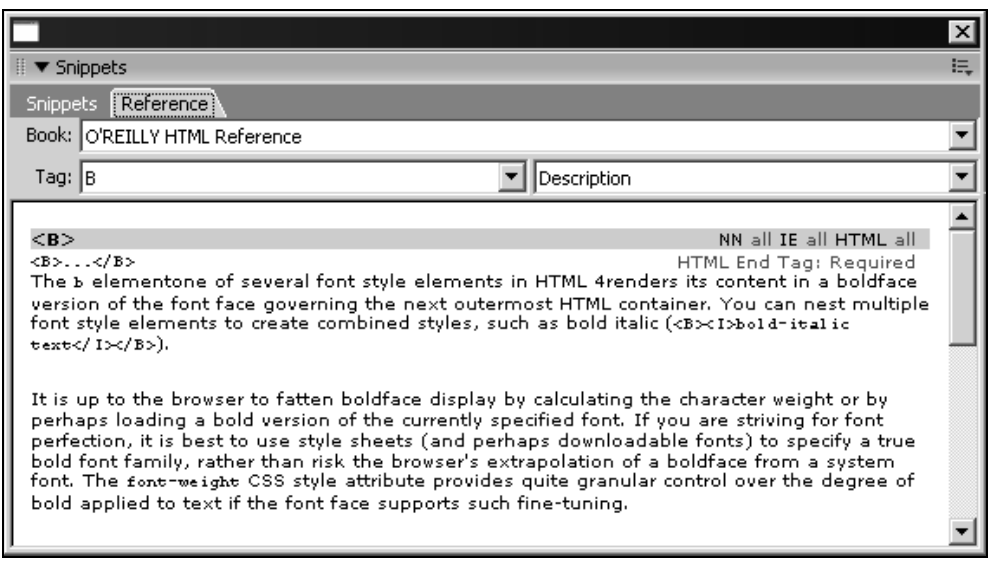


Рис. 3.35. Панель **Reference**

В составе Dreamweaver также поставляется полное руководство по HTML и CSS от издательства O'Reilly. Их содержимое отображается в особой панели **Reference** (рис. 3.35). Чтобы вывести ее на экран, нужно включить пункт-выключатель **Reference** меню **Window** или нажать комбинацию клавиш <Shift>+<F1>.

Чтобы просмотреть справочник по HTML, достаточно выбрать пункт **O'REILLY HTML Reference** в раскрывающемся списке **Book**. После этого в раскрывающемся списке **Tag** выбирается название нужного тега — в панели появится его описание.

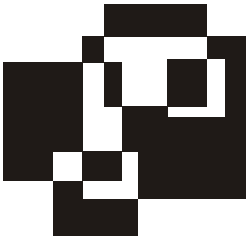
Справа от раскрывающегося списка **Tag** находится другой раскрывающийся список. В данный момент (см. рис. 3.35) там выбран пункт **Description** — это знак того, что в панели **Reference** отображаются сведения о самом теге. Если нужно получить сведения о каком-либо атрибуте выбранного в списке **Tag** тега, достаточно просто выбрать этот атрибут в правом раскрывающемся списке.

Справочник по CSS вызывается выбором пункта **O'REILLY CSS Reference** раскрывающегося списка **Book**. Далее в раскрывающемся списке **Style** выбирается название нужного атрибута — и в панели **Reference** появятся сведения о нем.

Что дальше?

Что ж, для начала хватит. Конечно, потом мы продолжим пользоваться Dreamweaver, а значит, будем изучать другие его возможности. Но это будет потом, в следующих главах.

Поскольку наш сайт готов, его пора публиковать на Web-сервере. Процесс публикации будет рассмотрен в следующей главе.



Глава 4

Работа с Web-сайтом в Dreamweaver

Ну вот, наш сайт готов. Мы создали все Web-страницы, связали их гиперссылками, даже оформили их с помощью стилей CSS. Настала пора опубликовать наш сайт на Web-сервере. Этим мы сейчас и займемся.

Да, но где можно найти приличный Web-сервер? Давайте решим для себя этот вопрос.

- Можно зарегистрироваться на одном из Web-серверов, предоставляющих место для сайтов их клиентов бесплатно. Таких серверов сейчас много, и выбрать подходящий нам несложно. Конечно, грандиозный Web-портал на них не опубликуешь, но для нашего небольшого архива файлов и статей такой вариант — более чем предпочтительный.
- Мы можем установить программу Web-сервера на своем собственном компьютере. Это делается быстро, минут за десять, да и то, если не очень торопиться. В *приложении 1* этой книги приведены инструкции по установке популярного бесплатного Web-сервера Apache. (Разумеется, можно использовать и другой Web-сервер, например, Microsoft Personal Web Server или Internet Information Server, если, конечно, имеются навыки работы с ним.)

Вариант с платным хостингом мы рассматривать не будем — сейчас незачем тратить на это деньги. Также не будем рассматривать вариант публикации на Web-сервере какой-либо организации — если там есть знакомый администратор, то пожалуйста. Так что у нас — всего два варианта, перечисленных ранее.

Однако давайте все-таки установим на свой компьютер Web-сервер — он очень пригодится нам в дальнейшем. В процессе создания сайта нам придется постоянно его тестировать, выявлять ошибки, исправлять их и снова тестировать. А такие вещи лучше делать на своем собственном сервере: не нужно устанавливать каждую минуту соединение с Интернетом, да и возможностей по настройке он предоставляет больше. Так что читаем *приложение 1* и выполняем все приведенные там рекомендации, пока Web-сервер не заработает.

Но как выполняется сам процесс публикации сайта? Сейчас мы об этом поговорим.

Подготовка к публикации сайта

Тут возникает небольшая проблема. Дело в том, что созданные нами ранее Web-страницы Dreamweaver не рассматривает как Web-сайт. Для него это всего лишь набор разрозненных страничек, не объединенных друг с другом ничем, кроме гиперссылок. Фактически Dreamweaver даже не в курсе, что мы сделали сайт!

Так что первое, что мы должны сделать, — это выполнить регистрацию нашего сайта в Dreamweaver. Поскольку мы сохраняли все Web-страницы в одной папке — Site1 — этот процесс не займет много времени.

Регистрация сайта в Dreamweaver

Регистрация сайта в Dreamweaver начинается выбором пункта **Manage Sites** меню **Site**. На экране появится диалоговое окно **Manage Sites**, показанное на рис. 4.1. Большую часть этого окна занимает список уже созданных сайтов (среди которых есть сайт автора этой книги — Vlad's site). Чтобы добавить сюда и наш сайт, нажмем кнопку **New** и в появившемся на экране небольшом меню выберем пункт **Site**.

Рис. 4.1. Диалоговое окно **Manage Sites**

После этого на экране появится диалоговое окно **Site Definition**, состоящее из двух вкладок. Если оно открыто на вкладке **Basic**, переключимся на вкладку **Advanced** — она предоставляет больше возможностей по настройке

нашего сайта. После этого проверим, выбран ли в списке **Category** пункт **Local Info**. После этого окно **Site Definition** должно иметь такой вид — рис. 4.2.

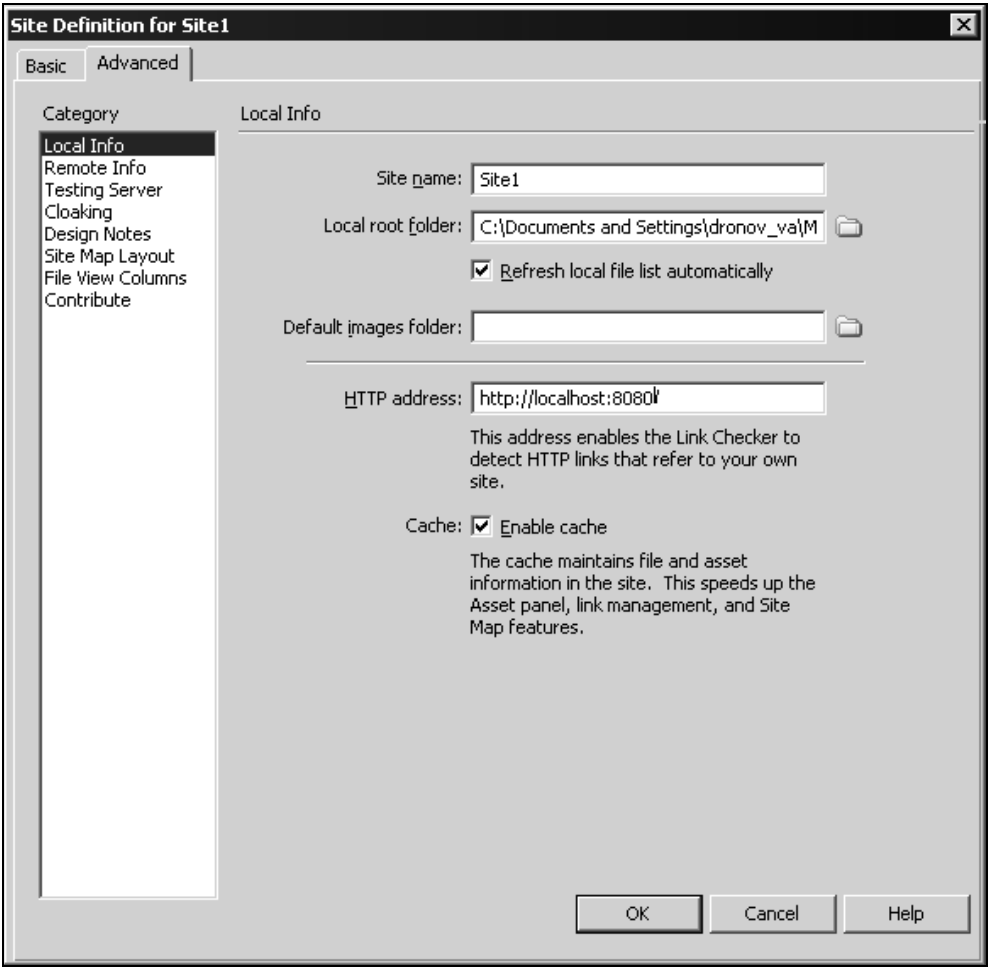


Рис. 4.2. Диалоговое окно **Site Definition** (категория **Local Info**)

Элементы управления категории **Local Info** позволяют задать сведения о тех файлах нашего сайта, с которыми в данный момент идет работа. Назовем их *локальной копией* сайта. Напротив, файлы, которые уже опубликованы на Web-сервере (неважно, локальном, то есть установленном на нашем компьютере, или удаленном), пусть называются *удаленной копией* сайта.

В поле ввода **Site name** вводится имя сайта, которое будет отображаться в списке сайтов окна **Manage Sites**. Введем туда `Site1`, как показано на рис. 4.2.

В поле ввода **Local root folder** указывается путь к корневой папке локальной копии сайта. Проще всего задать его, щелкнув по значку папки, расположенному справа от этого поля ввода, и выбрав нужную папку в появившемся на экране стандартном диалоговом окне Windows.

В поле ввода **Default images folder** вводится имя папки, в которую по умолчанию будут помещаться все графические изображения, помещенные нами на Web-страницах. Поскольку у нас единственное изображение Arrow.gif хранится прямо в корневой папке, ничего не будем туда вводить.

В поле ввода **HTTP address** вводится интернет-адрес нашего сайта. В принципе, вводить его необязательно, но тогда Dreamweaver не сможет проверить на правильность гиперссылки. Давайте введем туда особый интернет-адрес `http://localhost:8080/`, обозначающий наш собственный компьютер (*локальный хост*).

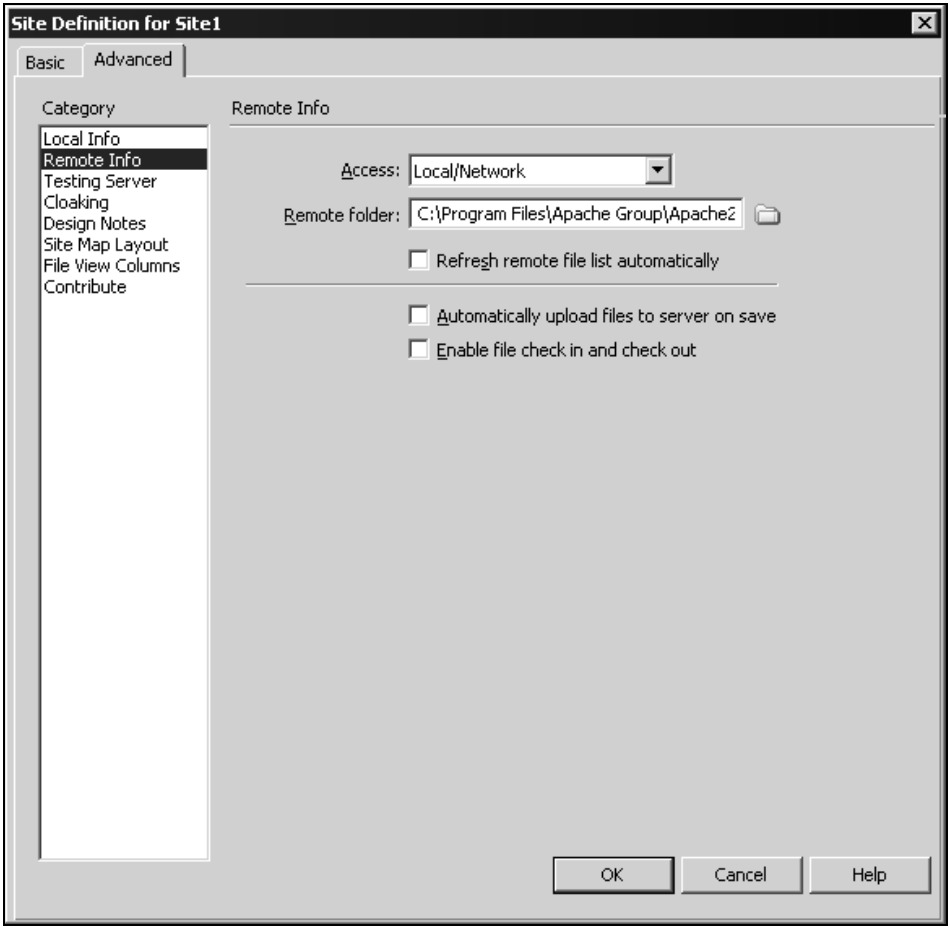


Рис. 4.3. Диалоговое окно **Site Definition** (категория **Remote Info**)

Внимание

Здесь предполагается, что Web-сервер настроен на порт 8080. Если же он использует другой порт, его нужно будет подставить в интернет-адрес вместо 8080.

Теперь выберем в списке **Category** пункт **Remote Info**, где задаются параметры удаленной копии сайта. Окно **Site Definition** примет такой вид — рис. 4.3.

Прежде всего, в раскрывающемся списке **Access** выберем пункт **Local/Network**. Этим мы сообщим Dreamweaver, что Web-сервер, на котором должна быть опубликована удаленная копия сайта, находится на том же компьютере, что и его локальная копия.

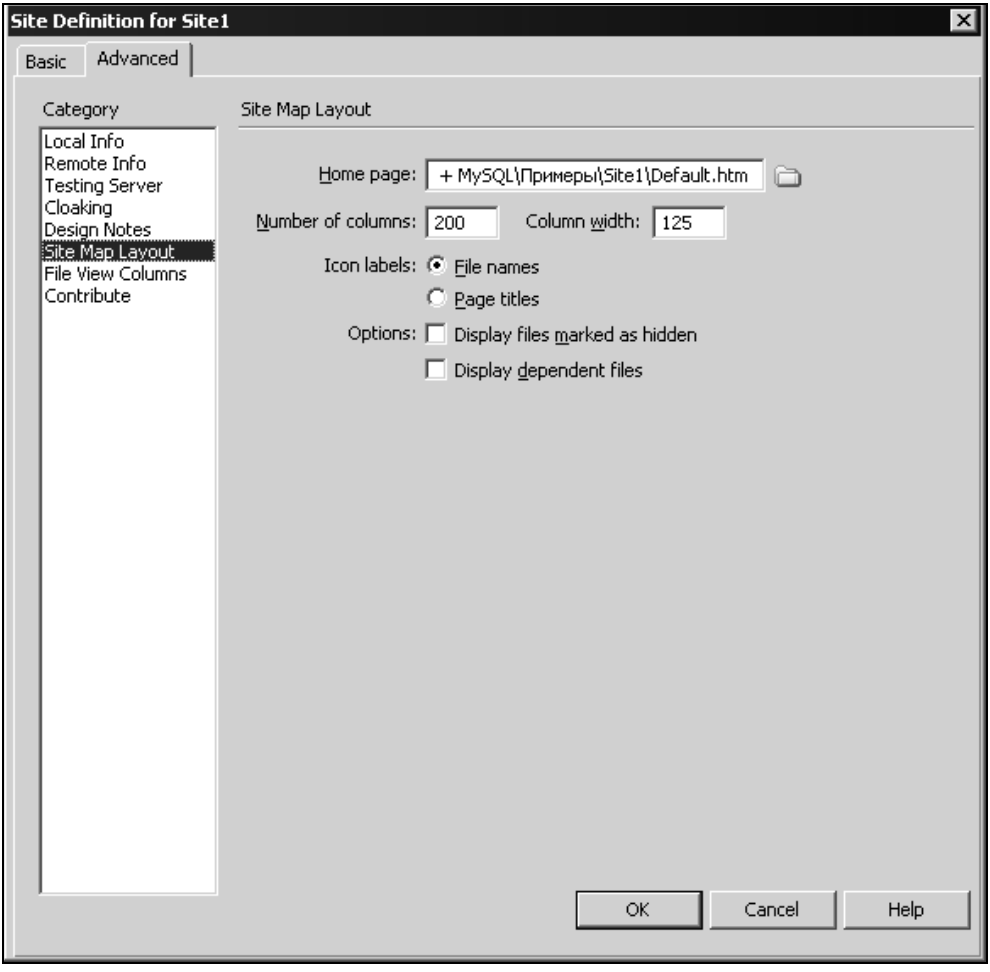


Рис. 4.4. Диалоговое окно **Site Definition** (категория **Site Map Layout**)

Сразу после этого ниже списка **Access** появится поле ввода **Remote folder**. В нем вводится путь к корневой папке удаленной копии сайта. Проще всего, конечно, щелкнуть значок папки, расположенный справа от поля ввода, и выбрать нужную папку в появившемся на экране стандартном диалоговом окне Windows. В случае Web-сервера Apache корневая папка по умолчанию — это <*папка, в которой установлен Apache*>\htdocs.

Остается только выбрать в списке **Category** пункт **Site Map Layout** (рис. 4.4), кое-что проверить и задать некоторые дополнительные настройки.

Прежде всего, проверим, подставил ли Dreamweaver в поле ввода **Home page** имя файла Web-страницы по умолчанию. Если нет, введем его или щелкнем по значку папки справа от поля ввода и выберем нужный файл в появившемся на экране диалоговом окне **Choose Home Page**, похожем на окно **Select File**.

Закончив ввод параметров вашего сайта, нажмем кнопку **OK** диалогового окна **Site Definition**. После этого Dreamweaver выполнит сканирование списка файлов и создаст в памяти их список. Далее нажмем кнопку **Done** диалогового окна **Manage Sites**. Регистрация сайта закончена.

Работа с файлами сайта. Панель *Files*

А теперь мы познакомимся с одним очень полезным и удобным инструментом Dreamweaver для работы с файлами, составляющими Web-сайт. Это панель **Files**. Чтобы вывести ее на экран, нужно включить пункт-выключатель **Files** в меню **Window** или нажать клавишу <F8>. Сама эта панель показана на рис. 4.5.

Большую часть панели **Files** занимает иерархический список файлов и папок, составляющих сайт, с "корнем" в корневой папке сайта (так себе каламбур). Также панель **Files** имеет свой собственный небольшой инструментарий с элементами управления, предоставляющими быстрый доступ к наиболее часто употребляемым командам, и строку статуса.

Список файлов и папок панели **Files** похож на аналогичный список Проводника: его можно сворачивать и разворачивать, а также выделять нужный файл или папку щелчком мыши. Если нужно выделить несколько файлов, достаточно щелкнуть по ним, удерживая нажатой клавишу <Ctrl>. Чтобы выбрать все файлы списка, можно нажать комбинацию клавиш <Ctrl>+<A> или щелкнуть небольшую иконку , расположенную в верхнем правом углу панели, и выбрать в подменю **Edit** появившегося на экране *дополнительного меню* пункт **Select All**.

Раскрывающийся список, расположенный слева в инструментарии панели **Files**, позволяет быстро выбрать любой из зарегистрированных в Dreamweaver сайтов для отображения в панели. Это значит, что мы можем зарегистрировать в Dreamweaver сколько угодно сайтов и с легкостью переходить от одного к другому. (Но это, конечно, в будущем.)

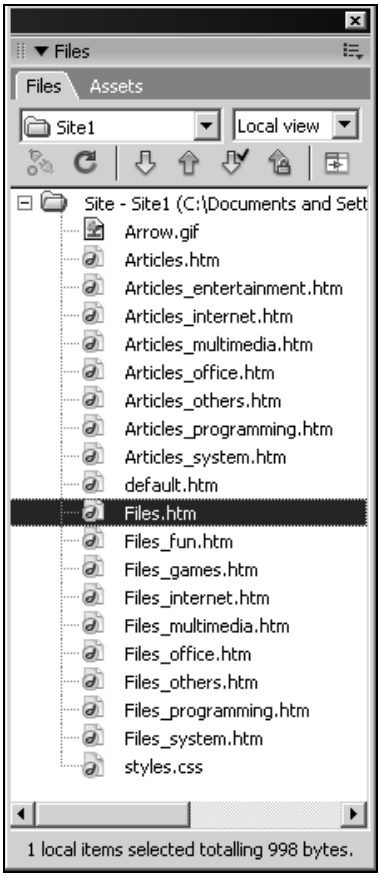


Рис. 4.5. Панель **Files**

А тот раскрывающийся список, что находится в инструментарии панели правее, позволяет задать копию сайта — локальную или удаленную, — содержимое которой нужно просмотреть в панели **Files**. Пункт **Local view**, выбранный по умолчанию, задает отображение списка файлов локальной копии, а пункт **Remote view** — файлов удаленной копии.

Ранее говорилось, что список файлов панели **Files** похож на аналогичный список Проводника Windows за тем исключением, что в нем отображаются и папки, и файлы. И так же, как Проводник, панель **Files** предоставляет возможности по управлению этими файлами и папками.

Так, чтобы удалить выделенный в списке файл или папку, нужно выбрать пункт **Delete** подменю **File** дополнительного меню или нажать клавишу <Del>. После этого Dreamweaver спросит, действительно ли мы хотим удалить этот файл (папку); после этого нужно нажать **Yes** или **No**, соответственно, для удаления или отказа от него.

Внимание

Dreamweaver выполняет удаление файлов и папок с файлами в корзину Windows. Однако пустые папки удаляются бесследно.

Чтобы переименовать выделенный файл, нужно щелкнуть по нему еще раз мышью или нажать клавишу <F2>. Также можно выбрать пункт **Rename** подменю **File** дополнительного меню. После этого на месте имени данного файла появится небольшое поле ввода, где будет подставлено его старое имя. Отредактируем его или введем новое, после чего нажмем клавишу <Enter> для сохранения нового имени или клавишу <Esc> для отмены.

Если на других Web-страницах присутствуют гиперссылки на переименованный файл, Dreamweaver предложит нам исправить их. На экране появится диалоговое окно **Update Files**, показанное на рис. 4.6. В списке **Update links in the following files?** находятся Web-страницы, на которых были найдены гиперссылки, указывающие на переименованный файл. Нажмем кнопку **Update**, чтобы обновить их, или **Don't Update**, чтобы отказаться от обновления; в последнем случае нам придется сделать это вручную.

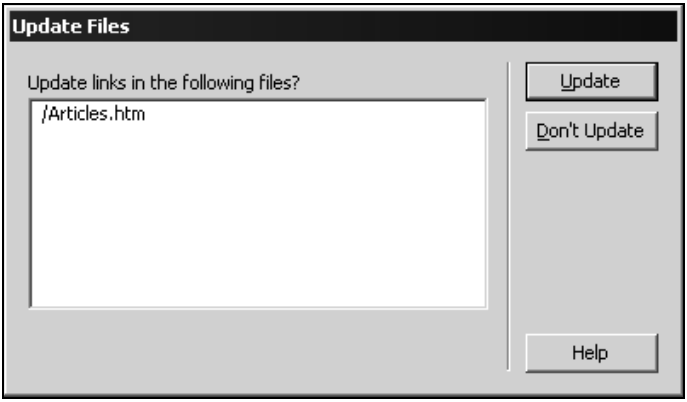


Рис. 4.6. Диалоговое окно **Update Files**

Если дважды щелкнуть по какому-либо файлу или выделить его и нажать клавишу <Enter>, Dreamweaver откроет его. Такой номер проходит с Web-страницами и таблицами стилей, а если у нас на компьютере установлена программа Web-графики Macromedia Fireworks, то и с графическими файлами.

С помощью панели **Files** можно также создать новую Web-страницу, выбрав пункт **New File** подменю **File** дополнительного меню или нажав комбинацию клавиш <Ctrl>+<Shift>+<N>. После этого в самом низу списка появится новый файл с именем untitled.htm, которое мы сразу же можем (и должны, собственно) изменить. Изменив его, нажимаем клавишу <Enter>, Dreamweaver открывает новую пустую страницу, и мы можем вводить ее содержимое.

Новая папка создается аналогично, только нужно выбрать пункт **New Folder** или нажать комбинацию клавиш <Ctrl>+<Shift>+<Alt>+<N>. После этого в самом низу списка появится новая папка с именем untitled, которое мы должны будем изменить и нажать клавишу <Enter>. Потом мы можем перетащить мышью нужные файлы (и другие папки) в эту папку, а Dreamweaver предложит изменить гиперссылки, указывающие на них, выведя уже знакомое нам диалоговое окно **Update Files** (см. рис. 4.6). Если же во время перетаскивания файлов удерживать нажатой клавишу <Ctrl>, Dreamweaver вместо переноса скопирует эти файлы.

И еще. Предположим, что при регистрации своего сайта в Dreamweaver мы ошиблись в задании файла страницы по умолчанию (или если ошибся Dreamweaver — ведь он пытается автоматически определить ее). Чтобы исправить эту ошибку, выделим Web-страницу, которая должна быть страницей по умолчанию, и выберем пункт **Set as Home Page** (сделать страницей по умолчанию) подменю **Site** дополнительного меню. Быстро и просто!

Проверка Web-страниц

Ну, теперь-то уж можно публиковать сайт? Нет, еще рано. Терпение — осталось совсем чуть-чуть! Вот проверим корректность HTML-кода и гиперссылок — и опубликуем наше творение в Сети.

Проверка правильности HTML-кода

В процессе создания Web-страницы (особенно если непосредственно править код HTML или переделывать чьи-то старые Web-страницы) немудрено допустить ошибки. Такими ошибками могут быть, например, неправильный интернет-адрес гиперссылки или отсутствие названия страницы (тега `<TITLE>`). И если второе еще более-менее терпимо, то первое совершенно недопустимо.

Итак, проверим правильность HTML-кода. Закроем все открытые окна документов, чтобы они нам не мешали. Выберем пункт **Reports** в меню **Site** главного окна программы или одноименный пункт в подменю **Site** дополнительного меню панели **Files**. На экране появится диалоговое окно **Reports**, показанное на рис. 4.7.

Большую часть этого окна занимает состоящий из двух "ветвей" иерархический список **Select reports**, который позволяет выбрать данные, включаемые в отчет. Для нас представляет интерес вторая "ветвь" — **HTML Reports** — содержащая следующие пункты:

- **Combinable Nested Font Tags** включает поиск вложенных тегов физического форматирования `<FONT>`, которые могут быть безболезненно объединены;
- **Accessibility** включает проверку на корректность задания различных элементов страницы (для всех ли страниц задана кодировка и т. п.);

- **Missing Alt Text** включает поиск незаполненных атрибутов `ALT` тегов `<IMG>` (альтернативный текст);
- **Redundant Nested Tags** включает поиск ненужных вложенных тегов;
- **Removable Empty Tags** включает поиск пустых тегов, которые могут быть безболезненно удалены;
- **Untitled Documents** включает поиск Web-страниц без названия (без тега `<TITLE>`).

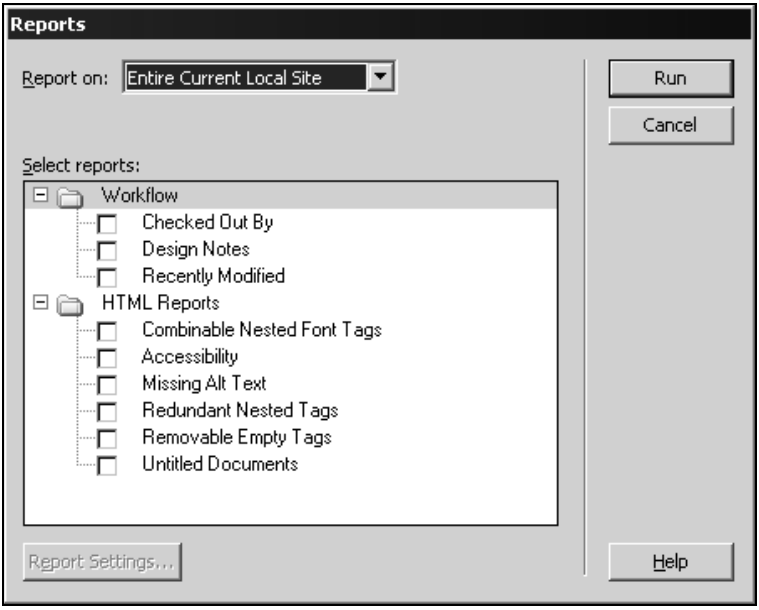


Рис. 4.7. Диалоговое окно **Reports**

Каждый из этих пунктов представляет собой флажок, который мы можем установить или сбросить. Давайте установим их все.

Далее проверим, чтобы в раскрывающемся списке **Report on** был выбран пункт **Entire Current Local Site**, предписывающий Dreamweaver проверить все страницы сайта. (Выбор пункта **Current Document** запускает проверку Web-страницы, открытой в активном окне документа.) После этого нажмем кнопку **Run**. Через некоторое время Dreamweaver выведет панель **Site Reports**, показанную на рис. 4.8.

В списке, находящемся в этой панели, перечислены все найденные ошибки и недостатки в нашем HTML-коде. Дважды щелкнув по любому пункту этого списка, мы откроем файл, в котором найдена ошибка; при этом окно документа откроется в режиме показа HTML-кода, а ошибочный фрагмент будет выделен. Чтобы получить более подробные сведения о найденной

ошибке, нужно нажать кнопку **More Info** (с изображением восклицательного знака), после чего откроется окно справки Dreamweaver с нужными сведениями.

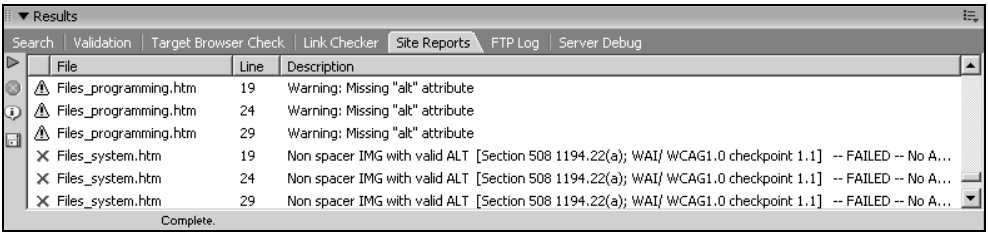


Рис. 4.8. Панель **Site Reports**

В принципе, то, что нашел в нашем коде Dreamweaver, не очень страшные ошибки — ни одна программа Web-обозревателей не обратит на них внимания. Но если там встретится что-то действительно серьезное, лучше это исправить.

Поскольку панель **Site Reports** нам после этого больше не понадобится, ее можно закрыть. Для этого достаточно выбрать пункт **Close Panel Group** в ее дополнительном меню.

Проверка гиперссылок

Давайте предположим, что мы переименовали какую-то страницу нашего сайта и не исправили указывающие на нее гиперссылки, а Dreamweaver не сделал это за нас. Такие гиперссылки, указывающие на несуществующий файл, называются *"оборванными"* или *"мертвыми"*.

Теперь, если посетитель нашего сайта щелкнет по гиперссылке, Web-сервер не сможет найти отсутствующую страницу и отправит Web-обозревателю сообщение об ошибке с кодом 404. Это и есть печально известная *ошибка 404*, с которой мы так часто сталкиваемся, когда путешествуем по Сети.

Конечно, Dreamweaver делает все, чтобы не допустить появления "оборванных" ссылок. Но, как говорится, и на старуху бывает проруха. Поэтому перед публикацией сайта стоит выполнить проверку корректности гиперссылок.

Проверить корректность гиперссылок можно как на отдельной странице, так и на всем сайте. Чтобы проверить гиперссылки на отдельной странице сайта, сначала откроем ее. После этого выберем пункт **Check Links** подменю **Check Page** меню **File** главного окна, одноименный пункт подменю **File** дополнительного меню панели **Files** или нажмем комбинацию клавиш <Shift>+<F8>. Если же мы хотим проверить ссылки на всех страницах сайта, нужно будет использовать пункт **Check Links Sitewide** подменю **Site** дополнительного меню панели **Files** или комбинацию клавиш <Ctrl>+<F8>.

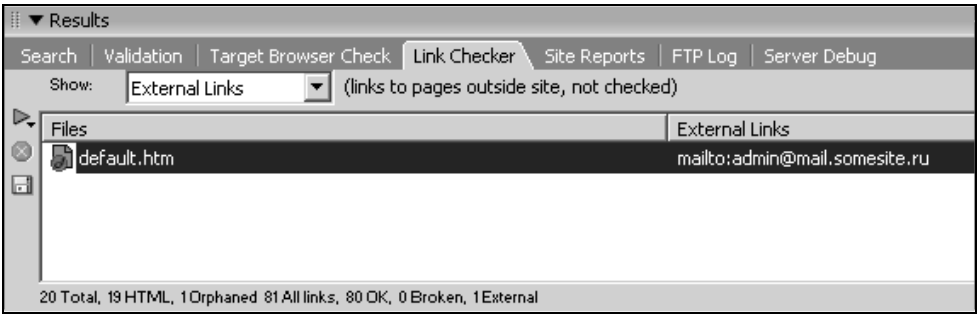


Рис. 4.9. Панель **Link Checker**

В любом случае после этого на экране появится панель **Link Checker**, показанная на рис. 4.9.

В верхней части этой панели находится раскрывающийся список **Show**, где можно выбрать, что будет показано в списке, занимающем панель. В этом списке доступны три пункта:

- **Broken Links** — "оборванные" гиперссылки;
- **External Links** — *"внешние"* гиперссылки, то есть ссылки, указывающие на другие сайты;
- **Orphaned Files** — *файлы-"сироты"*, т. е. файлы, на которые не указывает ни одна гиперссылка.

Бóльшую же часть панели **Files** занимает список найденных ошибочных ссылок. Каждый пункт этого списка содержит адрес Web-страницы, на которой была найдена гиперссылка, и интернет-адрес этой ссылки. Щелкнув по любой позиции данного списка, мы откроем соответствующую Web-страницу в окне документа, причем Dreamweaver сам выделит ошибочную ссылку.

Давайте посмотрим, все ли у нас в порядке с гиперссылками. Так, оборванных ссылок нет, а внешняя ссылка всего одна — адрес электронной почты. Файлов-"сирот" также нет. Значит, все нормально.

Взаимодействие панели *Files* и окна документа

Но предположим, что мы все-таки допустили ошибки в гиперссылках. Кроме того, мы забыли создать на страницах списков файлов и статей гиперссылки, указывающие на страницы списков соответствующих категорий. А это непорядок — посетитель нашего сайта должен иметь возможность без проблем путешествовать со страницы на страницу. Давайте же исправим ошибки!

Нам известны целых два способа создания гиперссылок, и оба они очень просты. Но Dreamweaver предоставляет нам еще два способа связать стра-

ницы нашего сайта воедино, доступный только в том случае, если мы зарегистрировали наш сайт (о регистрации сайта в Dreamweaver см. начало этой главы). Сейчас мы их рассмотрим.

Сначала проверим, открыта ли у нас панель **Files**; если же нет, включим пункт-выключатель **Files** в меню **Window** или нажмем клавишу <F8>. Далее откроем страницу Articles_internet.htm, дважды щелкнув по названию файла в списке файлов локальной копии сайта панели **Files**. После этого поставим текстовый курсор под таблицей списка статей и наберем текст `На список категорий` и выделим его. Именно этот текст мы и превратим в гиперссылку, указывающую на страницу Articles.htm.

Теперь немного облегчим себе жизнь. Как правило, Dreamweaver при первом запуске открывает уйму панелей, полностью загромождая правую часть своего главного окна. Чтобы освободить его, давайте закроем все панели, кроме **Files** и редактора свойств, для чего достаточно выбрать пункт **Close Panel Group** дополнительного меню. Далее поместим курсор мыши на левый край синего заголовка окна панели **Files**, туда, где находится своего рода "ручка", нажмем левую кнопку мыши и перетащим панель на середину экрана. Так мы освободим правую часть редактора свойств.

Да, но что такое интересное там находится? Давайте посмотрим на редактор свойств, а именно на поле ввода **Link**, где указан интернет-адрес гиперссылки. Справа от него находится небольшой значок . Так вот, чтобы создать гиперссылку, достаточно перетащить его прямо на требуемый файл в списке панели **Files**. Так, если мы перетащим этот значок на файл Articles.htm в списке файлов и "бросим" его, выделенный нами текст `На список категорий` превратится в гиперссылку. Это и есть первый способ создать гиперссылку с помощью панели **Files**.

Другой способ также прост. Для создания гиперссылки мы можем перетащить нужный файл из списка панели **Files** прямо в поле ввода **Link** редактора свойств. Когда мы будем создавать гиперссылки, указывающие на страницу списка категорий, на других страницах, то можем опробовать и этот способ.

Закончили? Вот теперь можно публиковать сайт на нашем собственном Web-сервере. А в перспективе — и в Сети.

Публикация сайта

Итак, настал тот желанный миг, когда наш сайт совсем готов! Мы создали все страницы, связали их друг с другом, исправили ошибки в HTML-коде и даже проверили на корректность интернет-адреса гиперссылок. Фактически мы сделали все, что очень часто забывают сделать даже довольно опытные Web-дизайнеры. Теперь наш сайт можно публиковать на Web-сервере.

Сначала мы рассмотрим, каким образом сайт публикуется на локальном Web-сервере, том самом, что мы установили на своем компьютере в начале этой главы. А уж потом, тщательно все проверив, мы перейдем к публикации нашего сайта в Сети.

Публикация сайта на локальном Web-сервере

Простейший способ опубликовать наш сайт — это выделить корневую папку сайта ("корень" иерархического списка) в списке файлов локальной копии в панели **Files** и нажать кнопку **Put File(s)** инструментария панели (). Данная кнопка запускает процесс копирования файлов в корневую папку Web-сервера. Также мы можем выбрать пункт **Put** подменю **Site** дополнительного меню панели или нажать комбинацию клавиш <Ctrl>+<Shift>+<U>.

Так как мы выбрали корневую папку, то есть, фактически, весь сайт, Dreamweaver переспросит нас, действительно ли мы хотим скопировать весь сайт на сервер. В ответ нажмем кнопку **OK**. Мгновение — и наш сайт опубликован на сервере.

Давайте проверим, правильно ли все Dreamweaver сделал. Выберем в правом раскрывающемся списке инструментария панели **Files** пункт **Remote View**. Список этой панели тотчас покажет все файлы удаленной копии сайта. Чтобы продолжить работу с локальной копией, достаточно выбрать в этом же списке пункт **Local View**.

А теперь можно запустить Web-обозреватель, набрать в строке адреса **http://localhost/** и посмотреть на наш сайт воочию.

Внимание

Если наш Web-сервер настроен на порт, отличный от 80-го, нам будет нужно набрать в строке адреса **http://localhost:<*номер порта*>/**.

Dreamweaver также позволяет нам опубликовать на сервере отдельные файлы сайта. Для этого достаточно выделить нужные файлы в списке панели **Files** и нажать все ту же кнопку **Put file(s)**. Если у нас какие-то Web-страницы открыты, изменены и не сохранены, Dreamweaver предложит сохранить их. В этом случае нужно нажать кнопку **Yes** для сохранения соответствующего файла, кнопку **No** — для отказа от сохранения или кнопку **Cancel** — для отказа от его публикации.

Также Dreamweaver может спросить нас, публиковать ли на сервере файлы, на которые ссылается данная страница, выведя на экран диалоговое окно **Dependent Files** (рис. 4.10). Если нажать кнопку **Yes**, эти файлы будут опубликованы, если **No** — будут опубликованы только те файлы, которые мы выделили в списке панели **Files**, если **Cancel** — публикация файлов на сервере будет прервана.

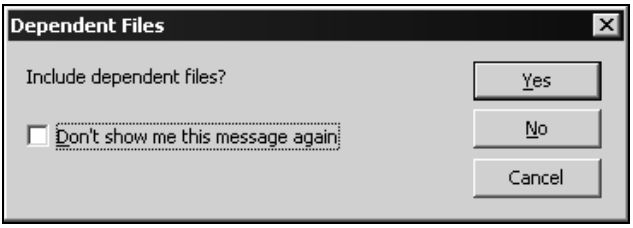


Рис. 4.10. Диалоговое окно **Dependent Files**

Да, все это просто. Если мы точно знаем, какие файлы хотим опубликовать. Но что делать, если мы забыли, какие файлы изменяли (обычная ситуация)? Публиковать заново весь сайт? Есть способ лучше.

Дело в том, что операционная система Windows (как и многие другие операционные системы) хранит дату и время последнего изменения каждого файла. Сравнивая две даты, можно выяснить, какой файл новее. И если файл, принадлежащий локальной копии сайта, имеет более позднюю дату последнего изменения, чем тот же файл удаленной копии, это значит, что он был изменен в Dreamweaver, но еще не опубликован. И, следовательно, его нужно скопировать на сервер, чтобы поддержать актуальность удаленной копии сайта.

Именно на таком принципе основан механизм *синхронизации* копий сайта. Dreamweaver проверяет даты разных копий файлов и принимает решение, какие из них нужно скопировать на сервер. Просто и надежно, если, конечно, встроенные часы и календарь нашего компьютера установлены правильно.

Чтобы запустить синхронизацию файлов, выберем пункт **Synchronize** подменю **Site** дополнительного меню панели **Files**. На экране появится диалоговое окно **Synchronize Files**, показанное на рис. 4.11.

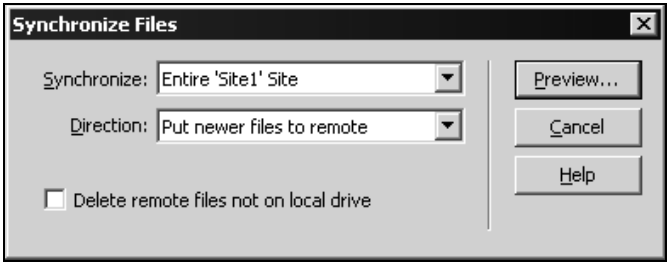


Рис. 4.11. Диалоговое окно **Synchronize Files**

Раскрывающийся список **Synchronize** позволяет задать, какие файлы мы хотим синхронизировать. Пункт **Selected Local File Only** позволяет синхронизировать только выделенные в списке файлы. А пункт **Entire <*название нашего сайта*> Site** позволяет синхронизировать весь сайт целиком, так что давайте выберем именно его.

В раскрывающемся списке **Direction** нам будет нужно выбрать пункт **Put newer files to remote**, чтобы Dreamweaver скопировал новые файлы на сервер, заменив ими устаревшие файлы удаленной копии. Если мы удалили какие-то файлы, то должны также будем включить флажок **Delete remote files not on local drive**.

Сам процесс синхронизации запускается нажатием кнопки **Preview**. После этого на экране появится окно списка синхронизируемых файлов, показанное на рис. 4.12. Здесь мы сможем дополнительно указать, какие файлы необходимо синхронизировать, а какие — нет.

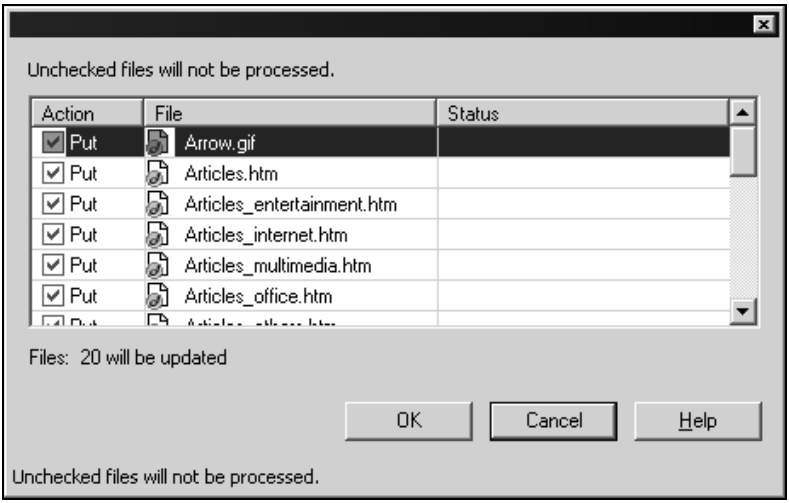


Рис. 4.12. Окно списка синхронизируемых файлов

Бóльшую часть этого окна занимает собственно список синхронизируемых файлов. Он организован в виде таблицы с тремя колонками, которые мы сейчас рассмотрим.

Колонка **Action** содержит название действия, которое будет применено к файлу. В нашем случае таких действий может быть два:

- **Put** — копирование на сервер,
- **Delete** — удаление.

Слева от названия действия находится флажок, включенный по умолчанию. Отключив его, вы сможете отменить действие над этим файлом (копирование или удаление соответственно).

Колонка **File** содержит имя файла, а колонка **Status** — текст, обозначающий состояние синхронизации и показываемый после ее завершения.

Синхронизация запускается нажатием кнопки **OK**. Если какие-либо файлы в результате синхронизации должны быть удалены, Dreamweaver предупре-

дит нас об этом; нажатие кнопки **OK** выполняет удаление этих файлов, а нажатие кнопки **Cancel** — отменяет.

По завершении синхронизации Dreamweaver выведет в то же самое окно результаты синхронизации, заполнив колонку **Status** (рис. 4.13). После этого остается закрыть это окно, нажав кнопку **Close**.

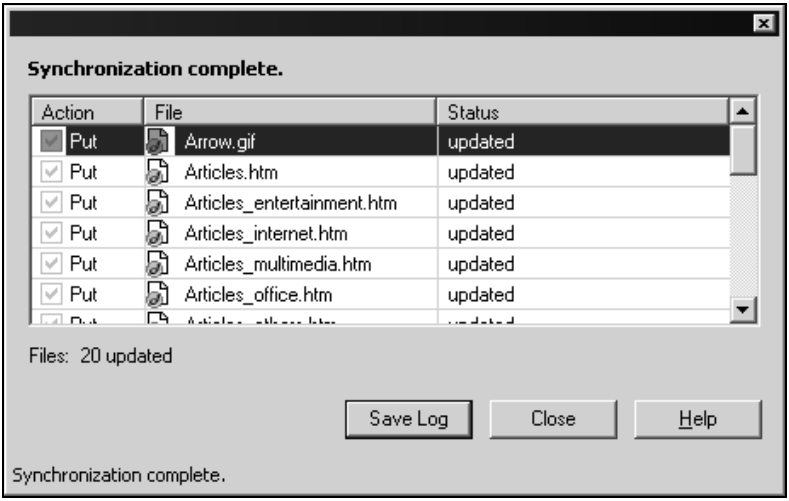


Рис. 4.13. Dreamweaver только что закончил синхронизацию

Замечание

Чтобы опубликовать наш сайт, можно также открыть Проводник Windows (или аналогичную программу управления файлами) и просто скопировать файлы локальной копии в корневую папку сервера. Но, разумеется, в Dreamweaver публикация сайта осуществляется проще и нагляднее.

Публикация сайта на удаленном Web-сервере

Разобравшись с публикацией сайта на локальном Web-сервере, давайте перейдем к процессу публикации на сервере удаленном, то есть в Интернете. Предположим, что мы все проверили и исправили все ошибки, так что наш сайт готов к запуску в эксплуатацию. Но как это сделать?

Использование для публикации Web-сайта протокола FTP

В настоящее время практически всегда для публикации сайта на удаленном сервере используется протокол FTP. В *главе 1* говорилось, что этот протокол может использоваться как для загрузки файлов с FTP-сервера FTP-клиентом, так и для записи их на сервер. Таким образом, мы можем запустить программу FTP-клиента (CuteFTP, FileZilla или любую другую) и просто пере-

писать файлы нашего сайта в корневую папку Web-сервера. Более того, нам для этого даже не понадобится FTP-клиент — сам Dreamweaver поддерживает публикацию сайтов на FTP-серверах.

Из этого следует, что на серверном компьютере, на котором работает программа бесплатного Web-сервера, также должен быть установлен и FTP-сервер (технически это сделать несложно). Обе этих программы-сервера настраиваются так, чтобы каждый их пользователь получил в свое распоряжение свою корневую папку (это также делается довольно просто). И, разумеется, на серверном компьютере должна работать программа, регистрирующая новых пользователей, заносящих сведения о них в списки пользователей Web- и FTP-сервера и создающая для каждого нового пользователя корневую папку.

Давайте подробно рассмотрим последовательность наших действий по публикации сайта на Web-сервере по FTP-протоколу.

1. Мы регистрируемся на бесплатном сервере, для чего задаем свое *имя пользователя* (или *логин* от английского login) и пароль. При этом имя и пароль заносятся в списки пользователей Web- и FTP-сервера, а на диске серверного компьютера автоматически создается корневая папка нашего сайта. Также автоматически создается интернет-адрес нашего сайта и сообщается нам.
2. Мы подключаемся к FTP-серверу с помощью программы FTP-клиента или прямо из Dreamweaver, введя при этом наше имя и пароль. FTP-сервер проверяет, есть ли в его списках такие имя и пароль, и, если есть, подключает нас прямо к корневой папке нашего сайта, пока пустой. Путь папки файловой системы серверного компьютера, соответствующей корневой папке нашего сайта, хранится также в списке пользователей FTP-сервера.
3. Мы копируем файлы сайта в нашу корневую папку.
4. Мы отключаемся от FTP-сервера. Это обязательно следует сделать, ведь программа FTP-сервера для поддержания соединения с клиентом использует память серверного компьютера, которая отнюдь не безгранична.
5. Мы запускаем Web-обозреватель, вводим полученный на шаге 1 интернет-адрес нашего сайта и просматриваем сайт.

В принципе, ничего сложного здесь нет — все задачи по настройке Web- и FTP-сервера берет на себя особая программа, работающая на серверном компьютере. Нам же остается только не забыть имя, пароль и интернет-адрес нашего сайта.

Замечание

Некоторые бесплатные Web-серверы позволяют пользователю загрузить файлы его сайта только через Web-обозреватель. Этот способ подходит только для начинающих пользователей, не умеющих обращаться с FTP-клиентом, и крайне неудобен.

Настройка Dreamweaver для публикации сайта по FTP

Конечно, для публикации сайта на Web-сервере по протоколу FTP можно использовать обычные программы FTP-клиентов. Но ведь мы работаем в Dreamweaver, который и сам это может сделать — его нужно просто об этом попросить. Так давайте же попросим!

Сначала нам будет нужно внести в настройки зарегистрированного сайта некоторые изменения. Выберем в меню **Site** главного окна пункт **Manage Sites**, в появившемся на экране диалоговом окне **Manage Sites** в списке сайтов выберем наш сайт и нажмем кнопку **Edit**. На экране появится диалоговое окно **Site Definition**; выберем пункт **Remote Info** в списке **Category**. И сразу же после этого в раскрывающемся списке **Access** выберем пункт **FTP**, давая понять Dreamweaver, что мы будем отправлять файлы на сервер по протоколу FTP (рис. 4.14).

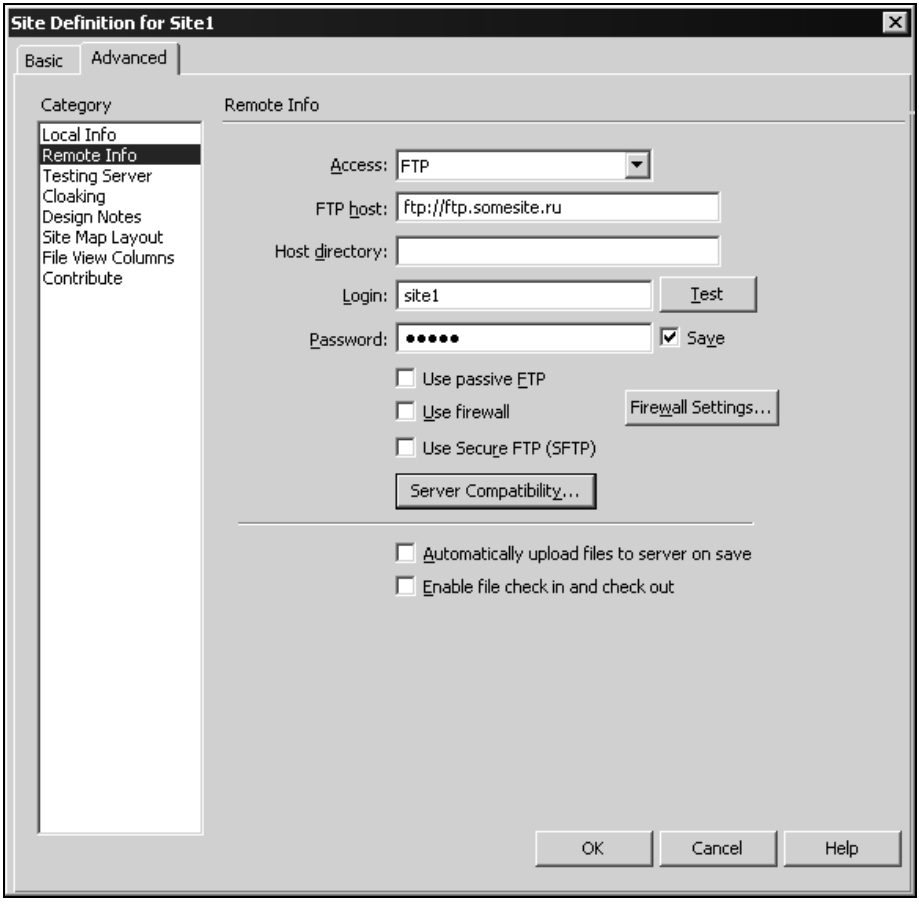


Рис. 4.14. Диалоговое окно **Site Definition** (категория **Remote Info**, выбран режим отправки файлов по протоколу FTP)

Далее нам нужно будет ввести следующие параметры в соответствующие поля ввода окна **Site Definition**:

- **FTP host** — адрес FTP-сервера, который "заведует" публикацией Web-сайтов;
- **Host directory** — путь корневой папки нашего сайта. Поскольку FTP-сервер, как правило, сразу подключает нас к этой папке, оставим это поле пустым;
- **Login** — имя пользователя, под которым мы зарегистрировались на сервере;
- **Password** — наш пароль (при его вводе в поле отображаются, в зависимости от установленной на нашем компьютере версии Windows, звездочки или точки).

По умолчанию Dreamweaver сохраняет введенный нами пароль и при подключении автоматически пересылает его FTP-серверу. Если же мы из соображений безопасности не хотим хранить этот пароль, а собираемся вводить его при каждом подключении, отключим флажок **Save**.

Нажав кнопку **Test**, мы можем проверить введенные нами параметры на правильность. Dreamweaver попытается подключиться к FTP-серверу и, в зависимости от успеха или неуспеха, выведет окно-предупреждение с соответствующим текстом.

Внимание

Если попытка подключения не удалась, но данные введены заведомо правильно, можно попробовать нажать кнопку **Server Compatibility**, чтобы вывести на экран одноименное диалоговое окно (рис. 4.15). В этом окне можно попробовать отключить флажок **Use FTP performance optimization** и включить флажок **Use alternative FTP move method**, не забыв после этого нажать кнопку **OK**.

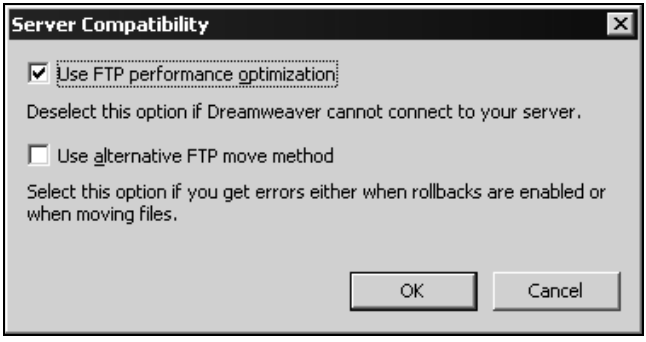


Рис. 4.15. Диалоговое окно **Server Compatibility**

Если FTP-сервер, с помощью которого мы собираемся публиковать сайт, поддерживает только *пассивный протокол FTP* (особая разновидность прото-

кола FTP, используемого из соображений безопасности), нам будет нужно включить флажок **Use passive FTP**. Вообще, лучше всегда его включать — хуже не будет.

Далее включим флажок **Use Secure FTP (SFTP)**, если для доступа на FTP-сервер используется защищенная версия протокола FTP — SFTP (Secure FTP, защищенный FTP).

Если же мы выходим в Сеть через *прокси-сервер* или *брандмауэр* (особые программы, защищающие локальную сеть организации от атак из Интернета), то нам будет нужно соответственно настроить Dreamweaver. Для этого включим флажок **Use firewall** и нажмем кнопку **Firewall Settings**. На экране появится уже знакомое нам по *главе 3* диалоговое окно **Preferences**, переключенное на категорию **Site** (рис. 4.16).

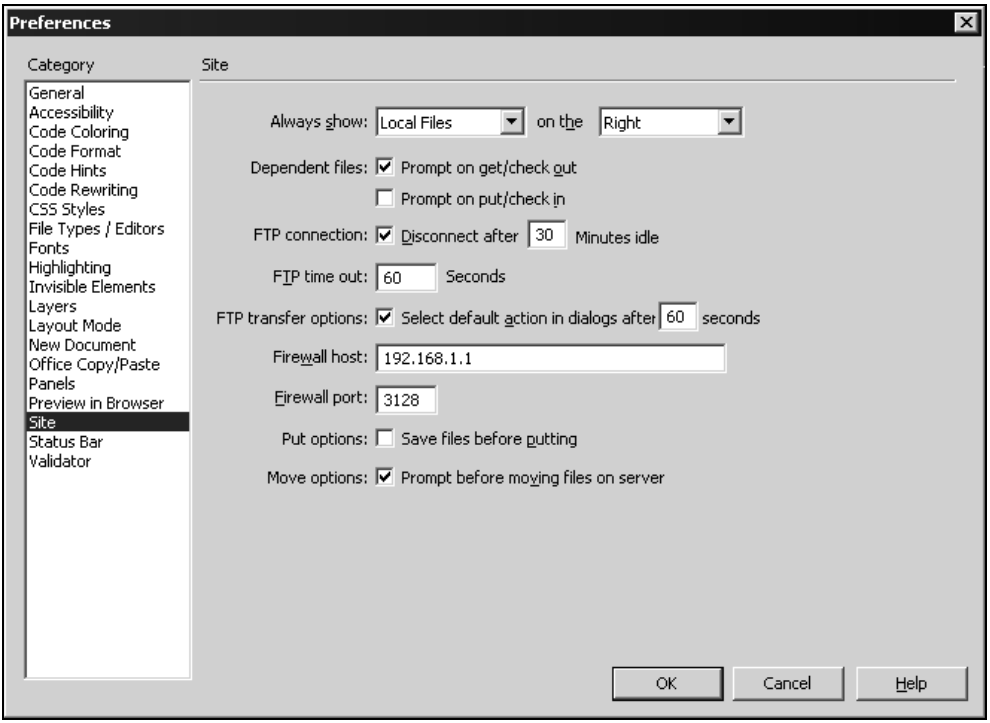


Рис. 4.16. Диалоговое окно **Preferences** (категория **Site**)

Здесь нас интересуют в настоящий момент только поля ввода **Firewall host** (адрес прокси-сервера) и **Firewall port** (порт прокси-сервера). Оба эти параметра можно выяснить у администратора прокси-сервера. Задав их, поочередно нажимаем кнопки **OK** диалоговых окон **Preferences** и **Site Definition** и кнопку **Done** окна **Manage Sites**. Все!

Замечание

Если выход в Сеть осуществляется через прокси-сервер или брандмауэр, то в диалоговом окне **Site Definition** также не помешает включить флажок **Use passive FTP**.

Публикация сайта по протоколу FTP

Итак, подготовительные действия закончены. Настал момент поместить наш Web-сайт во Всемирную Сеть.

Первое действие, которое нам необходимо выполнить, — это установить соединение с FTP-сервером. Для этого выведем на экран панель **Files** и нажмем кнопку **Connect to Remote Host** ( ), находящуюся в инструментарии этой панели. Если мы отключили флажок **Save** категории **Remote Info** диалогового окна **Site Definition**, отказавшись от сохранения введенного нами пароля, Dreamweaver выведет небольшое диалоговое окно с полем ввода, где мы должны будем ввести пароль, и кнопками **OK** (соединение) и **Cancel** (отказ от соединения).

Установив соединение, мы сможем опубликовать наш сайт одним из описанных ранее способов. Да-да, публикация сайта на удаленном сервере по протоколу FTP выполняется точно так же, как и публикация на локальном сервере. (А почему, собственно, они должны выполняться по-разному?..)

Конечно, ждать при этом придется несколько дольше, так как канал доступа в Интернет, даже относительно быстрый, работает значительно медленнее жесткого диска. А чтобы нам не было скучно, Dreamweaver во время отправки сайта на сервер будет отображать ход операции в небольшом диалоговом окне. Мы можем прервать отправку, щелкнув по кнопке **Cancel** этого окна.

Единственное: закончив работу с удаленной копией сайта, нужно обязательно отключиться от FTP-сервера, для чего достаточно щелкнуть еще раз кнопку **Connect to Remote Host** инструментария панели **Files**.

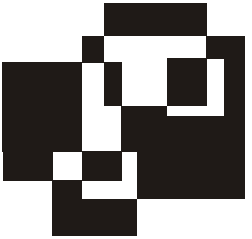
Внимание

Нужно обязательно отключаться от FTP-сервера после окончания работы с удаленной копией сайта! На поддержание каждого соединения с клиентом FTP-сервер расходует память серверного компьютера, и может наступить такой момент, когда другому клиенту этих ресурсов может не хватить. Давайте уважать других клиентов!

Что дальше?

Ура!!! Мы стали настоящими Web-дизайнерами! У нас есть свой Web-сайт в Интернете! У нас есть посетители и даже поклонники!

Что ж, это хороший повод, чтобы немного отдохнуть. А потом — вперед, на штурм неприступной крепости интернет-программирования. (Ход боевых действий см. в *части II* этой книги.)



Часть II

Наши первые серверные программы

Глава 5. Введение в Web-программирование

Глава 6. Базы данных

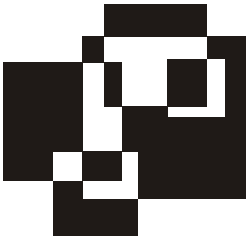
Глава 7. PHP — технология написания серверных приложений

Глава 8. Простейшие серверные Web-страницы. Вывод данных

Глава 9. Реализация ввода и правки данных

Создать первый Web-сайт было не так уж и сложно. Тем более, что при этом нам помогал Macromedia Dreamweaver.

Пора сделать второй шаг в мир современных интернет-технологий — заняться интернет-программированием. Наш второй сайт будет представлять собой самую настоящую программу, извлекающую данные из базы и превращающую их в обычные Web-страницы. И написать такую программу нам также поможет несравненный Dreamweaver.



Глава 5

Введение в Web-программирование

Вот мы и создали свой первый Web-сайт. Он прекрасно работает, он неплохо посещается, у него даже есть поклонники, что очень неплохо для "пробы пера". Так что же еще нам нужно?

Да, наш сайт пока еще не очень презентабелен внешне. Но это дело поправимое. Получше изучим Dreamweaver, HTML, CSS, напишем красивые страницы, подберем цвета, шрифты, оформим все это так, чтобы посетители не шарахались от одного их вида — и все будет замечательно. Основные принципы создания Web-страниц нам уже известны, а остальное приложится.

Но как только мы начнем добавлять в списки статей и файлов новые позиции, как только попытаемся расширить наш сайт, скажем, ввести в него списки вновь добавленных позиций или поиск, так столкнемся с непреодолимой стеной. Язык HTML не предлагает никаких инструментов для организации поиска, значит, нам нужно писать специальные поисковые программы. А как предоставить посетителю возможность отсортировать список статей не по названиям, а по авторам? Пользуясь только средствами HTML — никак.

И тут мы вступаем в область Web-программирования. Зачем и как — вот два вопроса, на который будет дан ответ в этой главе. Но чтобы дать этот ответ, придется начать издалека.

Недостатки статичных Web-страниц и их преодоление

Сначала мы поговорим о том, как хранится информация, которую мы выкладываем на наш сайт, и как она выдается посетителю. А именно о данных и их представлении. Разобравшись с этим, мы поймем и все остальное.

Данные и их представление

Давайте откроем любую из созданных нами ранее Web-страниц списков статей и файлов в любом текстовом редакторе или Dreamweaver, переключив его в режим отображения HTML-кода. (Можно открыть страницу списка категорий — это не суть важно. Только не нужно открывать главную страницу — никакой реальной информации, ради которой создавался сайт, она не содержит.) И внимательно посмотрим на нее. Что мы видим?

Если не обращать внимания на все хитросплетения кода HTML и хорошо подумать, можно выделить две части любой Web-страницы. Это сама информация, ради которой она, собственно, и создается, и представление этой информации для человека, пользователя, посетителя сайта. Давайте поговорим о них подробнее.

Информация — это сами данные, "сердце" любой Web-страницы. Как правило, она подчиняется жестким правилам, описывающим ее структуру, иначе говоря, — *структурирована*. Так, если мы посмотрим на список статей, то увидим, что он делится на отдельные строки, описывающие конкретные статьи, а каждая строка — на колонки названия статьи и имени автора. (Третья колонка с гиперссылками на статьи никакой реальной информации не содержит, поэтому не будем ее рассматривать.) Так что можно сказать, что наша информация структурирована в виде таблицы.

Кстати, структурированная в виде таблицы информация встречается на Web-страницах очень часто, поскольку она хорошо воспринимается посетителем. В самом деле, несравнимо удобнее иметь перед глазами таблицу (лучше всего — отсортированную по какому-либо столбцу), чем продираться через дебри сплошного текста. Кроме того, таблицы исключительно компактны — они занимают мало места на Web-страницах, что тоже очень важно.

Забегая вперед, скажем также, что таблицы прекрасно обрабатываются различными программами. Разумеется, в этом случае речь идет не о Web-страницах, а о способе хранения информации в файлах, обрабатываемых этими программами. В качестве классического примера можно привести реляционные базы данных, о которых мы узнаем в *главе 6*.

Замечание

Информация также может быть и *неструктурированной*. Типичнейший пример — главная страница нашего сайта, содержащая обычный текст. Информация, которую он несет, не подчиняется никакой структуре.

Хорошо, с информацией разобрались. А что же такое представление? *Представление* — это набор правил, согласно которым информация представляется конечному ее пользователю, в нашем случае — посетителю Web-сайта. Представление описывает, как форматируется каждый столбец таблицы, где он выводится, каков его порядковый номер, по какому столбцу сортируется

таблица и т. п. Можно сказать, что представление занимается переработкой "сырой" информации в конечный продукт — то, что пользователь видит на экране.

Кое-что из представления видно в HTML-коде невооруженным глазом. Во-первых, это секция заголовка нашей Web-страницы, содержащая служебные сведения для Web-обозревателя. Во-вторых, это гиперссылки перехода на тексты самих статей. В-третьих, это, конечно, таблицы стилей CSS, хоть они и не присутствуют в коде HTML, — просто хрестоматийный пример представления!

А теперь еще раз внимательно пробежимся по HTML-коду, тщательно отделяя информацию от представления. И узнаем одну очень неприятную, хоть и для кого-то забавную, вещь...

Недостатки статических Web-страниц

А именно — в наших Web-страницах информация фактически неразрывно связана с представлением!

В самом деле, HTML-код — это дикая мешанина "сырой" информации и правил ее отображения в Web-обозревателе. Так, порядок строк и столбцов жестко задан в самом коде; если нам нужно его изменить, скажем, поместить первым название статьи, нам придется переделывать весь код, меняя местами первую и вторую ячейки каждой строки. А это очень кропотливая работа.

Далее, попробуем добавить в список новую статью. Если мы поместим новую строку в конец таблицы, то она так и отобразится Web-обозревателем — в ее конце. И таблица уже не будет отсортирована по названиям статей. Нам придется вставить HTML-код, описывающий новую строку, точно в нужное место таблицы, раздвинув соседние строки.

А если нам нужно представить статьи не в виде таблицы, а в виде списка? А если нам нужно выводить список статей отсортированным по дате их добавления? Фактически, нам придется полностью переделывать Web-страницу, а это весьма трудоемко.

И еще. Статьи и файлы на нашем сайте разбиты на категории. И для каждой такой категории нам пришлось создавать отдельную страницу с отдельным списком. В результате, наш сайт состоит из множества абсолютно однотипных страничек, в которых разное только одно — информация.

Итак, мы выяснили, что представление информации на наших Web-страницах смешано с самой информацией, а вдобавок — и жестко задано. Именно поэтому классические страницы, написанные на языке HTML, и называются часто *статическими*.

Напрашивается вопрос: нет ли способа отделить информацию от представления? К сожалению, средствами HTML это сделать не получится. Хотя...

Первая попытка отделить информацию от представления, оказавшаяся удачной, — это таблицы стилей CSS. Мы уже знаем из *главы 2*, что для оформления текста на Web-страницах раньше использовали теги физического форматирования. Они "нагло влезали" прямо в информацию, внося в нее правила представления, а вдобавок — сильно запутывали HTML-код. Таблицы стилей позволили вынести хотя бы часть правил представления за пределы самой информации — подальше, в секцию заголовка, а то и в отдельные файлы.

К сожалению, таблицы стилей — частичное решение проблемы. Они задают только оформление информации: шрифт и цвет текста, выравнивание и величины отступов у абзацев и пр. С их помощью можно создать рамки вокруг абзаца текста или изображения, задать его размер и даже, если уж совсем нечего делать, заставить курсор мыши при наведении на абзац менять свою форму. (Как это сделать, описано в интерактивном справочнике по CSS, поставляемом в составе Dreamweaver.) Но задать сортировку таблицы или превратить ее в список они не могут.

Таблицы стилей были радикальным, хоть и односторонним, решением. Они стали дополнением к языку HTML, устраняющим его врожденные недостатки. Чтобы решить описанную ранее проблему полностью, нужно еще одно радикальное решение. И оно было найдено!

Серверные программы — радикальный способ отделить информацию от представления

Давайте отвлечемся от Web-дизайна и посмотрим на программы. Обычные программы, с которыми мы имеем дело каждый день: текстовые редакторы, электронные таблицы, графические пакеты и пр. Чем они могут нам помочь, что подсказать?

Взять тот же самый Microsoft Word — популярнейший и один из лучших (если не лучший) на данный момент текстовый редактор. Набранные в нем документы он сохраняет в файлах с расширением doc. Каким образом вся информация — текст, графика, сведения о форматировании — сохраняются там, нам не известно. Если открыть такой файл в обычном текстовом редакторе, оперирующий файлами txt (в том же Блокноте), он покажет нам непонятный набор закорючек.

Если же мы откроем файл doc в Microsoft Word (или любой другой программе, понимающей этот формат), мы увидим на экране отформатированный текст документа, возможно, содержащий графику и таблицы. То есть Word сам "знает", как отобразить и отформатировать документ, хранящийся в файле.

Так что же у нас получается? Выходит, файл документа Microsoft Word с расширением doc — это информация. А сам Microsoft Word — это пред-

ставление. Мы не знаем, как именно информация хранится в файлах doc — ее в приемлемом для чтения и правки виде нам выдает Microsoft Word.

Так вот оно, решение! Программа, работающая на том же компьютере, что и Web-сервер, читающая информацию из файлов и применяющая к ней правила представления, после чего информация в приемлемом для чтения виде выдается посетителю сайта. Если же посетителю нужно что-то сделать с информацией, он посылает этой программе запрос, в ответ на который она применяет к этой же информации другие правила представления (например, изменяет сортировку строк или выводит ее не в виде таблицы, а в виде списка).

Теперь предположим, что нам нужно изменить способ вывода этой информации посетителю. Мы переписываем эту программу, добавляя в нее новые правила, и помещаем ее на серверный компьютер. И все — саму информацию переделывать не придется!

Здесь возникают два интересных момента, на которые нам следует обратить особое внимание. И связаны они с тем, что посетитель, получающий эту информацию, работает с Web-обозревателем. Мы же Web-сайт делаем, в конце концов!

Во-первых, серверная программа, обрабатывающая информацию согласно правилам представления, должна работать совместно с Web-сервером. Ведь все Web-обозреватели могут работать только с Web-серверами; с другими программами они работать не "научены". Это значит, что Web-сервер будет принимать от клиентов запросы, перенаправлять их программе, обрабатывающей информацию, принимать от нее результаты обработки и отправлять их клиенту.

Во-вторых, результаты работы программы будут представлять собой обычную Web-страницу, которая и отображается в Web-обозревателе посетителя. Кроме Web-страниц (и еще обычных текстовых файлов), Web-обозреватель не поддерживает никаких других документов, так что с этим придется считаться.

Исходя из всего этого, давайте напишем последовательность получения информации от программы, работающей на серверном компьютере и применяющей к хранящейся отдельно информации заданные в ней правила представления.

1. Посетитель запрашивает с помощью Web-обозревателя некую информацию.
2. Web-обозреватель устанавливает соединение с Web-сервером и посылает ему клиентский запрос.
3. Web-сервер принимает запрос и расшифровывает его.
4. Web-сервер запускает нужную программу и, возможно, передает ей дополнительные параметры запроса.

5. Программа извлекает нужную информацию, записанную в файле или файлах, применяет к ней заданное представление и преобразует в Web-страницу, которую возвращает Web-серверу.
6. Web-сервер отправляет сформированную программой Web-страницу Web-обозревателю.
7. Web-обозреватель получает Web-страницу и выводит ее на экран, после чего разрывает соединение с сервером.

Осталось ввести несколько новых терминов, которые помогут нам не путаться в дальнейшем. Программу, работающую совместно с Web-сервером и занимающуюся обработкой информации согласно заданным программистом правилам представления, назовем *серверной программой*. А сформированная ею Web-страница, содержащая обработанную информацию, пусть называется *динамической страницей*.

Уже знакомая нам архитектура "клиент-сервер" дополнилась промежуточным звеном — серверной программой. Она стала *трехзвенной* — "клиент — серверное приложение — данные".

Кстати, серверные программы помогут нам решить еще одну проблему, не решаемую средствами HTML, а именно организовать обработку данных, отправленных посетителем сайта из Web-обозревателя. Нам довольно часто встречаются на Web-страницах так называемые *формы* — небольшие наборы элементов управления для ввода каких-либо данных посетителя (имя и пароль, адрес электронной и обычной почты, анкеты и пр.). Так вот, отправленные с их помощью данные обрабатываются именно серверными программами. Как писать такие программы и как создавать формы, мы узнаем в *главе 9*.

А пока что поговорим еще о серверных программах и их разновидностях. И узнаем, чем же мы будем заниматься на протяжении всей книги.

Технологии создания серверных программ

Разумеется, обычные, статические Web-страницы и серверные программы создаются совершенно по-разному. Более того, для этого используются совершенно разные языки. Да и сами принципы их создания сильно различаются. Вот об этих принципах, языках и различиях мы сейчас и поговорим.

Активные серверные Web-страницы

Когда шел разговор о серверных программах и принципах их работы, нам не могло не показаться, что временами они слишком напоминают Web-страницы. В самом деле, работают они совместно с Web-сервером, который по запросам клиентов запускает эти программы, передает им какие-то дополнительные параметры и принимает от них результаты. Даже интернет-

адрес серверной программы записывается так же, как адрес Web-страницы, за небольшим исключением:

```
http://www.somesite.ru/progs/prog1.exe
```

Здесь мы обратились к серверной программе prog1.exe, чей файл хранится в папке progs сервера `http://www.somesite.ru`. В ответ мы получим Web-страницу, сформированную этой программой на основе хранящихся на этом сервере данных. Результат почти такой же, как если бы мы обратились к обычной, статической Web-странице.

А поскольку серверная программа в качестве результата работы создает обычную Web-страницу, умные головы решили вот что. Почему бы в обычную Web-страницу не вставить небольшие фрагменты программного кода (так называемые *сценарии*) на каком-нибудь знакомом им языке программирования? Встретив такой фрагмент, Web-сервер обработает его и результат работы — хотя бы и таблицу со списком статей — вставит прямо в то же самое место Web-страницы. Разумеется, вырезав при этом сам фрагмент кода — Web-обозревателю он ни к чему.

Вероятно, идея создания таких вот *активных серверных Web-страниц*, содержащих и код HTML, и фрагменты программного кода, выполняющего обработку данных, что называется, носилась в воздухе. Только это объясняет практически одновременное появление сразу нескольких технологий создания таких страниц и соответствующего программного обеспечения, призванного "научить" Web-сервер понимать используемый в страницах программный код (так называемых *обработчиков*). Наибольшую популярность получили пять технологий, которые мы сейчас рассмотрим.

Самая первая технология создания активных серверных страниц, которую мы здесь рассмотрим, — это PHP (Personal Home Page, персональная домашняя страница). Она была разработана одним норвежским студентом еще в середине 90-х годов прошлого века и к настоящему времени стала, вероятно, самой популярной. В основном, популярность ее обусловлена тем, что программное обеспечение обработчика страниц PHP поставляется бесплатно с открытыми исходными текстами. Кроме того, обработчик поддерживает десятка три разных, популярных и малоизвестных, Web-серверов, а значит, может быть установлен практически где угодно.

Для написания сценариев в Web-страницах PHP используется особый язык программирования, основанный на популярном языке C. Этот язык так и называется — PHP. Он весьма прост для изучения и достаточно развит, чтобы создавать сложные программы. Что касается самих Web-страниц, содержащих код PHP, то они должны быть сохранены в файлах с расширением php — это обязательно, иначе обработчик посчитает их статическими Web-страницами.

В настоящее время сайты, содержащие страницы PHP, — обычное дело в Сети. Обработчик PHP установлен на большинстве бесплатных Web-серверов.

Да и на локальном Web-сервере установить его очень просто; как это делается, описано в *приложении 3* данной книги.

Примерно в то же самое время (середина 90-х) фирмой Microsoft была разработана технология серверных страниц ASP (Active Server Pages, активные серверные страницы). Изначально она использовалась в Web-сервере Microsoft Internet Information Server, но впоследствии появились решения третьих фирм, добавляющие обработчик страниц ASP в другие Web-серверы. Сценарии в страницах ASP пишутся на языке VBScript, хотя возможно также использование языка JavaScript и некоторых других. А сами страницы должны иметь расширение asp.

ASP.NET — это расширение технологии ASP, разработанное также Microsoft в начале XXI века. ASP.NET производительнее, надежнее, чем ASP, и имеет много дополнительных возможностей. В настоящее время ASP.NET поддерживается только сервером Microsoft Internet Information Server.

JSP (Java Server Pages, серверные страницы, написанные на JavaScript) — это своего рода "адекватный ответ" на ASP, разработанный фирмой Netscape также в середине 90-х годов прошлого века для своего собственного Web-сервера Netscape Web Server. JSP практически ничем не отличается от ASP за тем исключением, что для написания сценариев используется язык JavaScript. Сейчас технология JSP весьма популярна; обработчики страниц JSP существуют для довольно большого количества Web-серверов, включая и некоторые бесплатные.

Последняя технология, достойная упоминания, — это ColdFusion, разработанная фирмой Macromedia все в тех же "золотых" 90-х годах. Эта технология имеет коренное отличие от перечисленных ранее — вместо сценариев там используются особые теги, похожие на теги HTML, но предназначенные для обработчика страниц. Встретив такой тег, обработчик выполняет выборку и обработку данных и помещает результат в то место HTML-кода, где встретился этот тег. Технология ColdFusion поддерживается многими Web-серверами, позволяет манипулировать большими массивами данных и очень надежна, но до сих пор не получила широкого распространения (вероятно, из-за высокой цены обработчика страниц ColdFusion).

Существует также несколько других, менее популярных технологий активных серверных страниц. Как правило, они поддерживаются только каким-то одним Web-сервером и поэтому не получили распространения. Мы не будем их рассматривать.

Другие технологии серверного программирования

Напоследок вкратце коснемся других технологий создания серверных программ, которые сейчас имеют хождение. Всего их три.

Технология CGI (Common Gateway Interface, общий интерфейс обмена) — самая старая, но пока еще не устаревшая. Программы CGI — это обычные исполняемые файлы с расширением exe либо программы, написанные на языках Perl, Python и т. п. В ответ на клиентский запрос Web-сервер запускает копию такой программы, передает ей данные, принимает результаты, после чего завершает ее. Программы CGI очень легко создавать и отлаживать, но если Web-сервер получит слишком много клиентских запросов и, следовательно, запустит слишком много таких программ, то ресурсов серверного компьютера может не хватить, и он "зависнет".

Технология ISAPI (Internet Server Application Programming Interface, интерфейс программирования приложений интернет-сервера), разработанная фирмой Microsoft, свободна от этих недостатков. Программа ISAPI — это динамическая библиотека Windows DLL, которая постоянно работает совместно с Web-сервером и выполняет все поступившие клиентские запросы одновременно. Такие программы требуют меньше компьютерных ресурсов, но их очень сложно создавать и отлаживать. Существует также *технология NSAPI* (Netscape Server Application Programming Interface, интерфейс программирования приложений сервера Netscape), аналогичная ISAPI, но разработанная фирмой Netscape.

Очень близки к программам ISAPI и NSAPI так называемые *расширения Web-сервера*. Они также запускаются в единственном экземпляре и постоянно работают вместе с Web-сервером, выполняя все клиентские запросы. Единственное отличие: технологии ISAPI и NSAPI стандартизированы и поддерживаются многими Web-серверами, а расширения пишутся под какой-то один Web-сервер. Наиболее популярны сейчас расширения Web-сервера Apache, и это неудивительно, ведь сам Apache — популярнейший в настоящее время Web-сервер.

Осталось сказать только то, что эти технологии используются для написания самих программ обработчиков активных серверных страниц. Так, обработчик PHP существует как программа ISAPI, как расширение Web-обозревателя и как приложение CGI. Так что его можно "приспособить" практически к любому более-менее серьезному Web-серверу.

Наш второй Web-сайт будет использовать активные серверные страницы

Да, именно так! Пора прикоснуться к мощи серверного программирования.

Мы сохраним списки категорий, статей и файлов, которые хотим выложить на сайт, в файле данных. Этот файл будет храниться на Web-сервере, где-либо вне корневой папки сайта, чтобы никто из посетителей не смог его

загрузить и просмотреть. А для выборки данных мы используем две серверные страницы. Всего две!

- Categories.php будет выводить посетителю список категорий. В зависимости от того, по какой гиперссылке главной страницы щелкнул посетитель, это будет список категорий статей или файлов.
- Items.php покажет посетителю список статей или файлов, относящихся к выбранной им категории.

Главная страница — default.htm — у нас останется старой. Информация, находящаяся на ней, изменяется нечасто и не требует какой-либо структурированности. (Единственное — нам потом придется переделать интернет-адреса в гиперссылках `файлы` и `статьи`, но это минутное дело.)

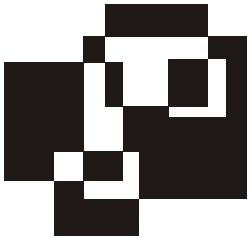
Разумеется, для создания серверных страниц мы выберем технологию PHP. Она широко распространена, поддерживается очень многими бесплатными Web-серверами, исключительно просто устанавливается и настраивается. Когда мы начнем тестировать наш новый сайт, мы установим обработчик страниц PHP на свой компьютер и подключим его к нашему локальному Web-серверу.

У нас получится замечательный сайт.

Что дальше?

Вот мы и сделали первый шаг в мир серверного интернет-программирования. Маленький, робкий, но все-таки шаг.

Стоп! Вот мы все говорили об обработке информации, о применении правил представления, о серверных программах и технологиях их создания. Но ни единым словом не обмолвились, как и где хранится сама информация, которую мы собираемся обрабатывать. Пора восполнить данный пробел в нашем образовании! Следующая глава как раз и будет посвящена этому.



Глава 6

Базы данных

Прежде чем начать обработку информации, ее нужно где-то найти и как-то сохранить. Вопросы поиска и сбора информации мы здесь рассматривать не будем. Давайте лучше поговорим о том, как сохранить собранную информацию, чтобы она никуда не пропала, но все время была под рукой.

Если говорить применительно к компьютерам, то информация на их дисках хранится в файлах. Это известно всем, кто хоть раз сохранял набранный в Microsoft Word документ. Файлы могут хранить программный код, данные конфигурации, текст, графику, звук, видео, таблицы и др. А обработкой всего этого богатства занимаются специальные программы, которые знают, что с этой информацией делать.

Что касается структурированных данных, о которых мы вели речь в *главе 5*, то для их хранения используются совершенно особые и весьма примечательные файлы. Их так и называют — базы данных.

Введение в реляционные базы 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*. Пример такой СУБД — популярнейшая программа Microsoft Access, работающая с базами данных mdb.

Базы данных делятся на несколько видов по системе структурирования содержащейся в них информации. В компьютерной индустрии самое широкое распространение нашли так называемые реляционные базы данных, о которых сейчас и пойдет разговор.

Что такое реляционные базы данных

Реляционные базы данных служат для хранения структурированной в виде таблиц информации, связанной друг с другом. Если говорить короче и проще,

то реляционная база данных содержит набор связанных друг с другом таблиц. Именно этот вид баз данных сейчас наиболее популярен.

Дело в том, что очень большой класс информации, обрабатываемой компьютерами, можно представить в виде набора таблиц. Это различные списки, журналы, каталоги, бухгалтерские книги, справочники и мн. др.; собственно, это и есть таблицы. А такая информация прямо-таки просится в реляционную базу данных. Вдобавок реляционные базы данных позволяют задать жесткие правила правильности и целостности информации, что тоже немаловажно. А к тому же, они еще и очень быстро обрабатываются соответствующими реляционными СУБД (об этом говорилось еще в *главе 5*).

Замечание

Существуют и *нереляционные* базы данных, информация в которых структурируется по другим правилам: сетевые, объектные и пр. Применяются они редко и только в специальных случаях, поэтому мы не будем их рассматривать.

Примеры реляционных СУБД: уже упоминавшаяся Microsoft Access, Corel Paradox, Borland dBase, Microsoft FoxPro. Это все *пользовательские* или *настольные* СУБД, предназначенные для обычного пользователя. Они хранят свои файлы на локальном диске пользовательского компьютера или дисках файлового сервера и работают с этими файлами напрямую.

Для обработки больших массивов данных, к которым, помимо того, подключается очень много пользователей, служат более мощные программы: Borland InterBase, MySQL, Microsoft SQL Server, Informix, Sybase, Oracle. Они функционируют по другому принципу, а как именно — мы узнаем позже.

Каждая из этих программ работает со своим собственным форматом файлов, хотя настольные СУБД, как правило, могут открывать и "чужие" файлы. Одна база данных может занимать как один большой файл (Microsoft Access, Borland InterBase), так и множество более мелких (Corel Paradox, Borland dBase, Oracle); в последнем случае эти файлы должны находиться в одной папке.

А теперь давайте познакомимся с реляционными базами данных поближе. Ведь именно с ними мы и будем работать.

Составные части реляционной базы данных

Реляционная база данных состоит из четырех частей, которые мы рассмотрим более подробно в следующих далее разделах.

Таблицы, поля и записи

Таблица — это набор структурированных данных. Пример таблицы представлен на рис. 6.1.

date	author	name
12.08.2004	Криворукий, Ю.	Тяп-ляп Web-дизайнер
24.09.2004	Сусанин-мл., И.	Путеводитель по всемирной паутине
30.10.2004	СуПеРхАкЕр	Как взломать компьютер с помощью кувалды
10.07.2004	Дуболом, Е.	К вопросу о прочности модемов

Рис. 6.1. Таблица — список статей

Эта таблица содержит список статей из трех столбцов: дата добавления статьи в список, автор статьи и ее название. Не стоит выдумывать какие-то абстрактные данные, если у нас уже есть готовые.

Таблица будет записана в файл базы данных практически в таком же виде: ячейки составляют строку, а строки — таблицу. Открыв этот файл, СУБД моментально разберется, что к чему, и выведет таблицу на экран в подходящем для работы с ней виде.

Каждая таблица, сохраненная 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.

Теперь давайте введем несколько новых терминов, которыми оперируют все пользователи и программисты, работающие с базами данных.

Для начала назовем отдельную строку таблицы, содержащую реальные данные, *записью*. (Строка заголовка, выделенная на рис. 6.1 черным цветом, записью являться не будет, так как содержит не реальные данные, описывающие какую-либо статью, а служебные сведения — названия столбцов.) На каждую статью, занесенную в список, будет отводиться одна запись.

Далее, пусть каждая ячейка отдельной строки-записи называется *полем*. Каждое поле обязано иметь уникальное в пределах таблицы имя; имена полей, кстати, и приведены в строке заголовка таблицы. Можно сказать, что поле — это порция данных, составляющих запись. А сами данные, помещенные в поле, называются его *значением*.

Записи разделяются на поля для удобства обработки таблицы программами СУБД. Так, выделив дату добавления статьи в отдельное поле с именем `date` (конечно, это поле можно назвать и по-другому), мы получим возможность сортировать записи по этому полю. СУБД в ответ на наш запрос считает значения полей `date` всех записей таблицы и переупорядочит их, чтобы выстроить их по возрастанию или убыванию даты. Также мы сможем отбирать для просмотра только те записи, значение поля `date` которых укладывается в определенный диапазон — СУБД просто не выведет на экран записи, не удовлетворяющие заданному нами условию.

Каждое 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 чисел. Типы данных, поддерживаемые большинством форматов баз данных, приведены в табл. 6.1.

Таблица 6.1. *Типы данных, поддерживаемые большинством форматов баз данных*

Название	Описание
Строковый	Строки фиксированной длины, содержащие любые символы: буквы, цифры, знаки препинания, пробелы и пр.
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
Счетчик	Постепенно увеличивающиеся и не повторяющиеся в пределах таблицы целые числа. Поля такого типа используются для специальных целей

Замечание

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 позже.

Жесткое задание типа данных, которые могут храниться в поле, также служит для удобства обработки данных СУБД. Так, значения целочисленных полей и полей с плавающей точкой могут участвовать в арифметических операциях. Логические поля занимают очень мало места и быстро обрабатываются. А поля memo обрабатываются очень медленно, зато могут хранить текст произвольного размера, а без этого часто не обойтись.

Давайте еще раз посмотрим на рис. 6.1. Представленная там таблица имеет три поля. А какого они типа? Давайте подумаем.

Поле `date` мы вводили для того, чтобы получить возможность сортировки по этому полю и поиска самых "свежих" статей. Значит, для ускорения обработки ему нужно будет присвоить тип даты. А поля `author` и `name` содержат обычный текст, так пусть они будут строковыми.

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

Правила

Ранее мы узнали, что поле таблицы имеет два обязательных параметра: имя и тип данных. Эти параметры задаются при создании таблицы и впоследствии могут быть изменены, например, если были заданы ошибочно или при переделке структуры таблицы.

Большинство форматов записи данных также поддерживают еще один параметр поля — *правила поля*. Правила задают условия, которым должны удовлетворять данные, записываемые в поле. Такими условиями могут быть:

- обязательное наличие в поле какого-либо значения (*обязательное поле*);
- диапазон, в который должно попадать числовое значение или дата;
- взаимосвязь значения данного поля со значениями других полей той же записи;
- значение, которое должно быть помещено в поле при создании новой записи (*значение поля по умолчанию*).

Из правил поля чаще всего применяются обязательные поля и значения по умолчанию — они поддерживаются большинством форматов баз данных. Реже применяются правила-диапазоны, еще реже — взаимосвязи значений данного поля и других полей таблицы.

Для примера обратимся снова к нашей таблице, приведенной на рис. 6.1. Поле `date` содержит дату добавления статьи в список, и вполне логично, чтобы его значение записывалось сразу при создании новой записи. Значит, мы можем создать для этого поля правило — значение по умолчанию, равное текущей дате.

Правила также могут быть применены не к отдельному полю, а сразу ко всей таблице (*правила таблицы*). К таким правилам относятся:

- требование уникальности значений какого-либо поля в пределах таблицы (*уникальное поле*);
- взаимосвязь значения какого-либо поля со значениями других полей всех записей этой таблицы.

Из правил таблицы чаще всего используются уникальные поля. Как правило, уникальные поля являются ключевыми (о ключевых полях см. далее). Взаимосвязи, опять же, применяются значительно реже, так как поддерживаются не всеми форматами баз данных, да и нужда в них возникает весьма нечасто.

Индексы и ключи

Предположим, что мы уже создали таблицу, изображенную на рис. 6.1, и теперь заполняем ее данными. Мы создали уже довольно много записей, и возникла потребность вывести их на экран отсортированными по какому-

либо полю. Здесь мы столкнемся с проблемой, о которой стоит поговорить подробнее.

Дело в том, что все программы работы с базами данных добавляют записи в таблицу в том порядке, в котором они вводятся. В результате записи оказываются не отсортированными, точнее, отсортированными по порядку их добавления. А если нам нужно отсортировать их по дате добавления (значение поля `date`)? Что делать в этом случае?

Собственно, хорошая СУБД должна уметь сортировать при выводе записи по любому заданному нами порядку. Только вот делать она это будет очень медленно. В самом деле, СУБД будет должна:

1. Прочитать из файла все записи, которые нам нужны.
2. Поместить их в оперативную память.
3. Отсортировать их по заданному порядку.
4. Вывести отсортированные записи на экран.

Если записей немного, то это, конечно, не займет много времени (и памяти). А если много? (Вот, кстати, именно поэтому СУБД отбирают при работе так много оперативной памяти — им нужно где-то хранить записи, участвующие в обработке.)

Есть ли способ решить эту проблему? Есть. И мы сейчас его рассмотрим.

Предположим, что мы создали прямо в базе данных особый блок значений поля, по которому мы должны сортировать записи. Причем значения в этом блоке расположены в том порядке, в котором должны быть отсортированы записи. Кроме того, этот массив содержит ссылки на записи, которым соответствуют сохраненные в нем значения поля (рис. 6.2).

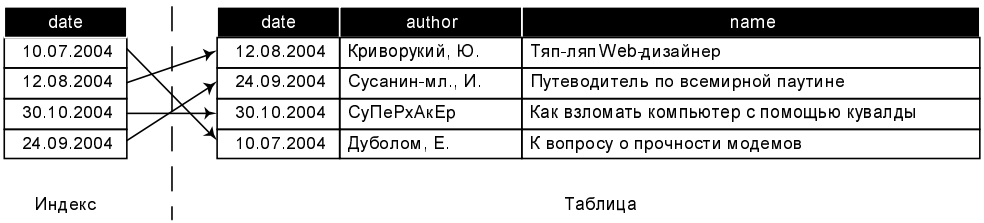


Рис. 6.2. Таблица — список статей с индексом

Назовем такой массив *индексом*. И посмотрим, что случится, если мы потребуем от СУБД отсортировать записи таблицы по полю, значения которого хранятся в индексе (*индексированному полю*).

СУБД откроет файл базы данных, найдет нужную таблицу и созданный для ее поля индекс, считает и то, и другое в память. Зная порядок следования записей для выполнения нужной сортировки — он, собственно, и записан

в индексе — СУБД очень быстро переставит записи таблицы согласно этому порядку и выведет отсортированную таблицу на экран.

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 создавать слишком много индексов — они будут обновляться очень долго.

Кроме сортировки, индексы также могут помочь при выполнении операции *фильтрации* записей — отбора их согласно некоторому заданному пользователем набору правил (*критерию*). СУБД считывает индекс в память, ищет значения, удовлетворяющие заданному критерию, и извлекает нужные записи из таблицы. Просто и быстро!

Индексы могут содержать и больше одного поля. Такие индексы называются *сложными*. Сложные индексы обрабатываются значительно дольше обычных (*простых*), поэтому используются довольно редко.

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 *первичным ключом* или *ключевым индексом*, а задействованное в нем поле — *ключевым*. Ключевой индекс может быть только один на всю таблицу.

Ключевое поле (или поля), участвующее в ключевом индексе, должно удовлетворять следующим условиям:

- оно обязательно должно содержать значение (то есть быть обязательным полем);
- оно должно содержать уникальные в пределах таблицы значения (быть уникальным полем). В случае сложного ключевого индекса это относится к совокупности значений всех полей, участвующих в этом индексе.

Можно сказать, что ключевой индекс при создании также создает в таблице два правила: обязательное поле (правило поля) и уникальное поле (правило таблицы).

Посмотрим на рис. 6.3. Там показана таблица, содержащая список категорий. Она имеет два поля: `name` — название категории и `file` — логическое поле, обозначающее, описывает ли данная запись категорию файлов.

Кроме того, эта таблица содержит ключевой индекс, содержащий — внимание! — оба этих поля. Если бы мы включили в индекс только поле `name`, в нашем индексе появились бы два одинаковых значения `Интернет`, принадлежащие, тем не менее, разным записям. СУБД выдала бы нам сообщение об ошибке и не создала бы такой индекс.

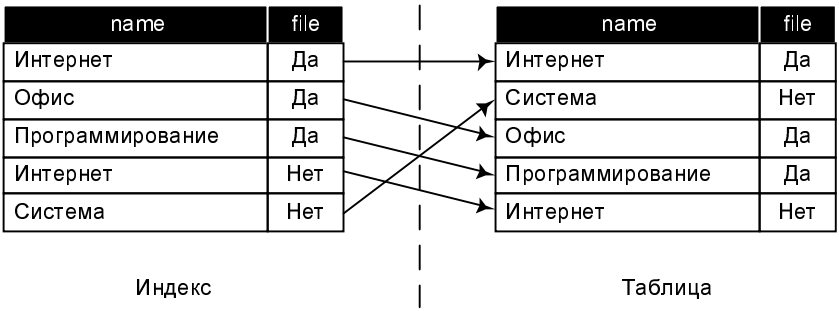


Рис. 6.3. Таблица — список категорий со сложным ключевым индексом

Поскольку у нас получился сложный индекс, содержащий два поля, давайте попытаемся от него избавиться. Создадим в таблице со списком категорий еще одно поле — `id` — и дадим ему тип счетчик. После чего создадим на основе одного этого поля ключевой индекс — см. рис. 6.4. Поскольку поле типа счетчик гарантированно содержит уникальные значения, это нам удастся без труда, и СУБД возражать не будет.

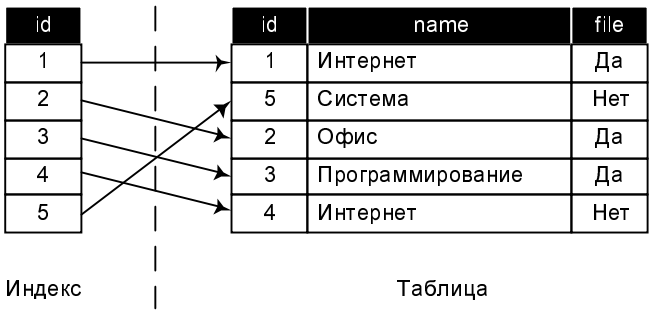


Рис. 6.4. Таблица — список категорий с суррогатным ключевым индексом

Такой ключевой индекс, содержащий не реальные, а "отвлеченные" данные, называется *суррогатным*. Практически всегда суррогатными делаются только ключевые индексы.

Возникает вполне резонный вопрос: зачем нужен такой индекс? Ведь ничего реально он не сортирует. Дело в том, что такие ключевые индексы используются для особых случаев, а именно для организации связей между таблицами, о которых будет говориться чуть позже. Кроме того, ключевые индексы (и суррогатные, и обычные) нужны, чтобы однозначно идентифицировать необходимую запись, например, для ее изменения или удаления. Но об этом речь также пойдет потом, когда мы начнем изучать язык управления данными SQL.

Многие форматы баз данных требуют, чтобы все индексы, кроме ключевых, имели уникальные в пределах таблицы имена. Ключевой же индекс имени иметь не должен, поскольку он один на всю таблицу. Другие же форматы вообще не требуют именования индексов.

Связи

Вернемся к таблице — списку статей, показанной на рис. 6.1. Чего нам в ней не хватает? Правильно — сведений о том, какой категории принадлежит каждая статья. Давайте добавим их, благо таблица — список категорий у нас уже есть (см. рис. 6.4).

На первый взгляд, это делается просто. Нужно только создать в таблице — списке статей еще одно строковое поле, которое будет содержать название категории. Вот так — см. рис. 6.5.

date	author	name	category
12.08.2004	Криворукий, Ю.	Тяп-ляп Web-дизайнер	Интернет
24.09.2004	Сусанин-мл., И.	Путеводитель по всемирной паутине	Интернет
30.10.2004	СуПеРхАкЕр	Как взломать компьютер с помощью кувалды	Система
10.07.2004	Дуболом, Е.	К вопросу о прочности модемов	Интернет

Рис. 6.5. Таблица — список статей с названиями категорий

Просто, да. Но это как раз такая простота, которая хуже воровства. Давайте выясним почему.

- Самые большие поля (то есть занимающие больше всего места в файле базы данных) — это строковые и memo. Причем если размер поля memo всегда равен размеру его значения, то размер строкового поля задается во время его создания и всегда остается постоянным. И нет никакой разницы, запишем мы туда один символ, строку, занимающую все поле, или вообще оставим пустым. Поэтому желательно свести количество строковых полей в таблице к минимуму.
- Созданное нами поле `category` фактически содержит ограниченный набор одних и тех же значений — названий категории. Опять же, это не очень красиво — поля по возможности должны содержать значения, уникальные для каждой записи. Так предписывают правила создания реляционных баз данных.
- Давайте посмотрим на нашу таблицу с практической точки зрения. Предположим, нам нужно изменить название какой-либо категории: исправить ошибку, написать слово "Интернет" по-английски — "Internet" и т. п. Для этого нам придется просмотреть всю таблицу, найти все записи, значение поля `category` которых нам нужно изменить, и внести исправления в каждой из найденных записей.
- У нас есть таблица — список категорий. Мы что, зря ее создавали?!

Напрашивается вот такое решение. А что если хранить в таблице — списке статей не само название категории, а ссылку на соответствующую запись таблицы — списка категорий? Благо таблица — список категорий имеет суррогатный ключевой индекс, а значит, мы можем однозначно указать на нужную нам запись-категорию. Возможно ли такое?

Конечно, возможно. Посмотрим на рис. 6.6. Там изображена таблица — список статей, в которой создано новое числовое поле `catid` (поле `category` удалено). Это поле хранит значение поля `id` таблицы — списка категорий, соответствующее нужной записи-категории. При обработке таблицы — списка статей СУБД извлечет эти значения, найдет соответствующие записи в таблице — списке категорий и выведет значения полей `name` этих записей (названия категорий) на экран.

date	author	name	catid
12.08.2004	Криворукий, Ю.	Тяп-ляпWeb-дизайнер	1
24.09.2004	Сусанин-мл., И.	Путеводитель по всемирной паутине	1
30.10.2004	СуПеРхАкЕр	Как взломать компьютер с помощью кувалды	5
10.07.2004	Дуболом, Е.	К вопросу о прочности модемов	1

name	file	id
Интернет	Да	1
Система	Нет	5
Офис	Да	2
Программирование	Да	3
Интернет	Нет	4

Рис. 6.6. Таблица — список статей, связанная с таблицей — списком категорий

Фактически мы создали *связь* между таблицами нашей базы данных, при которой таблица — список статей обращается за дополнительными данными к таблице — списку категорий. Таким образом мы убили всех зайцев: наша база данных уменьшилась в размерах, поля каждой записи содержат только уникальные для данной записи значения, да и таблица — список категорий оказалась у дел. А чтобы изменить название какой-либо категории, достаточно исправить значение поля `name` таблицы — списка категорий.

Профессиональные разработчики баз данных называют *первичной* или *родительской* таблицу, содержащую дополнительные данные, и *вторичной* или *дочерней* — таблицу, обращающуюся за дополнительными данными к первичной таблице. Значит, в нашем случае список статей — вторичная таблица, а список категорий — первичная. Кстати, связь, при которой на одну запись первичной таблицы могут ссылаться несколько записей вторичной таблицы, называется связью *"один-ко-многим"*.

Поле `catid` таблицы — списка статей, содержащее ссылки на записи первичной таблицы, 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.

Осталось только сказать, что реляционные базы данных получили свое название именно из-за того, что они состоят из набора связанных таблиц (от английского relation — связь).

Замечание

Некоторые форматы баз данных позволяют задать связь прямо в структуре таблиц (метаданных). Это предоставляет весьма полезные возможности, например, задание связей типа *"один-к-одному"*, когда на одну запись первичной таблицы может ссылаться только одна запись таблицы вторичной, или поддержание ссылочной целостности. (О ссылочной целостности см. *главу 12*.) Другие же форматы баз данных не позволяют задавать связи между таблицами в их структуре; забота об этом ложится на программиста, пишущего программы для обработки данных, или пользователя, запрашивающего данные у СУБД.

Вот, собственно, и все о четырех составных частях реляционной базы данных. Теперь давайте поговорим о самих программах СУБД, которые занимаются их обработкой.

Замечание

Некоторые форматы баз данных также позволяют хранить в базе небольшие программы, обрабатывающие эти данные. Такие программы бывают двух видов. *Триггеры* выполняются при операциях добавления, изменения и удаления записи и могут выполнять какую-либо дополнительную проверку или обработку данных. *Хранимые процедуры* вызываются самим пользователем и используются для сложной выборки или обработки данных.

Настольные и серверные реляционные СУБД

Нам уже известно, что СУБД — это программа для работы с базами данных. Именно с помощью СУБД пользователь и другие программы получают доступ к данным, хранящимся в базе.

Как правило, любая СУБД состоит из двух частей. Первая часть — это та программа, с которой работает пользователь, — *клиент данных*. Вторая же часть непосредственно занимается базой данных: принимает от клиента данных запросы на выборку и изменение данных, выполняет их и возвращает клиенту. Это так называемый *процессор данных*. Можно сказать, что клиент данных занимается приемом запросов от пользователя и выводом результатов, а процессор — собственно обработкой данных.

И в зависимости от того, как реализованы клиент и процессор данных, СУБД делятся на две большие группы: настольные и клиент-серверные. Сейчас мы о них поговорим.

Настольная СУБД реализована в виде одной-единственной программы; и клиент, и процессор данных слиты воедино в одном исполняемом файле. (Конечно, бывает всякое... Например, в Microsoft Access процессор данных реализован в виде набора библиотек DLL, но все равно фактически составляет с клиентом неразрывное целое.) Это первое отличие. Второе отличие: настольная СУБД работает непосредственно с файлами баз данных, точно так же, как Microsoft Word работает с файлами документов.

Когда пользователю нужно получить данные из базы, он с помощью СУБД открывает содержащий эту базу файл. СУБД считывает начало файла (так называемый *заголовок файла*), содержащее служебную информацию, загружает первый фрагмент данных и обрабатывает его, потом — второй, третий и т. д., пока все нужные пользователю данные не будут выведены на экран. Если пользователь изменяет какие-то данные, СУБД записывает их в нужное место файла, изменяет различные служебные структуры и, возможно, записывает что-либо в заголовок файла. Закончив работу, пользователь закрывает файл с базой данных.

Обычный принцип, и, вроде бы, неплохой. К тому же настольные СУБД работают весьма быстро, но только в том случае, если файл базы данных находится на дисках того же компьютера, где установлена сама СУБД. Если же файл нужной пользователю базы находится на другом компьютере, скорость работы СУБД резко падает, ведь по сети данные пересылаются значительно медленнее, чем внутри компьютера. А если одну и ту же базу открыли сразу несколько пользователей, работать становится совершенно невозможно — бóльшую часть времени пользователь ждет, пока СУБД получит очередной фрагмент данных из файла базы.

Поэтому еще в 60-х годах прошлого века были созданы *серверные* СУБД (или *серверы данных*), принадлежащие ко второй группе. Серверная СУБД — это процессор данных, оформленный в виде отдельной программы и работающий на специально выделенном для этого серверном компьютере. Как и любой другой сервер, он принимает от клиентов запросы, считывает из файла базы данные, обрабатывает их и пересылает результаты обработки клиентам.

Да, но какие программы можно использовать в качестве клиентов серверных СУБД? Ну, с этим проблем нет. Во-первых, это специально написанные программы, предназначенные для работы с какой-то определенной базой данных. Как правило, и клиенты, и сама база данных поставляются в виде единого пакета, предназначенного для хранения и обработки какого-то рода данных: бухгалтерских, каталогов, продаж и пр. Во-вторых, очень и очень многие программы настольных СУБД поддерживают работу с их серверными "коллегами".

Поскольку набор программ, работающих с серверной СУБД, весьма обширен, нужно как-то "научить" их всех взаимодействовать с ней по сети. Для этого

на клиентских компьютерах, кроме самого клиента данных, устанавливается также и *клиентская часть сервера* — небольшая программа, выступающая посредником между клиентом и сервером данных. Клиент передает свои запросы клиентской части сервера, та "упаковывает" их в сетевые пакеты и отправляет серверу. А приняв от сервера ответ, все та же клиентская часть "распаковывает" его и отправляет клиенту.

Преимущества настольных СУБД: исключительная легкость установки и использования и нетребовательность к дополнительному программному обеспечению (ведь им не нужен сервер данных). Недостатки: невысокое быстродействие при многопользовательском доступе к базе данных по сети, недостаточная надежность и защищенность. Поэтому настольные СУБД используются для ведения персональных баз данных (телефонных книг, каталогов литературы в домашней библиотеке) и для создания совсем небольших, как правило, несетевых систем обработки данных.

Преимущества серверных СУБД: бóльшая производительность (поскольку по сети пересылаются только запросы и ответы, которые меньше по размерам, чем фрагменты файлов), бóльшая надежность и защищенность. Недостатки: сложность установки, настройки и сопровождения. Но, поскольку серверные СУБД применяются для создания больших — уровня предприятия — систем обработки данных, эти недостатки не играют особой роли.

Для хранения данных, с которыми работают рассмотренные нами в *главе 5* серверные программы, в основном, используются серверные СУБД. Однако, если сайт невелик по размерам и не очень часто посещается, можно использовать и настольные СУБД. Именно так поступил автор этой книги: его сайт использует для хранения данных базу данных Microsoft Access 97.

Осталось привести примеры программ настольных и серверных СУБД. Итак, настольные СУБД — это хорошо знакомые нам Microsoft Access, Corel Paradox, Borland dBase, Microsoft FoxPro и другие, менее известные. А к серверным СУБД относятся Borland InterBase, MySQL, Microsoft SQL Server, PostgreSQL, Informix, Oracle, Sybase, IBM DB2 и мн. др.

Замечание

Существует также особый класс программ для работы с данными — так называемые *универсальные процессоры данных*. Эти программы служат для предоставления настольным СУБД возможности работы с различными форматами баз данных, как настольных, так и серверных. Типичный пример универсального процессора данных — ODBC (Open DataBase Connectivity, открытый доступ к базам данных), поставляемый в составе Windows, и его аналог JDBC (Java DataBase Connectivity, доступ к базам данных для программ, написанных на языке Java). (ODBC, кстати, использовал автор этой книги для создания своего сайта.) В настоящее время практически все настольные СУБД поддерживают ODBC; JDBC распространен гораздо меньше.

Среди серверных СУБД выделяется программа MySQL. Это весьма мощный, очень быстрый и нетребовательный к ресурсам сервер данных, к тому же, бесплатный и распространяемый с открытыми исходными текстами. Именно его используют в подавляющем большинстве Web-сайтов.

Язык обработки данных SQL

Ну, вот и все о реляционных базах данных. Теперь разговор пойдет о том, каким образом клиенты данных формируют свои запросы к серверу. А именно о языке обработки данных SQL.

Зачем нужен SQL

Существует довольно много программ СУБД. Чуть меньше существует форматов хранения данных в базах. Давайте посмотрим на список различных СУБД, приведенный ранее: каждая СУБД хранит данные в своем формате.

С настольными СУБД все просто: мы открываем файл базы данных, и программа сама разбирается с ним. Если же формат, в котором сохранен этот файл, не относится к "понимаемым" ею, тогда — увы! — придется искать другую, более "грамотную" программу. (Можно также воспользоваться универсальным процессором данных наподобие ODBC, но тогда не будут доступны различные дополнительные возможности, предоставляемые этим форматом баз данных.)

Серверные же СУБД создавались для того, чтобы предоставить надежный доступ к данным для любых программ, которым эти данные нужны. Часто бывает так, что пользователь вообще не знает, к какому серверу данных он обращается. (Собственно, это и не должно его беспокоить.) Выходит, нужен какой-то стандартный способ послать серверу данных запрос и получить от него ответ, что-то вроде стандартного языка запросов. И такой язык был создан — SQL (Structured Query Language, структурированный язык запросов).

Язык SQL позволяет записать запросы на извлечение, добавление, изменение и удаление записей в виде вполне осмысленных предложений; в этом смысле он близок к обычному английскому языку. Для написания запросов служит относительно небольшой набор *ключевых слов* — особых команд, понимаемых сервером данных.

Итак, чтобы клиент данных нормально взаимодействовал с сервером, оба должны понимать язык SQL. Представим схему такого взаимодействия:

1. Клиент данных каким-либо образом формирует запрос на языке SQL. (Он может сделать это сам либо попросить пользователя, чтобы он ввел запрос вручную. Заказные специализированные программы используют первый подход, а настольные СУБД с возможностью доступа к серверам данных — оба.)

2. Клиент данных передает сформированный запрос клиентской части сервера данных, установленной на клиентском компьютере.
3. Клиентская часть "упаковывает" принятый запрос в сетевые пакеты и передает его серверу данных.
4. Сервер данных принимает запрос, расшифровывает его и выполняет, после чего отсылает результат обратно.
5. Клиентская часть сервера данных принимает результат, "распаковывает" его и возвращает клиенту данных.
6. Клиент данных принимает результат и выводит его на экран либо предпринимает какие-то действия (например, сообщает пользователю об ошибке).

Основной набор ключевых слов SQL жестко стандартизирован и поддерживается всеми серверами данных. Но каждый сервер для поддержки собственных уникальных возможностей может ввести в SQL свои ключевые слова, которые поддерживаются только им. Эти уникальные возможности и соответствующие им ключевые слова описываются в документации к серверу данных.

Замечание

Для работы с универсальными процессорами данных наподобие ODBC также используется язык SQL.

Далее будет описан основной набор ключевых слов SQL, предназначенных для обработки данных.

Выборка записей из таблицы

Начнем с запросов выборки данных как с самых простых. Заодно изучим некоторые вещи, которые в дальнейшем нам обязательно понадобятся.

Простейшие запросы выборки данных

Формат простейшего запроса SQL для выборки данных из таблицы таков:

```
SELECT [DISTINCT] *|<список полей, разделенных запятыми>
↳FROM <имя таблицы>;
```

Сразу же после ключевого слова `SELECT` идет список полей, которые нужно извлечь из таблицы. Он состоит из имен полей, разделенных запятыми; при этом поля в результате будут следовать тому же порядку, в каком они были перечислены в запросе. Если вместо списка полей подставить знак звездочки (`*`), будут выбраны все поля.

После списка полей следует ключевое слово `FROM`, после которого ставится имя таблицы, из которой извлекаются данные. Завершается запрос SQL знаком точки с запятой.

Внимание

Некоторые СУБД, например, MySQL, требуют, чтобы каждый запрос SQL завершался точкой с запятой. Для других это необязательно. В любом случае, нужно читать документацию к каждому конкретному серверу данных.

Давайте рассмотрим несколько примеров запросов SQL для выборки данных из созданных нами таблиц — списков категорий и статей.

```
SELECT * FROM items;
```

Этот запрос вернет нам все записи таблицы `items` (список статей).

```
SELECT name FROM categories;
```

А этот запрос вернет только значения поля `name` таблицы `categories` (список категорий) вот в таком виде — см. рис. 6.7.

SELECT name FROM categories;

name
Интернет
Система
Офис
Программирование
Интернет

Рис. 6.7. Список значений поля `name` таблицы `categories`, возвращенных запросом SQL

Мы еще не рассмотрели ключевое слово `DISTINCT`, которое может присутствовать в запросе SQL. Если это слово указано, то возвращаются только уникальные строки. Например, запрос:

```
SELECT DISTINCT name FROM categories;
```

вернет результат, показанный на рис. 6.8. Видно, что вместо двух строк `Интернет` мы получили одну.

SELECT DISTINCT name FROM categories

name
Интернет
Система
Офис
Программирование

Рис. 6.8. Список уникальных значений поля `name` таблицы `categories`, возвращенных запросом SQL

Сортировка данных

Изначально запрос выборки данных SQL возвращает записи в том порядке, в котором они были добавлены в таблицу. Для задания порядка сортировки служит дополнительное ключевое слово `ORDER BY`, которое ставится в конец запроса:

. . . ORDER BY <*список критериев сортировки через запятую*>

А сами критерии сортировки имеют такой вид:

<*имя поля, по которому ведется сортировка*> [DESC]

Итак, поля, по которым должна вестись сортировка записей, перечисляются через запятую после ключевого слова `ORDER BY`, которое, в свою очередь, ставится в конце запроса перед знаком точки с запятой. При этом сервер данных будет сортировать записи по следующим правилам:

1. Сначала записи сортируются по полю, указанному первым в списке.
2. Если для некоторых записей значения этого поля одинаковы, то записи далее сортируются по полю, указанному вторым в списке.
3. Если для каких-то записей значения и этого поля одинаковы, то они будут отсортированы по полю, указанному третьим в списке.
4. И т. д.

По умолчанию записи сортируются так, чтобы значения поля выстроились по возрастанию. Если нужно отсортировать их по убыванию значений данного поля, нужно после имени этого поля поставить ключевое слово `DESC`.

Вот этот запрос выведет все записи таблицы `items` отсортированными по имени автора статьи:

```
SELECT * FROM items ORDER BY author;
```

А этот запрос сначала отсортирует записи таблицы `categories` по полю `file`, а потом — по полю `name`, причем по убыванию:

```
SELECT file, name FROM categories ORDER BY file, name DESC;
```

SELECT file, name FROM categories SORT BY file, name DES(

file	name
Нет	Система
Нет	Интернет
Да	Программирование
Да	Офис
Да	Интернет

Рис. 6.9. Отсортированный список значений полей `file` и `name` таблицы `categories`, возвращенных запросом SQL

В результате мы получим то, что показано на рис. 6.9. Обратим внимание — поля в результате приведены точно в таком порядке, в каком мы перечислили их в запросе.

Фильтрация данных

Для фильтрации записей по значениям полей используется ключевое слово `WHERE`. Это слово ставится между ключевыми словами `FROM` и `ORDER BY`:

```
. . . WHERE <список критериев фильтрации через запятую> . . .
```

А сами критерии фильтрации имеют вид обычных условий вида:

```
<имя поля> <оператор сравнения> <заданное значение>
```

Оператор сравнения — это особая команда SQL, задающая равенство или неравенство заданного значения и значения поля. Например, оператор `=` задает равенство:

```
id = 3
```

Здесь значение поля `id` сравнивается с заданным значением `3`. И если значение поля `id` равно `3`, то запись попадает в результат запроса.

Все доступные в стандарте SQL операторы сравнения перечислены в табл. 6.2.

Таблица 6.2. *Доступные в стандарте SQL операторы сравнения*

Оператор сравнения	Описание
`=`	Равно
`<>` или `!=`	Не равно
`<`	Меньше
`>`	Больше
`<=`	Меньше или равно
`>=`	Больше или равно

Таким образом, чтобы отобрать все статьи Ю. Криворукого, нам будет нужно написать такой запрос:

```
SELECT * FROM items WHERE author="Криворукий, Ю.";
```

Как видим, оператор `=` может использоваться также для сравнения двух строк. Соответственно, оператор `<>` может быть использован для проверки неравенства строк, вот так:

```
SELECT * FROM items WHERE author<>"Сусанин-мл., И.";
```

Этот запрос вернет нам все статьи, кроме тех, чьим автором является И. Сусанин-младший.

Внимание

Строковые величины, являющиеся частью критериев в запросах SQL, должны заключаться в кавычки!

Но что делать, если нам нужны статьи и Криворукого, и Сусанина-младшего? В этом случае используем особый *логический оператор* `OR`:

```
SELECT * FROM items WHERE author="Криворукий, Ю." OR
↳author="Сусанин-мл., И.";
```

Оператор `OR` говорит серверу данных, что должно выполняться ИЛИ первое, ИЛИ второе условие (собственно, "or" в переводе с английского — "или"). В этом случае запись попадет в результат.

Другой логический оператор — `AND` (в переводе с английского — "и") — требует, чтобы было выполнено И первое, И второе условие.

```
SELECT id FROM categories WHERE name="Интернет" AND file=true;
```

Приведенный ранее запрос вернет значение поля `id` записи таблицы `categories`, значение поля `name` которого равно `Интернет`, а значение логического поля `file` — `true` ("истина").

Последний логический оператор — `NOT` — означает по-английски "не". Например, запрос:

```
SELECT * FROM categories WHERE NOT id = 3;
```

вернет все записи, значение поля `id` которых НЕ равно `3`.

Логические операторы можно комбинировать. Рассмотрим, например, такой запрос:

```
SELECT * FROM items WHERE NOT (author="Криворукий, Ю." OR
↳author="Сусанин-мл., И.");
```

Он вернет все статьи, авторами которых не являются ни Криворукий, ни Сусанин-младший. Обратите внимание, что мы использовали скобки, предписывающие серверу данных выполнить сначала часть запроса

```
author="Криворукий, Ю." OR author="Сусанин-мл., И."
```

а уже потом применить к ней оператор `NOT`, так как он выполняется перед операторами `OR` и `AND` (как говорят профессиональные программисты, имеет бóльший *приоритет*). Если мы не поставим скобки:

```
SELECT * FROM items WHERE NOT author="Криворукий, Ю." OR
↳author="Сусанин-мл., И.";
```

то сервер данных будет трактовать наш запрос как "все статьи, авторами которых являются ИЛИ НЕ Криворукий, ИЛИ Сусанин-младший". То есть он сначала выполнит условие

```
author="Криворукий, Ю."
```

потом применит к нему оператор `NOT`, далее выполнит условие

```
author="Сусанин-мл., И."
```

и в конце применит к обоим этим условиям оператор `OR`. И результат этого запроса будет совсем другим.

К фильтрации записей мы еще вернемся. А пока продолжим изучение запросов SQL выборки данных.

Задание связей между таблицами

Чтобы связать две таблицы и получить из них данные, используется все то же ключевое слово `WHERE`. Только критерий фильтрации, записываемый после этого слова, имеет несколько другой вид. Какой — мы сейчас рассмотрим на примере.

Предположим, нам нужно получить названия авторов статей, а также названия категорий, в которые попадают эти статьи. Для этого нам будет нужно связать таблицы `items` (статьи) и `categories` (категории) так, как показано на рис. 6.6. Напишем такой запрос SQL:

```
SELECT items.author, items.name, categories.name FROM items, categories
↳WHERE items.catid=categories.id;
```

Нам будет проще разобрать его по частям, в порядке слева направо.

- После ключевого слова `SELECT`, как мы уже знаем, записывается список имен нужных нам полей. Здесь возникает небольшая проблема: обе таблицы — и `items`, и `categories` — имеют поле `name`, и, чтобы сервер данных знал, из какой таблицы брать это поле, мы запишем перед именем поля имя таблицы, в которой оно находится, разделив их точкой. Вот так: `items.name` или `categories.name`.

Внимание

Вообще, предварять имя поля именем таблицы, в которой оно находится, — хороший стиль написания запросов SQL. Будем так поступать в дальнейшем.

- После ключевого слова `FROM` записываются имена обеих таблиц, из которых мы получаем данные: `items` и `categories`. Эти имена нужно разделить запятой.
- А вот после ключевого слова `WHERE` и записывается *критерий связывания* таблиц. Он имеет такой же вид, как критерий фильтрации — `items.catid=categories.id` — и предписывает серверу данных для каждой записи таблицы `items` найти такую запись таблицы `categories`, чтобы значение ее поля `id` было равно значению поля `catid` записи таблицы `items`.

Выполнив этот запрос, мы получим результат, показанный на рис. 6.10.

SELECT items.author, items.name, categories.name FROM items, categories WHERE items.catid=categ

author	name	name
Криворукий, Ю.	Тяп-ляп Web-дизайнер	Интернет
Сусанин-мл., И.	Путеводитель по всемирной паутине	Интернет
СуПеРхАкЕр	Как взломать компьютер с помощью кувалды	Система
Дуболом, Е.	К вопросу о прочности модемов	Интернет

Рис. 6.10. Список значений полей `author` и `name` таблицы `items` и поля `name` таблицы `categories`

Стоп! У нас же в таблице `items` находятся и статьи, и файлы! А нам нужны только статьи. Давайте исправим эту ошибку, немного дополнив наш запрос (добавленный фрагмент кода SQL выделен полужирным шрифтом):

```
SELECT items.author, items.name, categories.name FROM items, categories
↳WHERE items.catid=categories.id AND categories.file=false
↳ORDER BY categories.name, items.name;
```

Здесь мы записали еще и критерий фильтрации, отбирающий только те записи таблицы `categories`, значение поля `file` которых равно `false` ("ложь"). Обратим внимание, что данный критерий связан с критерием связывания логическим оператором `AND`. Это нужно, чтобы действовали оба критерия: И связывания, И фильтрации.

Ну, и напоследок мы отсортировали отобранные записи сначала по названиям категорий (поле `categories.name`), а потом — по названиям статей (поле `items.name`).

Псевдонимы полей

Давайте еще раз посмотрим на полученный нами результат связывания таблиц (рис. 6.10). Что в нем не так?

О, ужас! У нас два поля имеют одно и то же имя — `name`. Если мы будем использовать этот результат в программе (а мы и будем использовать!), то могут возникнуть проблемы: как дать понять программе, какое поле нам нужно? Давайте что-то сделаем с этими одинаковыми именами!

Специально для таких случаев язык SQL предоставляет возможность дать полю другое, придуманное нами имя — так называемый *псевдоним поля*. Псевдоним создается с помощью ключевого слова `AS` таким образом:

```
SELECT . . . <имя поля> AS <псевдоним>, . . .
```

И записывается все это в списке полей, следующем за ключевым словом `SELECT`.

Что ж, яснее некуда. Так давайте еще раз перепишем наш предыдущий запрос. Он примет такой вид (добавленный и измененный код SQL выделен полужирным шрифтом):

```
SELECT items.author, items.name AS item_name, categories.name AS cat_name
↳FROM items, categories WHERE items.catid=categories.id AND
↳categories.file=false ORDER BY categories.name, items.name;
```

Здесь мы дали полю `items.name` псевдоним `item_name` (в именах полей таблиц недопустимы пробелы, поэтому мы использовали знак подчеркивания), а полю `categories.name` — псевдоним `cat_name`. Теперь каждое поле результата нашего запроса будет иметь уникальное имя.

Агрегатные функции SQL

Самая сложная и самая замечательная возможность языка SQL — это, конечно, *агрегатные функции*, позволяющие выполнять различные действия сразу над многими записями. Благодаря им мы может переложить часть работы по сбору статистики на сам сервер данных.

Предположим, нам нужно получить значения количества статей в каждой из категорий. Для этого нам придется воспользоваться одной из агрегатных функций, считающей количество записей. Но сначала нам будет нужно выполнить кое-какие предварительные действия, а именно — выполнить группировку записей по категориям.

Группировка — это объединение записей в группы по какому-либо критерию, называемому *критерием группировки*. Выполняется группировка с помощью ключевого слова `GROUP BY`, после которого записываются сами критерии группировки:

```
GROUP BY <имена полей, по которым будут группироваться записи, через
↳запятую>
```

Ставится ключевое слово `GROUP BY` вместе с критерием группировки перед ключевым словом `ORDER BY`.

Итак, нам нужно получить количество статей в каждой категории. Для этого нужно сначала сгруппировать записи по категориям (а именно по полю `categories.name`), в чем нам поможет вот такой запрос:

```
SELECT categories.name FROM items, categories WHERE
↳items.catid=categories.id GROUP BY categories.name
↳ORDER BY categories.name;
```

Отметим два важных момента, которые необходимо учитывать при составлении запроса с группировкой, иначе сервер данных не сможет его выполнить.

- Поля, по которым ведется группировка записей, должны быть первыми в списке полей ключевого слова `SELECT` и располагаться в том же порядке, в котором они перечислены после ключевого слова `GROUP BY`.

- Поля, по которым ведется группировка записей, должны быть первыми в списке полей ключевого слова `ORDER BY` и, опять же, располагаться в том же порядке, в котором они перечислены после ключевого слова `GROUP BY`.

Сгруппировав записи, мы можем выполнить подсчет количества записей в каждой группе, воспользовавшись агрегатной функцией `COUNT(<имя поля, по которому ведется подсчет>)`:

```
SELECT categories.name, COUNT(items.name) AS item_count
↳FROM items, categories WHERE items.catid=categories.id
↳GROUP BY categories.name ORDER BY categories.name;
```

Здесь мы подставили в качестве параметра (*аргумента*) функции имя поля `name` таблицы `items`, так как нам нужно считать записи именно этой таблицы. Вообще, можно было выбрать любое поле, например, `author` или `date` — в данном случае это непринципиально.

Осталось задать критерий отбора только категорий статей:

```
SELECT categories.name, COUNT(items.name) AS item_count
↳FROM items, categories WHERE items.catid=categories.id AND
↳categories.file=false GROUP BY categories.name ORDER BY categories.name;
```

Окончательный запрос вернет такой результат — см. рис. 6.11.

```
SELECT categories.name, COUNT(items.name) AS item_count
FROM items, categories WHERE items.catid=categories.id AND
categories.file=false GROUP BY categories.name ORDER BY
categories.name;
```

name	item_count
Интернет	3
Система	1

Рис. 6.11. Список значений поля `name` таблицы `categories` и количества статей в каждой категории

Заметим, что результат нашего запроса содержит только две записи, соответствующие категориям `Интернет` и `Система`. Где же остальные? А остальные сервер данных не вывел, так как в таблице `items` нет ни одной записи, которая бы на них ссылалась. Эту особенность группировки нужно иметь в виду.

Стандарт SQL определяет несколько агрегатных функций, которые мы можем использовать в своих запросах. Все эти функции перечислены в табл. 6.3. Все они применяются только к числовым полям, кроме уже знакомой нам функции `COUNT`.

Таблица 6.3. *Агрегатные функции, доступные в языке SQL*

Агрегатная функция	Описание
COUNT(<*поле*>)	Количество записей
SUM(<*поле*>)	Сумма значений поля всех записей группы
AVG(<*поле*>)	Среднее значение поля всех записей группы
MIN(<*поле*>)	Минимальное из значений поля во всех записях группы
MAX(<*поле*>)	Максимальное из значений поля во всех записях группы

Вот, собственно, и все о запросах SQL, реализующих выборку данных. Мы узнали то, что пригодится нам на первых порах. Полностью же язык SQL описан в документации, поставляемой с сервером данных.

Изменение записей таблицы

Теперь обратимся к запросам изменения данных: добавления, изменения и удаления записи.

Добавление записи

Запрос SQL добавления записи в таблицу — самый простой из запросов изменения данных. Он строится с помощью ключевого слова INSERT INTO:

```
INSERT INTO <имя таблицы> (<имена полей, разделенные запятыми>) VALUES
↳(<значения полей, разделенные запятыми>);
```

Здесь после ключевого слова INSERT INTO ставится имя таблицы, в которую добавляется новая запись. Далее в скобках перечисляются имена полей новой записи, в которые должны быть помещены значения. Сами эти значения перечисляются в скобках, в том же порядке, что и поля (это важно!), после ключевого слова VALUES.

Вот пример добавления новой записи в таблицу items:

```
INSERT INTO items (name, author) VALUES ("Программы, которые не делают
↳ничего", "Лобо-Тряс, Ф.");
```

Изменение записи

А вот запрос изменения записи из всех запросов изменения данных — самый сложный. Он строится с помощью ключевого слова UPDATE:

```
UPDATE <имя таблицы> SET <имя 1-го поля>=<новое значение 1-го поля>,
↳<имя 2-го поля>=<новое значение 2-го поля>... WHERE <критерий
↳фильтрации, необходимый для нахождения изменяемой записи>;
```

После ключевого слова `UPDATE` указывается имя таблицы, чью запись нужно изменить. Затем идет ключевое слово `SET`, а за ним — набор пар вида "*имя поля=его новое значение*", разделенных запятыми. Последним идет уже знакомое нам по запросам выборки данных ключевое слово `WHERE`. И вот тут начинается самое интересное.

Чтобы изменить какую-либо запись, сервер данных должен ее найти. А чтобы ее найти, нам придется задать после ключевого слова `WHERE` такой критерий фильтрации, чтобы он однозначно указывал на нужную нам запись. Такой критерий можно задать двумя способами.

Во-первых, мы можем найти запись по старым значениям всех ее полей. Вот пример запроса, использующего такой поиск:

```
UPDATE items SET name="Web-ломастер" WHERE name="Тяп-ляп Web-дизайнер"
↳AND author="Криворукий, Ю." AND date=(12.08.2004);
```

Видно, что такой запрос очень громоздок, к тому же, он очень медленно выполняется, так как серверу данных придется выполнить поиск по всем полям таблицы. Но, если таблица не содержит поля, однозначно идентифицирующего запись, это единственный способ построить запрос изменения записи.

Замечание

Значения даты в запросах SQL могут задаваться по-разному, в зависимости от конкретного сервера данных. Мы выбрали способ, когда значение даты заключается в скобки.

Во-вторых, мы можем использовать поиск записи по полю, однозначно ее идентифицирующему. Такое поле может иметь, например, тип счетчика (как наша таблица `categories`).

```
UPDATE categories SET name="Internet" WHERE id=1;
```

Такой запрос значительно компактнее из-за того, что используется поиск только по одному полю — `id`. Вдобавок, поскольку на основе этого поля создан ключевой индекс, поиск выполняется очень быстро.

Отсюда следует один очень важный вывод. Каждая таблица в серверной базе данных, по возможности, должна иметь поле, однозначно идентифицирующее каждую ее запись. Это позволит сократить размер запросов SQL и ускорить поиск.

Сделаем себе пометку на память — добавить в таблицу `items` поле `id` типа счетчика.

Удаление записи

Запрос удаления записи также очень прост. Он создается на основе ключевого слова `DELETE FROM`:

```
DELETE FROM <имя таблицы> WHERE <критерий фильтрации,
↳необходимый для нахождения удаляемой записи>;
```

Здесь все то же самое, что и в запросе изменения записи: имя таблицы, из которой удаляется запись, и критерий фильтрации для ее нахождения. Вот пример запроса, удаляющий запись из таблицы `categories`:

```
DELETE FROM categories WHERE id=3;
```

Другие запросы SQL

Мы только что рассмотрели некоторые запросы SQL, предназначенные для выборки и изменения данных. Но этим возможности языка SQL не ограничиваются. В его составе есть запросы, выполняющие другие действия над данными; мы рассмотрим их потом.

Вообще, запросы SQL можно разделить на три группы.

- *Запросы управления данными.* Сюда входят все рассмотренные нами запросы выборки данных, добавления, изменения и удаления записей.
- *Запросы определения данных.* Это запросы создания, изменения и удаления баз данных, таблиц, индексов, связей и пр. Мы не будем рассматривать их, так как не будем сами создавать эти запросы.
- *Служебные запросы.* Выполняют различные технические задачи: сбор статистики использования баз данных, резервное копирование и пр. Некоторые из служебных запросов мы рассмотрим далее в этой книге.

На этом мы пока закончим рассмотрение языка SQL. Разумеется, мы не узнали обо всех его возможностях: их слишком много, вдобавок, далеко не все из них нам пригодятся на первое время. Кое-что мы дополнительно изучим потом, когда будем писать серверные программы. А полное описание языка SQL со всеми его особенностями содержится в документации программ — серверов данных.

Разграничение доступа. Права

Теперь нам нужно рассмотреть еще один важный момент, касающийся серверов данных (да и вообще всех серверных программ). Это так называемое разграничение доступа: система зарегистрированных в серверной программе пользователей, их имен и паролей, а также прав, предоставляемых пользователям на выполнение различных операций.

Нам уже известно, что каждая программа-сервер, будь то FTP-сервер, Web-сервер или сервер электронной почты, имеет список пользователей, которые могут с ней работать. (С этим мы столкнулись, когда загружали свой Web-сайт на бесплатный Web-сервер — нам пришлось регистрироваться, а потом вводить свое имя и пароль.) Как правило, пользователи, не указанные в этом списке, вообще не могут подключиться к серверу.

Правда, из этого правила есть исключение. Многие серверные программы, те же самые Web- и FTP-серверы, предоставляют возможность так называемого *анонимного доступа*, когда не указанный в списках пользователь может подключиться к серверу и работать с ним. В частности, все Web-серверы поддерживают анонимный доступ, иначе нам бы для входа на любой сайт пришлось бы сначала регистрироваться на нем, а потом вводить свои имя и пароль. Конечно же, права незарегистрированного пользователя в этом случае очень ограничены — в основном, только просмотр.

Почтовые серверы и серверы данных, наоборот, предоставляют только *авторизованный доступ*, с регистрацией в списке пользователей и вводом имени и пароля. Сделано это для того, чтобы важные данные не попали в чужие руки. Вот об авторизованном доступе и о реализации его в серверах данных мы и поведем дальнейший разговор.

Итак, каждый сервер данных (как и любой другой сервер, поддерживающий авторизованный доступ) имеет список зарегистрированных пользователей, которым разрешено подключаться к базам данных и пользоваться ими. Этот список ведет человек, обслуживающий сервер данных, — администратор сервера: добавляет новых пользователей, удаляет старых, изменяет сведения о пользователях, если это нужно. При попытке открыть любую базу данных сервер запрашивает у пользователя его имя и пароль и, получив их, проверяет, имеет ли этот пользователь право на доступ к данной базе. Если имеет, сервер разрешает ему доступ к базе, если нет — возвращает сообщение об ошибке подключения.

Однако здесь не все так просто. Дело в том, что, кроме имени и пароля пользователя, сервер данных хранит также сведения о том, к каким именно базам данных и таблицам имеет доступ пользователь и какие операции над ними он может проделывать. Эти сведения называются *правами*.

Права, поддерживаемые серверами данных, можно разделить на четыре группы.

- Права на выборку данных из таблиц базы (так называемые *права на чтение*).
- Права на изменение данных — добавление, изменение и удаление записей (*права на запись*).
- Права на создание баз данных, таблиц, индексов, связей и пр. (*права администратора баз данных*).

- Права на выполнение служебных операций — получение статистики об использовании баз данных, резервное копирование и пр. (*права администратора сервера*).

Права из разных групп могут назначаться отдельно. Так, можно дать пользователю `user4257` права только на запись, но не давать права на чтение.

Права могут назначаться отдельно на:

- все базы данных сервера;
- базу данных;
- таблицу;
- поле таблицы.

Например, тот же пользователь с именем `user4257` может иметь права на чтение для таблицы `categories`, права на запись и чтение для таблицы `items` и права только на запись для поля `items.date`.

Все это понадобится нам, когда мы будем создавать базу данных для своего нового сайта. А сейчас давайте закончим разговор о серверных базах данных вообще и познакомимся поближе с конкретным сервером данных под названием MySQL.

Сервер данных MySQL и его возможности

MySQL — это популярный сервер данных, применяемый при создании Web-сайтов. Используем его и мы. Почему?

- MySQL — весьма быстрый и нетребовательный к ресурсам компьютера сервер данных.
- Возможностей MySQL с лихвой хватает для создания Web-сайтов. Конечно, конкурирующие серверы значительно мощнее, и разработчики больших Web-порталов используют именно их, но для мелких сайтов вполне подойдет MySQL.
- MySQL распространяется бесплатно, более того — его исходные тексты открыты для изучения и доработки.
- И последнее — MySQL прекрасно работает в связке с PHP, технологии создания активных серверных Web-страниц, которую мы выбрали для создания нашего сайта в *главе 5*.

Чтобы проверить свой сайт, нам придется установить MySQL на нашем компьютере. Как это сделать, описано в *приложении 2* этой книги.

Сначала давайте коротко "пробежимся" по возможностям MySQL, чтобы знать, что он может, а чего не может. И заодно "увяжем" его возможности с тем, что уже изучили в этой главе.

В принципе, возможности MySQL аналогичны возможностям других серверов данных. MySQL поддерживает запросы SQL, одновременный доступ нескольких пользователей к базам данных, индексы, права, множество типов данных и пр. Этого нам хватит для создания нашего сайта.

Некоторые из поддерживаемых MySQL типов данных перечислены в табл. 6.4. Их довольно много (а полный список поддерживаемых типов данных еще больше).

Таблица 6.4. *Некоторые типы данных, поддерживаемые MySQL*

Тип данных	Обозначение в MySQL	Примечание
Строковый	VARCHAR	Строки длиной от 1 до 255 символов. Длина строки задается при создании поля
Целочисленный	SMALLINT	Целые числа от –32768 до 32767 или от 0 до 65535
	MEDIUMINT	Целые числа от –8388608 до 8388607 или от 0 до 16777215
	INT	Целые числа от –2147483648 до 2147483647 или от 0 до 4294967295
	BIGINT	Целые числа от –9223372036854775808 до 9223372036854775807 или от 0 до 18446744073709551615
С плавающей точкой	FLOAT	Число с плавающей точкой от $-3{,}402823466*10^{38}$ до $-1{,}175494351*10^{-38}$, 0 и от $1{,}175494351*10^{-38}$ до $3{,}402823466*10^{38}$
	DOUBLE	Число с плавающей точкой от $-1{,}7976931348623157*10^{308}$ до $-2{,}2250738585072014*10^{-308}$, 0 и от $2{,}2250738585072014*10^{-308}$ до $1{,}7976931348623157*10^{308}$
Логический	BOOL	Логическая величина "истина" (true) или "ложь" (false)
Дата	DATE	Значения даты от 01.01.1000 до 31.12.9999
Дата и время	DATETIME	Объединенные значения даты от 01.01.1000 00:00:00 до 31.12.9999 23:59:59
Memo	BLOB	Строки переменной длины. Могут вмещать до 65535 символов
	MEDIUMBLOB	Строки переменной длины. Могут вмещать до 16777215 символов
	LONGBLOB	Строки переменной длины. Могут вмещать до 4294967295 символов

Кроме типа данных, поля имеют и другие дополнительные параметры, называемые *атрибутами полей*. Так, атрибут `UNSIGNED` запрещает числовым полям принимать отрицательные значения. Атрибут `AUTO_INCREMENT` превращает обычное целочисленное поле в поле счетчика (отдельный тип данных счетчик MySQL не поддерживает). А атрибут `DEFAULT` позволяет задать для поля значение по умолчанию, то есть фактически задает правило для поля.

Здесь нужно сказать еще вот что. При создании новой записи всем полям, которым не были присвоены значения (неважно — запросом добавления записи или значение по умолчанию, заданное атрибутом `DEFAULT`), присваивается особое значение `NULL`. Оно означает, что поле вообще не содержит никаких данных. Если же необходимо, чтобы это поле обязательно содержало значение, то есть потребовать от пользователя всегда вводить значение в это поле, нужно дать ему атрибут `NOT NULL`.

Набор разных прав, поддерживаемых MySQL, просто впечатляет — можно давать отдельные права на выполнение разных видов запросов SQL. Так, чтобы дать пользователю возможность извлекать данные из таблицы с помощью запроса `SELECT`, нужно использовать *атрибут прав*, который так и называется — `SELECT`. Соответственно, права на добавление записей дает атрибут `INSERT`, на изменение — `UPDATE`, а на удаление — `DELETE`. Также можно дать права на создание, изменение и удаление таблиц и индексов, выполнение служебных операций и пр. Так что с разграничением доступа у MySQL все в порядке.

Кроме имени пользователя и пароля, MySQL позволяет также задать интернет-адрес компьютера, с которого данный пользователь может подключаться к серверу. Фактически интернет-адрес компьютера в MySQL является частью имени пользователя, которое в этом случае записывается вот так:

```
<имя пользователя>@<интернет-адрес компьютера>
```

то есть как адрес электронной почты. Например:

```
root@localhost
```

Пользователь `root` имеет право подключаться к серверу данных только с локального компьютера (интернет-адрес `localhost`). Забегая вперед, скажем, что пользователь `root` обычно является администратором сервера и, соответственно, имеет максимальные права.

```
remote_user@dev.domain.ru
```

А пользователь `remote_user` может подключиться к серверу только с компьютера `dev.domain.ru` и ни с какого другого (даже локального).

Если же нужно дать пользователю возможность подключаться с любого компьютера, нужно будет вместо интернет-адреса подставить шаблон `%`, задающий любой интернет-адрес. Например:

```
travelling_user@%
```

Пользователь `travelling_user` может подключаться к серверу с любого компьютера — и локального, и удаленного.

Шаблон `%` можно использовать и вместо имени пользователя; тогда он будет задавать любого пользователя. Так, если написать

```
%@localhost
```

то с локального компьютера к серверу сможет подключиться любой пользователь (в смысле, с любым именем, даже если оно явно не записано в списке пользователей). А если записать

```
%@%
```

то к серверу сможет подключиться любой пользователь с любого компьютера. Конечно, такому пользователю будет нужно дать минимальные права, иначе он сможет творить с данными все что пожелает.

Полностью все возможности MySQL описаны в электронной документации, которую можно загрузить с сайта разработчиков MySQL — **http://www.mysql.com**. В том числе, доступна документация и на русском языке.

А мы приступим к созданию базы данных для нашего сайта. (Подразумевается, что мы уже установили MySQL и все сопутствующие программы. Если же нет, то процессы установки и настройки MySQL и вспомогательных программ описаны, соответственно, в *приложениях 2* и *4* этой книги.)

Создаем базу данных для нашего сайта

Собственно, базу данных для своего сайта мы уже практически создали. Всего нам понадобятся две таблицы: список категорий `categories` и список статей и файлов `items`. Список категорий изображен на рис. 6.4, а список статей и файлов — на рис. 6.1. Нам осталось только внести в них некоторые изменения.

Первое, что мы сделаем, — это добавим новое поле в таблицу списка статей и файлов `items`. Назовем это поле `id` и дадим ему тип данных счетчик. Это поле позволит нам точно указать на нужную нам запись таблицы, что очень пригодится при изменении и удалении записей.

Далее посмотрим на поле `date`, точнее, на имя этого поля. Оно совпадает с ключевым словом `DATE`, которое используется для задания типа данных даты в запросе SQL создания таблицы. А по правилам SQL, имена полей, таблиц и индексов не должны совпадать с ключевыми словами. Поэтому давайте изменим имя этого поля на `added`.

Окончательная структура таблицы `items` с учетом этих исправлений показана в табл. 6.5. Также там для полей приведены реальные типы данных, поддерживаемые MySQL — так будет проще создавать таблицу.

Таблица 6.5. *Структура таблицы* `items`

Имя поля	Описание поля	Тип данных поля	Описание типа данных
`id`	Идентификатор записи	`SMALLINT UNSIGNED`	Целые числа от 0 до 65535. Статей и файлов у нас будет немного, так что этого должно хватить
`author`	Автор статьи или файла	`VARCHAR(30)`	Строка фиксированной длины 30 символов
`name`	Название статьи или файла	`VARCHAR(60)`	Строка фиксированной длины 60 символов
`added`	Дата добавления статьи или файла в список	`DATE`	Дата
`href`	Интернет-адрес файла или статьи	`VARCHAR(255)`	Строка фиксированной длины 255 символов — именно такова максимальная длина интернет-адреса
`catid`	Ссылка на категорию (значение поля `id` соответствующей записи таблицы `categories`)	`SMALLINT UNSIGNED`	Целые числа от 0 до 65535

На основе поля `id` этой таблицы создадим ключевой индекс. Также создадим еще два индекса: один — на основе поля `added`, другой — на основе поля `catid`. Назовем их, соответственно, `added` и `catid`. Эти индексы помогут нам ускорить сортировку по дате и поиск записей, относящихся к данной категории.

К сожалению, MySQL может использовать в качестве значения по умолчанию для записи только конкретные числа, строки или значения даты. Так что заносить значение в поле `added` при добавлении записи придется нашей будущей серверной программе.

Изменений в таблице `categories` не будет вообще — там нечего менять. Ее структура приведена в табл. 6.6.

Здесь мы, опять же, создадим ключевой индекс на основе поля `id`. Второй индекс мы создадим уже на основе двух остальных полей: `name` и `file`. Дадим ему имя `namefile`.

А теперь — внимание! Мы сделаем индекс `namefile` *уникальным*, присвоив ему особый *атрибут индекса* `UNIQUE`. Уникальный индекс может содержать только уникальные, не повторяющие в его пределах значения. Если же пользователь попытается ввести в поля записи значения, нарушающие уникальность индекса, сервер данных оповестит его об ошибке ввода и откажется заносить введенные данные в базу.

Таблица 6.6. *Структура таблицы* `categories`

Имя поля	Описание поля	Тип данных поля	Описание типа данных
`id`	Идентификатор записи	`INT UNSIGNED`	Целые числа от 0 до 65535. Категорий у нас будет немного, так что этого должно хватить
`name`	Название категории	`VARCHAR(20)`	Строка фиксированной длины 20 символов
`file`	Если "истина", то запись обозначает категорию файлов, в противном случае — категорию статей	`BOOL`	Логические величины

Осталось придумать имя для самой базы данных. Не будем ломать голову и назовем ее просто — `site`.

Теперь осталось только запустить сервер MySQL и, пользуясь одним из поддерживающих его клиентов данных, например MySQL Control Center, создать новую базу данных, таблицы и индексы.

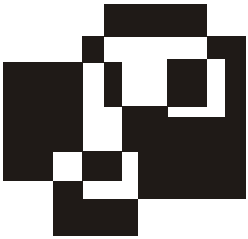
База данных готова? Отнюдь. Еще нужно добавить в список пользователей сервера MySQL нового пользователя, от имени которого наша будущая серверная программа будет работать с только что созданной базой данных. Назовем его так же, как и базу, — `site`. И дадим ему права на чтение и изменение данных в обеих таблицах базы. Вот теперь действительно можно праздновать рождение нашей первой базы данных.

И еще вот что. Давайте введем в таблицы свежесозданной базы данных несколько записей. Они пригодятся нам, когда мы начнем создавать наши первые серверные Web-страницы, — мы сможем сразу увидеть, правильно ли они работают или нет. Ведь впоследствии мы всегда можем эти записи удалить, не так ли?

Что дальше?

В этой главе мы выяснили, что такое базы данных и как с ними работать. Мы узнали о программах серверов данных, о языке SQL и даже составили несколько простейших запросов. Также мы узнали о правах и разграничении доступа на основе этих прав — они помогут нам защитить наши драгоценные данные от непрошеных глаз и рук.

Но одной базой данных нам не обойтись. Нужна еще программа, которая будет их обрабатывать и выводить пользователям — посетителям нашего сайта. А чтобы написать эту программу, нужно знать подходящий язык программирования. Поэтому в следующей главе мы начнем изучать технологию PHP и принципы написания активных серверных Web-страниц. До создания реальных, работающих серверных программ еще далеко, поэтому наберемся терпения.



Глава 7

PHP — технология написания серверных приложений

Данные без обрабатывающей их программы — ничто, мертвый капитал. Так что наш следующий шаг — написание серверной программы, которая будет работать с созданной нами в *главе 6* базой данных.

Еще раньше, в *главе 5*, мы решили, что наша серверная программа будет представлять собой набор активных серверных страниц — обычных Web-страниц, содержащих фрагменты программного кода, обрабатывающего эти данные. Встретив такой фрагмент, Web-сервер передает его обработчику серверных страниц, получает от него результат в виде обычного кода HTML и вставляет его точно в то место, где находится данный фрагмент программного кода.

Тогда же мы выбрали технологию для создания активных серверных страниц нашего сайта — PHP. Она дает достаточно возможностей для создания сложных сайтов, весьма производительна, бесплатна (даже ее исходные тексты открыты!) и, вдобавок, замечательно работает в одной "связке" с сервером данных MySQL. (Вообще-то, PHP может работать с очень многими серверами данных и даже поддерживает ODBC.) Другие же технологии, имеющиеся на рынке, либо платны, либо привязаны к какому-то одному Web-серверу.

Хорошо! Выбор сделан. Так чем же мы сейчас займемся?

Основные понятия PHP

А начнем мы с того, что изучим основные понятия и принципы работы с PHP. И заодно напишем несколько простейших серверных страничек, чтобы попрактиковаться и заодно проверить, как работает наш Web-сервер.

Но сначала нам нужно установить на свой компьютер сами программные модули PHP и настроить установленный ранее Web-сервер на поддержку данной технологии. Как это сделать, описано в *приложении 3*.

Написание сценариев PHP

Фрагменты программного кода (так называемые *сценарии*) PHP записываются прямо в коде HTML Web-страниц. При этом они помещаются внутрь особого парного тега `<?php...?>`.

Давайте наберем в Блокноте вот такой HTML-код, содержащий небольшой сценарий PHP (выделен полужирным шрифтом):

```
<HTML>
  <HEAD>
    <TITLE>Проба силы в PHP</TITLE>
  </HEAD>
  <BODY>
    <?php echo "<P>Привет!</P>"; ?>
  </BODY>
</HTML>
```

Оператор вывода `echo` выводит строковое значение, указанное после него, в то место страницы, где сам находится. Что касается самого строкового значения, то оно заключено в двойные кавычки (это важно!) и содержит HTML-код обычного абзаца, содержащего слово `Привет!`.

Сохраним этот код в файле 7.1.php (все активные серверные страницы PHP должны обязательно иметь расширение php, иначе Web-сервер примет их за обычные Web-страницы и просто отправит Web-обозревателю). После этого скопируем этот файл в корневую папку сайта нашего локального Web-сервера и запустим сам Web-сервер. Теперь, если мы откроем Web-обозреватель и наберем в строке адреса **http://localhost:8080/7.1.php**, увидим то, что показано на рис. 7.1.

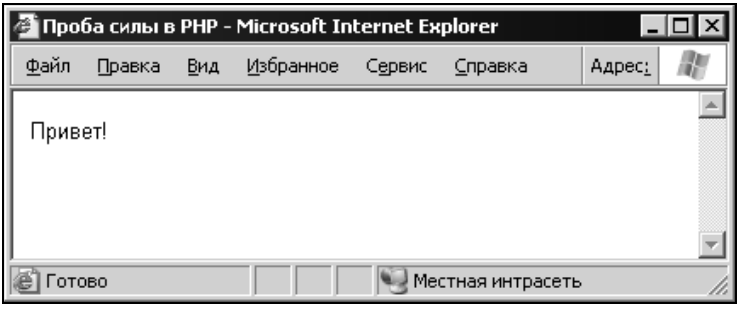


Рис. 7.1. Результат открытия серверной страницы 7.1.php в Web-обозревателе

Внимание

Сказанное ранее справедливо только для того случая, если мы настроили наш Web-сервер на порт 8080 (как описано в *приложении 1*). Если же Web-сервер настроен на другой порт, его номер нужно будет подставить в приведенный ранее интернет-адрес вместо 8080.

В дальнейшем мы всегда будем предполагать, что Web-сервер настроен на порт 8080.

Давайте теперь выведем на экран HTML-код окончательной Web-страницы, сформированной обработчиком PHP. (В Microsoft Internet Explorer это можно сделать, выбрав пункт **Просмотр HTML-кода** (View HTML code) в меню **Вид** (View).) Мы увидим следующее:

```
<HTML>
  <HEAD>
    <TITLE>Проба силы в PHP</TITLE>
  </HEAD>
  <BODY>
    <P>Привет!</P>
  </BODY>
</HTML>
```

То есть вместо сценария обработчик PHP вставил результат его выполнения. Значит, все работает!

Мы можем написать код нашей первой PHP-страницы немного по-другому:

```
<HTML>
  <HEAD>
    <TITLE>Проба силы в PHP</TITLE>
  </HEAD>
  <BODY>
    <P><?php echo "Привет!"; ?></P>
  </BODY>
</HTML>
```

Здесь мы "вынесли" сами теги `<P>` и `</P>` из строкового значения в сценарии. Посмотрим, справится ли с этим обработчик PHP. Копируем новую страницу в корневую папку сервера, открываем в Web-обозревателе. Результат показан на рис. 7.1.

Более того, мы даже можем написать вот такой код:

```
<P>Привет, <?php echo "мир!"; ?></P>
```

или даже такой:

```
<?php echo "<P>"; ?><?php echo "Привет, "; ?><?php echo "мир!"; ?>
↳<?php echo "</P>"; ?>
```

и обработчик PHP все равно с ним справится.

Если код сценария PHP занимает больше одной строки, то записывать его лучше всего вот так:

```
<?php
  echo "<P>";
  echo "Привет, ";
  echo "мир!";
  echo "</P>";
?>
```

то есть внутри одного тега `<?php...?>`. Так мы сделаем код нашей серверной страницы более компактным и легким для чтения.

Нужно еще сказать, что активные серверные Web-страницы PHP могут вообще не содержать кода HTML — только сценарии, точнее, единственный сценарий PHP. Такое встречается, и нередко. Но даже в подобном "вырожденном" случае сценарий нужно помещать внутрь тега `<?php...?>`, иначе Web-сервер не поймет, что с ним делать.

Операторы, аргументы и выражения

Теперь давайте поближе рассмотрим небольшой сценарий PHP, только что написанный нами. Уберем весь HTML-код, чтобы он нам не мешал, и сосредоточимся на коде PHP.

```
echo "<P>Привет!</P>";
```

Итак, здесь мы видим уже знакомый нам оператор вывода `echo`. Он принимает один аргумент — строку — и выводит ее на экран, точнее, в то место кода HTML, где встретился. (В принципе, это одно и то же, так как сформированная серверной программой Web-страница все равно попадет на экран посетителю сайта.)

Пора дать определения оператора и аргумента. *Оператор* — это команда языка PHP, выполняющая какое-либо действие над переданными ему данными (*аргументом*) или самим сценарием.

Операторы в PHP бывают разные. Мы уже знакомы с оператором вывода данных `echo`. Существуют также *арифметические* операторы, выполняющие элементарные действия над числами: сложение, вычитание, умножение и деление. Вот пример сценария, использующего арифметические операторы:

```
echo 2 + 2;
```

Здесь мы видим оператор сложения +, который принимает два аргумента — числовые значения 2 и 2. Сложив их, он возвращает полученную сумму (*результат*), который, в свою очередь, принимает оператор `echo` в качестве аргумента. Заметим, что числовые значения не берутся в кавычки.

Приведенный ранее сценарий PHP состоит из одного выражения. *Выражение* — это неразрывный фрагмент кода PHP, выполняющий одно законченное действие. Так, выражение

```
echo 2 + 2;
```

выполняет одно законченное действие: складывает два числа и выводит сумму на экран. К тому же, оно неразрывно, то есть не дробится на более мелкие части.

Вот этот сценарий состоит из четырех выражений:

```
$a = 2;
$b = 3;
$c = $a + $b;
echo $c;
```

Каждое выражение обязательно должно завершаться знаком точки с запятой. Точка с запятой — это знак конца выражения; встретив его, обработчик PHP считает, что выражение завершено и его нужно выполнить.

Выражения — это своего рода молекулы, из которых собирается сценарий. Операторы и их аргументы (а также функции, которые мы рассмотрим потом) — суть атомы, стандартные элементы языка PHP, из которых мы собираем наши выражения. Своего рода программистская физика.

Осталось сказать, что для представления операторов язык PHP так же, как и SQL, использует особые зарезервированные слова, называемые ключевыми. Причем регистр, в котором набраны буквы ключевых слов, значения не имеет. Так, можно записать `echo`, `Echo` или `ECHO` — и это будет один и тот же оператор вывода.

Переменные

Давайте посмотрим еще раз на сценарий из трех выражений, приведенный чуть выше. Что он, вообще, делает? И что это за латинские буквы, перед которым стоит знак доллара?

Дело в том, что язык PHP предоставляет в распоряжение программиста так называемые переменные. *Переменная* — это участок памяти компьютера, отведенный программисту для хранения каких-то данных: аргументов или результатов выполнения операторов. Программист может писать выражения, помещающие какие-либо данные в переменные или извлекающие их оттуда.

Каждая переменная должна иметь уникальное имя. Это имя должно всегда начинаться символом доллара и содержать только буквы латинского алфавита, цифры и знаки подчеркивания. Причем следующим после знака доллара символом обязательно должна быть буква или знак подчеркивания. Также имя переменной не должно совпадать с ключевыми словами PHP. Длина имени переменной не ограничена, но лучше делать их как можно короче и как можно понятнее.

Вот несколько примеров правильно написанных имен переменных:

```
$var
$extended_result2
$_temp
```

А это — неправильные имена:

```
$2result
$extended output
$расширенный вывод
```

В первом из этих имен после знака доллара идет цифра, второе содержит пробел, а третье набрано русскими буквами.

В отличие от ключевых слов, имена переменных в PHP чувствительны к регистру. Так, `$var` и `$Var` — это разные переменные.

А теперь снова давайте обратимся к нашему "загадочному" сценарию. Рассмотрим его по частям.

```
$a = 2;
$b = 3;
```

Здесь мы присваиваем числовые значения двум переменным: `$a` и `$b`. Делается это с помощью особого *оператора простого присваивания* `=`. Слева от него записывается имя переменной, а справа — значение, которое должно быть ей присвоено.

Поскольку переменных `$a` и `$b` еще не существует, обработчик PHP создаст или, как говорят программисты, объявит их. *Объявление* переменной происходит при первом присвоении ей значения.

```
$c = $a + $b;
```

Здесь мы объявляем третью переменную — `$c` — и присваиваем ей результат вычисления выражения `2 + 3`.

```
echo $c;
```

А в этом, последнем, выражении нашего сценария оператор `echo` извлекает значение переменной `$c` (это будет сумма 2 и 3 — 5) и выводит его на экран.

Осталось сказать, что переменная, объявленная в каком-либо сценарии серверной страницы PHP, доступна во всех сценариях, находящихся в этой

странице. Так, если мы напишем два сценария в коде Web-страницы (сам HTML-код опущен):

```
. . .
<?php $a = "Test!!!"; ?>
. . .
<?php echo $a; ?>
```

то они оба выполнятся правильно.

Типы данных

В отличие от полей таблиц, изученных нами в *главе 6*, переменные PHP могут содержать данные различных типов. Можно, например, объявить переменную, присвоив ей число, а потом ей же присвоить строковое или логическое значение, и обработчик все выполнит. Хотя, конечно, такого лучше не допускать — это плохой стиль программирования.

Давайте получше познакомимся с типами данных, поддерживаемыми PHP. Их не так много.

Логический

Логический тип представляет только два значения: "истина" (true) и "ложь" (false). Оба эти значения записываются ключевыми словами языка PHP `true` и `false`.

```
$flag = true;
```

Целочисленный

Целочисленный тип представляет, как ясно из названия, целые числа от −2 147 483 648 до 2 147 483 647. В отличие от типов полей в таблицах баз данных, он всего один.

```
$counter = 10;
$delta = -193;
```

По умолчанию целые числа задаются в десятичной системе счисления. Если нужно задать число в восьмеричной или шестнадцатеричной системе счисления, достаточно предварить его знаком `0` и `0x` соответственно.

```
$octal_number = 0123;
$hex_number = 0x4F;
```

С плавающей точкой

Тип с плавающей точкой представляет дробные числа от $-1{,}79769313486232*10^{308}$ до $-2{,}225073858507 2*10^{-308}$, 0 и

от $2{,}2250738585072*10^{-308}$ до $1{,}79769313486232*10^{308}$ с точностью примерно до 14-го знака после запятой.

```
$square = 10.56;
```

Заметим, что вместо знака запятой для представления числа в коде PHP используется точка.

Для представления чисел в нормализованном виде (<*мантисса*>*10<*порядок*>) можно использовать специальный синтаксис вида `<мантисса>E<порядок>`.

```
$distance = 2.648E+12;
```

В таком виде представлено число $2{,}648*10^{12}$. Обратите внимание, что знак + в значении порядка необходим.

```
$millimeter = 1E-3;
```

А так представлено число $1*10^{-3}$.

Строковый

Строковый тип представляет строки текста практически неограниченного размера (по крайней мере, так написано в документации по PHP). Строковые значения должны быть взяты в двойные кавычки. Внутри строк можно использовать любые символы, которые могут быть выведены на экран.

```
$output = "Привет!";
```

PHP также предоставляет несколько так называемых *специальных символов*, которые могут быть использованы в строках. Эти символы либо выполняют особые действия, либо не могут быть включены в строки обычным образом. Все они перечислены в табл. 7.1.

Таблица 7.1. *Специальные символы PHP*

Специальный символ	Описание
`\n`	Перевод строки
`\r`	Возврат каретки
`\t`	Горизонтальная табуляция
`\\`	Обратный слеш (\)
`\$`	Знак доллара
`\"`	Двойная кавычка
`\<код>`	Символ с заданным восьмеричным кодом
`\x<код>`	Символ с заданным шестнадцатеричным кодом

Здесь нужно дать некоторые пояснения.

Символы обратного слеша, доллара и двойной кавычки не присутствуют в строках, поэтому вместо них должны использоваться соответствующие им специальные символы. Например, это неправильное выражение:

```
$output = "Гостиница "Волжская"";
```

так как внутри строки не должны присутствовать двойные кавычки. Нужно использовать вместо них последовательность символов `\"`, вот так:

```
$output = "Гостиница \"Волжская\"";
```

то есть предварять каждый недопустимый в строковых значениях символ знаком "обратный слеш" `\`.

Знаки возврата каретки и перевода строки, следующие друг за другом, — `\r\n` — позволят начать вывод данных с помощью оператора `echo` с новой строки. Например, сценарий:

```
echo "Начнем";
echo "вывод\r\n";
echo "с новой строки!!!";
```

выведет на экран вот что:

```
Начнем вывод
с новой строки!!!
```

Обратим внимание, что первое и второе выражение этого сценария выполнили вывод в одну строку.

Замечание

Иногда строковые значения берутся в одинарные кавычки. Однако для таких строк не доступны некоторые возможности PHP.

NULL

Тип NULL означает, что переменная не содержит никакого значения. Это может случиться, если мы пытаемся получить значение переменной, которая еще не объявлена. Также мы можем прямо присвоить переменной NULL, воспользовавшись ключевым словом `NULL`.

```
$null_value = NULL;
```

Операторы

Операторы, как мы уже знаем, выполняют непосредственное действие над аргументами и, возможно, возвращают результат. Язык PHP предоставляет нам достаточно много операторов, которые мы сейчас рассмотрим.

Арифметические

Арифметические операторы, как нам уже известно, служат для выполнения арифметических действий над аргументами. Все арифметические операторы, поддерживаемые PHP, перечислены в табл. 7.2.

Таблица 7.2. *Арифметические операторы PHP*

Оператор	Описание
`-`	Смена знака числа
`+`	Сложение
`-`	Вычитание
`*`	Умножение
`/`	Деление
`%`	Остаток от деления
`++`	Инкремент (увеличение на единицу)
`--`	Декремент (уменьшение на единицу)

Дадим некоторые пояснения.

Оператор смены знака числа ставится перед аргументом и меняет его знак:

```
$r = -$r;
```

Это выражение извлекает значение переменной `$r`, меняет его знак и снова помещает в переменную `$r`.

Оператор инкремента может стоять как перед аргументом, так и после него. Давайте рассмотрим два выражения с этим оператором.

```
$t = ++$r;
```

Это выражение сначала инкрементирует значение переменной `$r`, а потом заносит его в переменную `$t`.

```
$t = $r++;
```

А это выражение, наоборот, сначала заносит значение переменной `$r` в переменную `$t`, а потом инкрементирует значение переменной `$r`. Точно так же работает и оператор декремента.

Внимание

Операторы инкремента и декремента рекомендуется использовать, если значение какой-либо переменной нужно просто увеличить или уменьшить на единицу. Эти операторы выполняются быстрее, чем операторы сложения и вычитания.

Оператор объединения строк

Оператор объединения строк . (точка) позволяет соединить две строки в одну. Например, сценарий:

```
$s1 = "Dreamweaver";
$s2 = "PHP";
$s3 = "MySQL";
$output = s1 . " " . s2 . " " . s3;
```

поместит в переменную `$output` строку `Dreamweaver PHP MySQL`.

Операторы присваивания

Единственный оператор простого присваивания = нам уже знаком. С его помощью переменной присваивается новое значение:

```
$a = 2;
```

Кроме оператора простого присваивания, PHP поддерживает несколько *операторов сложного присваивания*. Такие операторы позволяют выполнять операцию присваивания одновременно с другой операцией:

```
$a = $a + $b;
$a += $b;
```

Эти два выражения эквивалентны по результату. Просто во втором был использован оператор сложного присваивания +=, выполняющий присваивание вместе со сложением.

Все операторы сложного присваивания, поддерживаемые PHP, и их эквиваленты приведены в табл. 7.3.

Таблица 7.3. *Операторы сложного присваивания PHP*

Оператор	Эквивалентное выражение
`$a += $b;`	`$a = $a + $b;`
`$a -= $b;`	`$a = $a - $b;`
`$a *= $b;`	`$a = $a * $b;`
`$a /= $b;`	`$a = $a / $b;`
`$a %= $b;`	`$a = $a % $b;`

Внимание

Операторы сложного присваивания выполняются быстрее совокупности арифметического оператора и оператора присваивания.

Самое забавное в том, что операторы присваивания тоже возвращают результат! Он равен тому значению, которое получит переменная, стоящая слева от оператора. Например, в сценарии:

```
$b = $a = 3;
```

сначала переменная `$a` получит значение 3, а потом переменная `$b` получит значение вычисления `$a = 3`, которое также равно 3.

Операторы сравнения

Операторы сравнения сравнивают два аргумента согласно какому-то условию и возвращают логическое значение. Если условие сравнения выполняется, возвращается значение "истина" (`true`), если не выполняется — "ложь" (`false`). Вот примеры выражений, использующих операторы сравнения:

```
$a1 = (2 < 3);
$a2 = (-4 > 0);
$a3 = ($r < $t);
```

В результате вычисления этих выражений переменная `$a1` получит значение `true` (2 меньше 3), переменная `$a2` — значение `false` (−4 не может быть больше нуля), а значение переменной `$a3` будет зависеть от значений переменных `$r` и `$t`. Такие выражения, возвращающие логические величины, называются *логическими*.

Все поддерживаемые PHP операторы сравнения приведены в табл. 7.4.

Таблица 7.4. *Операторы сравнения PHP*

Оператор	Описание
<	Меньше
>	Больше
==	Равно
<=	Меньше или равно
>=	Больше или равно
!= или <>	Не равно
===	Строго равно
!==	Строго не равно

На двух последних операторах — "строго равно" и "строго не равно" — нужно остановиться подробнее. Это операторы так называемого *строгого сравнения*. Обычные операторы "равно" и "не равно", если встречают аргументы, содержащие значения разных типов, пытаются преобразовать их к одному

типу (о преобразованиях типов см. далее в этой главе). Операторы строгого равенства и строгого неравенства такого преобразования не делают, а в случае несовместимости типов аргументов всегда возвращают значение `false`.

Логические операторы

Логические операторы выполняют действия над логическими значениями. Все они приведены в табл. 7.5. А в табл. 7.6 и 7.7 показаны результаты выполнения этих операторов.

Таблица 7.5. *Логические операторы PHP*

Оператор	Описание
`!`	НЕ (логическая инверсия)
`&&` или `and`	И (логическое умножение)
`||` или `or`	ИЛИ (логическое сложение)

Таблица 7.6. *Результаты выполнения операторов И и ИЛИ*

Аргумент 1	Аргумент 2	&& (И)	|| (ИЛИ)
`true`	`true`	`true`	`true`
`true`	`false`	`false`	`true`
`false`	`true`	`false`	`true`
`false`	`false`	`false`	`false`

Таблица 7.7. *Результаты выполнения оператора НЕ*

Аргумент	! (НЕ)
`true`	`false`
`false`	`true`

Основная область применения логических операторов — логические выражения с операторами сравнения. Небольшой пример такого выражения:

```
$flag = !($status == 0);
```

Здесь переменная `$flag` получит значение `true`, если значение переменной `$status` НЕ равно нулю (обратим внимание на оператор логической инверсии, стоящий перед условием `$status == 0`).

```
$flag = ($a > 3) || ($b + $c != 10);
```

А здесь переменная `$flag` получит значение `true`, если значение переменной `$a` больше трех ИЛИ сумма значений переменных `$b` и `$c` не равна десяти.

Как PHP вычисляет выражения, содержащие логические операторы

При вычислении выражений, содержащих логические операторы, PHP старается максимально облегчить себе жизнь. Он придерживается следующего правила: если в какой-то момент стало ясно, каков будет результат логического выражения, его вычисление прекращается.

Для примера давайте возьмем приведенное ранее выражение

```
$flag = ($a > 3) || ($b + $c != 10);
```

и посмотрим, как PHP поступит при его вычислении. (Вообще, это выражение должно вернуть значение `true`, если значение переменной `$a` больше трех ИЛИ сумма значений переменных `$b` и `$c` не равна десяти.)

Пусть значение переменной `$a` больше трех. Тогда значение `($a > 3)` будет равно `true`. А, как мы помним из табл. 7.6, если один из аргументов оператора `||` (логическое ИЛИ) равен `true`, то результат этого оператора также будет равен `true`. Выходит, если значение `($a > 3)` равно `true`, то нет разницы, какое значение будет у `($b + $c != 10)` — результат вычисления всего этого выражения будет `true`.

PHP считает так же. Поэтому, выяснив, что вычисление `($a > 3)` дает результат `true`, он прекращает вычисление этого выражения и помещает в переменную `$flag` результат — `true`.

Давайте теперь рассмотрим другое выражение:

```
$d = ($t == 0) && ($y < 100);
```

Пусть результат вычисления `($t == 0)` будет равен `false`. Оператор логического И `&&` вернет `true`, если оба его аргумента равны `true`. В данном же случае ясно, что значение всего выражения будет равно `false` — один из аргументов этого оператора точно не равен `true`. И поэтому PHP прекратит выполнение выражения раньше времени.

Эта особенность заметно ускоряет обработку сценариев, содержащих логические выражения с операторами `&&` и `||`. Опытные программисты знают о ней и часто используют.

Совместимость и преобразование типов данных

Теперь нам нужно обязательно рассмотреть два важных вопроса: *совместимость типов данных* и *преобразование* одного типа к другому.

Что получится, если сложить два числовых значения? Правильно — еще одно числовое значение. А если сложить число и строку? Трудно сказать. Тут обработчик PHP сталкивается с проблемой несовместимости типов данных и пытается сделать эти типы совместимыми, преобразуя один из них к другому. Он попытается преобразовать строку в число и после этого выполнит сложение.

Преобразование строки в число выполняется по следующим правилам:

- если начало строки содержит цифры, которые можно преобразовать в целое число, строка будет преобразована в целое число;
- если начало строки содержит цифры, которые можно преобразовать в число с плавающей точкой, строка будет преобразована в число с плавающей точкой;
- если начало строки вообще не содержит цифр, строка будет преобразована в 0.

Рассмотрим такой сценарий:

```
$a = 11;
$b = "10 негритят";
$c = $a + $b;
```

Строка, содержащаяся в переменной `$b`, будет преобразована в число 10. Стало быть, переменная `$c` получит значение 21 (11+10). Заметим при этом, что исходная строка, содержащаяся в переменной `$b`, так и останется строкой.

При попытке объединить строку и число последнее будет преобразовано в строку:

```
$s = "PHP;
$n = 4;
$result = $s . $n;
```

Здесь переменная `$result` получит значение `PHP4`.

Логические величины преобразуются либо в числовые, либо в строковые, в зависимости от конкретного случая. Значение `true` будет преобразовано в число 1 или строку `"1"`, а значение `false` — в 0 или `"0"`.

При преобразовании строк или чисел в логические величины правила более сложные. Так, числовое значение 0 и строки `""` (пустая строка, не содержащая ни единого символа) и `"0"` будут преобразованы в значение `false`. Все остальное (ненулевые числа и непустые строки) будут преобразованы в `true`.

PHP изо всех сил пытается правильно выполнить даже некорректно написанные выражения. Иногда это получается, но чаще все работает не так, как планировалось, и, в конце концов, выполнение сценария прерывается в связи с обнаружением ошибки совсем в другом, абсолютно верном выражении. Поэтому лучше всего не допускать подобных казусов, оперировать только данными совместимых типов.

Если же все-таки нужно объединить в одном операторе разнотипные аргументы, нам придется выполнить *приведение типов*. Для этого достаточно поставить перед аргументом, который нужно преобразовать в другой тип, ключевое слово, обозначающее нужный нам тип данных, взяв его в скобки:

- `(bool)` — для преобразования в логический тип;
- `(int)` — для преобразования в целочисленный тип;

- `(float)` — для преобразования в тип с плавающей точкой;
- `(string)` — для преобразования в строковый тип.

Вот пример исправленного сценария, объединяющего строку и число, — теперь он использует приведение типов:

```
$s = "PHP;
$n = 4;
$result = $s . (string) $n;
```

В логических выражениях, использующих операторы сравнения, PHP также выполняет преобразование типов. Исключение составляют только операторы строгого сравнения, описанные ранее.

Приоритет операторов

Последний вопрос, который мы здесь рассмотрим, — это *приоритет* операторов. Мы помним, что приоритет влияет на порядок, в котором выполняются операторы в выражении. Сейчас настала пора поговорить об этом подробнее.

Пусть мы написали вот такой сценарий из одного выражения:

```
$a = $b + $c - $10;
```

Здесь сначала к значению переменной `$b` будет прибавлено значение переменной `$c`, а потом из получившейся суммы будет вычтено 10. Операторы этого выражения имеют одинаковый приоритет и поэтому выполняются строго слева направо.

Теперь рассмотрим другой сценарий:

```
$a = $b + $c * 10;
```

А здесь сначала будет выполнено умножение значения переменной `$c` на 10, а уже потом к полученному произведению будет прибавлено значение переменной `$b`. Дело в том, что оператор умножения * имеет бóльший приоритет, чем оператор сложения, поэтому обработчик PHP выполнит его первым.

Операторы присваивания — и простого, и сложного — имеют очень низкий приоритет (ниже — только у операторов `and` и `or`). Поэтому всегда сначала вычисляется само выражение, а уже потом его результат присваивается переменной.

Так что основной принцип выполнения всех операторов таков: сначала выполняются операторы с более высоким приоритетом, а уже потом — операторы с более низким. Операторы с одинаковым приоритетом выполняются в порядке их следования в выражении — слева направо.

В табл. 7.8 перечислены все изученные нами операторы в порядке убывания их приоритета.

Таблица 7.8. *Приоритет операторов PHP (в порядке убывания)*

Операторы	Описание
`++ -- - !`	Инкремент, декремент, смена знака, логическое НЕ
`* / %`	Умножение, деление, взятие остатка
`+ - .`	Сложение, вычитание и объединение строк
`< > <= >= = !=`	Сравнение
`&&`	Логическое И
`||`	Логическое ИЛИ
`= <оператор>=`	Присваивание, простое и сложное
`and`	Логическое И
`or`	Логическое ИЛИ

Но что делать, если нам нужно нарушить обычный порядок выполнения операторов? Воспользуемся скобками. При такой записи заключенные в скобки операторы имеют более высокий приоритет, чем находящиеся вне скобок, и поэтому всегда выполняются первыми.

```
$a = ($b + $c) * 10;
```

В этом сценарии сначала будет выполнено сложение значений переменных `$b` и `$c`, а потом получившаяся сумма будет умножена на 10.

Операторы, заключенные в скобки, также подчиняются приоритету. Поэтому часто используются многократно вложенные скобки.

```
$a = (($b + $c) * 10 - $d) / 2 + 9;
```

В этом сценарии операторы будут выполнены в такой последовательности:

1. Сложение значений `$b` и `$c`. (Этот оператор вложен в двойные скобки, поэтому имеет наивысший приоритет.)
2. Умножение полученной суммы на 10. (Сейчас выполняются операторы, вложенные в одинарные скобки.)
3. Вычитание значения `$d` из получившегося произведения.
4. Деление разности на 2. (А эти операторы находятся вне скобок, поэтому выполняются в последнюю очередь.)
5. Прибавление 9 к частному.

В общем, все достаточно просто. Так что мы пока закончим с операторами и перейдем к более сложным выражениям, без которых в достаточно сложном сценарии не обойтись. И заодно рассмотрим еще несколько операторов, с помощью которых создаются эти самые сложные выражения.

Сложные выражения PHP

Сложные выражения получили свое название потому, что составлены из нескольких простых выражений. Сложные выражения делятся на несколько видов и используются, в основном, для управления тем, как выполняется код сценариев. Сейчас мы их рассмотрим.

Блоки

Самые простые из сложных выражений (опять каламбур) — это *блочные выражения* или просто *блоки*. Они представляют собой несколько более простых выражений, объединенных в одно. Чтобы создать блок, достаточно заключить составляющие его простые выражения в фигурные скобки:

```
{
  $a = 2;
  $b = 3;
  $c = $a + $b;
  echo $c;
}
```

Заметим, что после закрывающей фигурной скобки точку с запятой ставить не нужно.

Обычно блоки не используются сами по себе. Чаще всего они входят в состав других сложных выражений.

Условные выражения

Условное выражение позволяет нам выполнить один из двух входящих в него блоков в зависимости от выполнения или невыполнения какого-либо *условия*. Таким условием может быть значение логической переменной или результат вычисления логического выражения.

Условное выражение создается с помощью особых операторов `if` и `else` и имеет следующий формат:

```
if (<условие>) {
  <блок, выполняющийся, если условие истинно>
} else {
  <блок, выполняющийся, если условие ложно>
}
```

`Условие` — это и есть логическое выражение, в соответствии с которым PHP принимает решение, какой блок выполнить. Если условие имеет значение `true` ("истина"), то выполняется первый блок. Если же условие имеет значе-

ние `false` ("ложь"), то выполняется второй блок. Заметим, что в условном выражении должны использоваться только блоки — это обязательно!

Вот пример условного выражения:

```
if ($x == 1) {
  $a = "Единица";
  $b = 1;
} else {
  $a = "Не единица";
  $b = 22222;
}
```

Здесь мы сравниваем значение переменной `$x` с единицей и, в зависимости от результатов сравнения, присваиваем переменным `$a` и `$b` разные значения.

Условие может быть и более сложным:

```
if (($x == 1) && ($y > 10)) {
  $f = 3;
} else {
  $f = 33;
}
```

Здесь мы использовали сложное условие, возвращающее значение `true` в случае, если значение переменной `$x` равно единице И значение переменной `$y` больше десяти. Заметим также, что даже в этом случае мы использовали блоки, хоть каждый из них состоит всего из одного простого выражения.

Существует также другая, "вырожденная" разновидность условного выражения, содержащая только одно выражение, которое выполняется при выполнении условия и пропускается, если условие не выполнено. Записывается оно так:

```
if (<условие>)
  <выражение, выполняющееся, если условие истинно>
```

А вот в этом случае необязательно использовать блоки. Давайте посмотрим сами:

```
if ($flag) {
  $status = "Операция выполнена.";
  $error_code = 0;
}
```

В этом выражении мы использовали блок, так как при истинности выражения (это значение переменной `$flag`) нам нужно выполнить два простых выражения.

```
if ($x < 0)
  $x = 0;
```

А здесь мы не использовали блоки, так как при истинности условия нам нужно выполнить всего одно простое выражение.

Кстати, последнее условное выражение мы можем записать по-другому, воспользовавшись особым *условным оператором* ?:

```
<условие> ? <выражение, выполняющееся, если условие истинно> :
↳<выражение, выполняющееся, если условие ложно>;
```

Здесь все так же, как и в случае "полновесного" условного выражения. Если условие истинно, выполняется первое выражение, если ложно — второе. При этом результат вычисления одного из этих выражений (первого или второго, в зависимости от истинности условия) будет оператором возвращен и может быть, например, сохранен в переменной или использован в другом выражении.

Давайте перепишем наше последнее условное выражение с помощью условного оператора:

```
$x = ($x < 0) ? 0 : x;
```

Получилось, как видим, очень компактно. Вот только с помощью оператора ? можно записывать только самые простые условные выражения, не использующие блоков.

Выражения выбора

Выражение выбора — это фактически несколько условных выражений, объединенных в одном. Его формат таков:

```
switch (<переменная или выражение>) {
  case <значение 1> :
    <набор выражений 1>
    [break;]
  [case <значение 2> :
    <набор выражений 2>
    [break;]
  . . .
  <другие секции case>
  . . .]
  [default :
    <набор выражений, выполняемых для остальных значений>]
}
```

В выражениях выбора используются операторы switch, case, default и break.

В выражении выбора значение *переменной* или результат вычисления выражения последовательно сравнивается со *значением 1*, *значением 2* и т. д. и, если такое сравнение увенчалось успехом, выполняется соответствующий *набор выражений* (первый, второй и т. д.). Если же ни одно сравнение не увенчалось успехом, выполняется набор выражений, находящийся в секции `default` (если, конечно, она есть).

Вот пример сценария, использующий выражение выбора:

```
switch ($a) {
  case 1 :
    $out = "Единица";
    break;
  case 2 :
    $out = "Двойка";
    break;
  case 3 :
    $out = "Тройка";
    break;
  default :
    $out = "Другое число";
}
```

Обратим внимание, что в секциях `case` используются не блоки, а именно наборы простых выражений — это важно! Выражения просто записываются одно за другим, но не заключаются в блоки.

Вообще, выражения выбора очень "прозрачны" и не требуют дополнительных пояснений. Давайте лучше поговорим о другом, а именно — об операторе `break`. Что он делает и можно ли без него обойтись?

Встретив оператор `break`, PHP прерывает выполнение набора выражений, в котором этот оператор присутствует, и начинает выполнять код, следующий за выражением выбора. Если в каком-то наборе выражений оператор `break` отсутствует, то PHP, завершив его, начнет выполнять набор выражений в следующей секции `case` и т. д., пока все-таки не встретит оператор `break` или пока не кончится выражение выбора.

Так, если мы уберем в нашем примере все операторы `break`:

```
switch ($a) {
  case 1 :
    $out = "Единица";
  case 2 :
    $out = "Двойка";
  case 3 :
```

```
    $out = "Тройка";
  default :
    $out = "Другое число";
}
```

то все блоки будут выполняться последовательно, один за другим. И переменной `$out` всегда будет присваиваться строка `Другое число`.

Циклы

Циклы — это особые выражения, позволяющие выполнить один и тот же блок несколько раз. Существует несколько разновидностей циклов, которые мы сейчас рассмотрим.

Цикл со счетчиком

Цикл со счетчиком используется, если какой-то блок нужно выполнить определенное число раз. Это наиболее часто используемый вид цикла.

Для подсчета того, сколько раз был выполнен блок, используется переменная, называемая *счетчиком цикла*. Перед каждым выполнением блока (*тела цикла*) производится проверка значения счетчика и, если оно еще не достигло предельного значения, блок выполняется. Если же значение счетчика достигло предельного значения, начинает выполняться код, находящийся после цикла.

Цикл со счетчиком создается с помощью оператора `for`, поэтому такие циклы часто называют "циклами for":

```
for (<выражение инициализации>; <условие>; <выражение приращения>)
  <тело цикла>
```

`Выражение инициализации` присваивает счетчику начальное значение. Далее проверяется *`условие`* цикла и, если его значение истинно (`true`), выполняется *`тело цикла`*. После этого выполняется *`выражение приращения`*, изменяющее значение счетчика, затем снова проверяется *`условие`* и т. д., пока *`условие`* не станет ложным (`false`), т. е. пока значение счетчика не достигнет своего предельного значения.

Вот пример цикла со счетчиком:

```
for ($i = 1; $i < 11; $i++) {
  $a += 3;
  $b = $i * 2 + 1;
}
```

Этот цикл будет выполнен 10 раз. Мы присваиваем счетчику `$i` начальное значение 1 и после каждого выполнения тела цикла увеличиваем его

на единицу. Цикл перестанет выполняться, когда значение счетчика увеличится до 11, и условие цикла станет ложным.

Счетчик цикла можно использовать в одном из выражений тела цикла (мы это и сделали). В самом деле, он содержит последовательно возрастающие значения от 1 до 10, которые можно использовать в вычислениях.

Приведем еще два примера цикла со счетчиком:

```
for ($i = 10; $i > 0; $i--) {
  $a += 3;
  $b = $i * 2 + 1;
}
```

Здесь значение счетчика декрементируется. Начальное его значение равно 10. Цикл выполнится 10 раз и завершится, когда счетчик `$i` будет содержать 0; при этом значения последнего будут последовательно уменьшаться от 10 до 1.

```
for ($i = 2; $i < 21; $i += 2) {
  $b = $i * 2 + 1;
}
```

А в этом примере начальное значение счетчика равно 2, а конечное — 21, но цикл выполнится опять же 10 раз. А все потому, что значение счетчика увеличивается на 2 и последовательно принимает значения 2, 4, 6... 20.

Заметим, что в циклах со счетчиком тело цикла обязательно состоит из блока, даже если оно представляет собой одно выражение.

Цикл с постусловием

Цикл с постусловием во многом похож на цикл со счетчиком, а именно в том, что его тело выполняется до тех пор, пока остается истинным условие цикла. Единственное исключение: условие проверяется не до, а после выполнения тела цикла, отчего цикл с постусловием и получил такое название.

Для задания цикла с постусловием используются операторы `do` и `while`. Поэтому такие циклы часто называют "циклами do-while".

```
do
  <тело цикла>
while (<условие>);
```

Вот пример цикла с постусловием:

```
$a = 0;
$i = 1;
do {
  $a = $a * $i + 2;
  $i = ++$i;
} while ($a < 100);
```

С ним все понятно. А вот еще один пример:

```
$a = 0;
$i = 1;
do {
  $a = $a * $i + 2;
  $i = ++$i;
} while ($i < 20);
```

Посмотрим на него внимательно. Мы фактически создали цикл со счетчиком, не так ли?

Интересная особенность: цикл с постусловием выполнится хотя бы один раз, даже если его условие с самого начала ложно. Это в некоторых случаях можно использовать.

Цикл с предусловием

Цикл с предусловием отличается от цикла с постусловием тем, что условие проверяется перед выполнением тела цикла. Так что, если условие изначально ложно, тело не выполнится ни разу.

Для создания цикла с постусловием используется уже знакомый нам оператор `while`. Поэтому такие циклы называют еще "циклами while".

```
while (<условие>)
  <тело цикла>
```

Пример цикла с предусловием:

```
while ($a < 100) {
  $a = $a * $i + 2;
  $i = ++$i;
}
```

А теперь давайте посмотрим вот на такой цикл:

```
while (true) {
  $i = ++$i;
}
```

Условие этого цикла истинно всегда — используется значение `true`. Это значит, что такой цикл будет выполняться вечно (*бесконечный цикл*). Такие циклы иногда используются в программировании, но при этом предусматривается условие, по которому бесконечный цикл можно прервать. Вот о прерывании выполнения цикла и предназначенных для этого двух операторах нужно поговорить особо.

Замечание

На самом деле, бесконечный цикл будет выполняться не вечно. Во-первых, программист может сам прервать его выполнение, закрыв содержащую его программу. Во-вторых, сама программа может совершить ошибку и будет закрыта операционной системой или обработчиком PHP. Ну, и, в-третьих, как уже говорилось, программист может предусмотреть условие для прерывания цикла.

Прерывание цикла

Как мы уже выяснили, бесконечный (а иногда — и обычный, "конечный") цикл нужно в конце концов прервать. Специально для этого PHP предусматривает два оператора: `break` и `continue`.

Оператор `break` прерывает выполнение цикла, после чего начинает выполняться следующий за циклом код.

```
while ($a < 100) {
  $a = $a * $i + 2;
  if ($a > 50) {
    break;
  }
  $i = ++$i;
}
```

В этом примере выполнение цикла прервется, если значение переменной `$a` превысит 50.

Собственно, мы уже знакомы с оператором `break`. Он использовался в выражении выбора, где вел себя точно так же.

А оператор `continue` выполняет *перезапуск* цикла. Он оставляет невыполненными все следующие за ним выражения, входящие в тело цикла, и запускает выполнение следующего шага цикла: проверка условия, приращение счетчика, выполнение тела и т. д.

```
while ($a < 100) {
  $i = ++$i;
  if (($i > 9) && ($i < 21)) {
    continue;
  }
  $a = $a * $i + 2;
}
```

Здесь мы пропускаем последнее выражение тела, если значение переменной `$i` находится в диапазоне от 10 до 20, и начинаем выполнять цикл с начала.

Вот и все о сложных выражениях.

Функции

Функция — это особым образом написанный и оформленный фрагмент кода PHP, который можно вызвать из любого места любого сценария, присутствующего в данной серверной странице. Как правило, в виде функции оформляется код, выполняющий однотипные и часто используемые в сценариях задачи. Тогда вместо того, чтобы писать его несколько раз в разных сценариях, его записывают в виде функции, а в нужных местах кода просто ставят выражение ее вызова.

Собственно код, ради которого и была создана функция, называется *телом функции* и оформляется в виде блока. Каждая функция должна иметь уникальное имя, по которому к ней обращаются. Функция, как и оператор, может принимать один или несколько аргументов и возвращать результат, который можно использовать в выражениях.

Создание функций

Прежде чем функция будет где-то использована, ее нужно объявить. Объявление функции выполняется с помощью ключевого слова `function`.

```
function <имя функции>([<список формальных аргументов, разделенных
↳запятыми>])
  <тело функции>
```

Имя функции, как уже говорилось, должно быть уникальным. Для имен функций действуют те же правила, что и для имен переменных: только латинские буквы, цифры и знаки подчеркивания, причем первым должна быть либо буква, либо знак подчеркивания. Однако — внимание! — знак доллара перед именем функций не нужен (знак доллара — это в языке PHP признак переменной). Кроме того, имена функций не зависят от регистра символов, которыми они набраны, так что `Func`, `func` и `FUNC` — это одна и та же функция.

Список формальных аргументов представляет собой набор переменных, в которые при вызове функции будут помещены значения ее аргументов. Мы можем придумать для этих переменных любые имена — все равно они будут использованы только внутри *тела функции*. Они так и называются — *формальные аргументы*.

Список формальных аргументов функции помещается в круглые скобки после имени функции, а сами аргументы отделяются друг от друга запятыми. Если функция не требует аргументов, сами скобки все равно нужно указать.

В пределах *тела функции* над принятыми ей аргументами (если они есть) и другими данными выполняются некоторые действия и, возможно, выраба-

тывается результат. Чтобы вернуть результат из функции в выражение, из которого она была вызвана, используется оператор `return`:

```
return <переменная или выражение>;
```

Здесь *переменная* должна содержать возвращаемое значение, а *выражение* должно его вычислять.

Вот пример объявления простейшей функции:

```
function divide($a, $b) {
  $c = $a / $b;
  return $c;
}
```

Данная функция принимает два аргумента — `$a` и `$b`, — после чего делит `$a` на `$b` и возвращает в качестве результата частное от этого деления. При этом для хранения промежуточного результата она использует собственную (так называемую локальную) переменную `$c`.

Разумеется, можно обойтись и без переменной, записав функцию `divide` в одну строку:

```
function divide($a, $b) { return $a / $b; }
```

Так намного короче.

Некоторым формальным аргументам функции можно дать значение по умолчанию, создав *необязательный аргумент*. Значение по умолчанию просто присваивается нужному аргументу с помощью оператора = прямо внутри скобок:

```
function divide($a, $b = 2) {
  $c = $a / $b;
  return $c;
}
```

При вызове этой функции значение второго аргумента можно будет опустить — тогда оно станет равным 2. Только нужно иметь в виду, что аргументы, имеющие значения по умолчанию, должны находиться самыми последними в списке формальных аргументов.

Вызов функций

После того как мы объявили функцию, мы можем *вызвать* ее из любого сценария в пределах текущей активной серверной страницы. Вызов функции записывается так:

```
<имя функции>([<список фактических аргументов, разделенных запятыми>])
```

Здесь указывается *имя* нужной функции и в круглых скобках перечисляются *фактические* аргументы, над которыми и нужно выполнить соответствую-

щие действия. Функция вернет результат, который можно присвоить переменной или использовать для вычисления выражения.

Внимание

При вызове функции нужно подставить именно фактические аргументы, а не формальные, использованные в объявлении функции.

Вот пример вызова объявленной нами ранее функции `divide`:

```
echo divide(3, 2);
```

Здесь мы подставили в выражение вызова функции фактические аргументы 3 и 2.

```
$s = 4 * divide($x, $r) + $y;
```

Здесь же мы выполняем вызов функции с переменными в качестве фактических аргументов. А возвращенный функцией результат используется для вычисления выражения.

Вызов функции, не возвращающей результата, еще проще:

```
somefunction($d, $f, 5, 0);
```

Кстати, таким же образом можно вызвать любую функцию, возвращающую результат. В таком случае этот самый результат будет проигнорирован.

При вызове функции можно опустить аргументы, для которых были заданы значения по умолчанию. Например, так:

```
echo divide(3);
```

В этом случае второй формальный параметр (`$b`) функции `divide` получит значение 2 (см. объявление этой функции ранее).

Функции можно вызывать друг из друга:

```
function samplefunc1($a, $b) {
  . . .
}
function samplefunc2($c) {
  . . .
  $k = $y + samplefunc1($x, 2);
}
```

Нужно только всегда иметь в виду, что функция должна быть объявлена прежде, чем будет использована.

Использование переменных внутри тела функции

Кстати, переменные, объявленные и используемые в функциях для хранения промежуточных результатов, сто́ят отдельного разговора. Такие переменные называются *локальными*.

Если мы объявим какую-нибудь переменную вне функции, то сможем обратиться к ней из любого сценария, находящегося в той же серверной странице. (Единственная оговорка: обращаться к переменной мы можем только после того, как ее объявим.) Программисты говорят, что такая переменная "видна" внутри страницы и называется *переменной уровня страницы*.

Что касается переменных, объявленных внутри тела функции, то они считаются локальными и не "видны" вне ее тела. То есть вот такой сценарий:

```
function func2() {
  $c = 2;
  return $a * $b + $c;
}
echo $c;
```

работать не будет. Перефразируем известную пословицу: "Своя переменная ближе к телу (функции)".

А теперь — внимание! Если мы попытаемся обратиться к объявленной вне функции переменной внутри ее тела, то потерпим неудачу. Ведь PHP подумает, что мы пытаемся обратиться к локальной переменной функции, которая еще не объявлена. А необъявленные переменные всегда имеют значение `NULL`.

Но что делать, если нам нужно, чтобы объявленная вне тела функции переменная была "видна" внутри него? Для этого нужно объявить эту переменную как *глобальную*, то есть доступную отовсюду. Синтаксис данного объявления таков:

```
global <список имен переменных, разделенных запятыми>;
```

Вот пример сценария, использующего глобальные переменные:

```
$a = 3;
$b = 4;
function func1() {
  global $a, $b;
  return $a * $b;
}
echo func();
```

Этот сценарий будет работать, выведя на экран 12 (произведение 3 и 4).

После того как выполнение тела функции завершится, все объявленные в нем локальные переменные уничтожаются. Иногда бывает нужно, чтобы они не уничтожались, а сохраняли свои значения, для чего достаточно поставить в начале выражений, объявляющих эти переменные, ключевое слово `static`.

```
function count() {
  static $i = 0;
  return ++$i;
}
```

При первом вызове этой функции создается локальная переменная `$i`, и ее значение устанавливается в 0. Далее в теле функции это значение инкрементируется и возвращается в качестве результата. Когда выполнение тела функции завершится, переменная `$i` сохраняется в памяти вместе со значением и будет использована при последующих вызовах этой функции. Таким образом, функция `count` при каждом вызове будет инкрементировать значение переменной `$i` и возвращать результат.

Такие переменные, хранящие свои значение между вызовами функции, называются *статическими*.

Встроенные функции PHP

Мы узнали, как можно создавать свои функции. Но язык PHP предлагает нам огромный набор уже существующих в нем (*встроенных*) функций, которые мы также можем использовать в своих сценариях.

Полное описание всех встроенных функций приведено в интерактивной документации по PHP. Здесь мы рассмотрим только некоторые из них.

Не возвращающая результата функция `unset` позволяет удалить из памяти ненужную переменную. Формат ее вызова таков:

```
unset(<список удаляемых переменных, разделенных запятыми>);
```

Например:

```
unset($a, $b, $c);
```

Функция `gettype` в качестве результата возвращает строку, описывающую тип данных переданного этой функции аргумента:

```
gettype(<переменная или выражение>);
```

Эта функция может вернуть одно из следующих строковых значений:

- `boolean` — логический тип;
- `integer` — целочисленный тип;
- `double` — тип с плавающей точкой;

- `string` — строковый тип;
- `NULL`.

В дальнейшем, когда мы начнем писать реальные сценарии, мы изучим еще несколько встроенных функций PHP. А пока закончим с функциями.

Массивы

Мы уже довольно много знаем о переменных и работе с ними. Но наши знания все еще неполны. Так, мы ничего пока не знаем о массивах — особом способе хранения данных, доступном в PHP. Давайте же выясним, что это такое.

Создание массивов и работа с ними

Массив — это пронумерованный набор переменных, представляющих собой единое целое. Переменные, входящие в массив, называются его *элементами*. Доступ к нужному элементу массива выполняется по его порядковому номеру, называемому *индексом*. А общее число элементов массива называется его *размером*.

Массивы идеально подходят для тех случаев, когда нужно хранить набор однотипных данных в одной структуре. Так, можно сохранить в массиве список названий всех семи дней недели, а потом получать нужное название по его номеру, который станет индексом этого массива.

Чтобы создать массив, достаточно просто присвоить любой переменной список его элементов, разделенных запятыми и заключенных в квадратные скобки:

```
$days = ["понедельник", "вторник", "среда", "четверг", "пятница",
↳"суббота", "воскресенье"];
```

Элементы созданного нами массива `$days` получат индексы от 0 до 6. А размер массива `$days` будет равен 7.

Внимание

Нумерация элементов массива начинается с нуля, а не единицы.

Теперь, чтобы получить доступ к нужному элементу массива, необходимо указать его индекс после имени массива в квадратных скобках:

```
echo $days[2];
```

Это выражение выведет на экран значение третьего элемента массива `$days` — строку `среда`.

Есть и другой способ создать массив — использование особой функции array:

```
$days = array("понедельник", "вторник", "среда", "четверг", "пятница",
↳"суббота", "воскресенье");
```

Видно, что в этом случае элементы будущего массива просто указываются в качестве аргументов этой функции.

Если будет нужно, мы легко сможем добавить к созданному ранее массиву еще один элемент, просто присвоив ему требуемое значение. Вот так:

```
$arr = [1, 2, 3, 4];
$arr[4] = 9;
```

Здесь мы сначала создали массив $arr из четырех элементов, а потом добавили к нему еще один, пятый по счету, элемент со значением 9.

Можно даже сделать так:

```
$arr[7] = 23;
```

Это выражение добавит массиву $arr еще один, шестой элемент со значением 23. Но индекс у него будет равен 7. Интересная возможность, не правда ли?

Если же мы создадим новый элемент массива с помощью выражения вида

```
$arr[] = 888;
```

то есть не укажем в скобках индекс создаваемого элемента, PHP присвоит ему индекс, равный индексу последнего элемента, увеличенному на единицу (в нашем случае — 7 + 1 = 8).

Кстати, пользуясь для создания массива функцией array, мы можем прямо в ней указать индексы элементов этого массива:

```
$arr = [1, 2, 3, 4, 9, 7 => 23, 888];
```

Это выражение создаст массив $arr из семи элементов. Первые пять будут иметь индексы от 0 до 4 соответственно. Шестой элемент получит индекс 7 (в записи вида 7 => 23 слева записывается индекс создаваемого элемента массива, а справа — значение этого элемента). И последний, седьмой, элемент получит следующий по счету "свободный" индекс — 8.

И еще одна замечательная возможность PHP — присвоение элементам массива строковых индексов:

```
$digits = array("один" => 1, "два" => 2, "три" => 3);
echo $digits["два"] + 2;
```

Первое выражение этого сценария создаст массив из трех элементов, имеющих индексы один, два и три соответственно. А второе выражение извлекает значение элемента с индексом два, прибавляет к нему 2 и выводит полученную сумму на экран.

Замечание

Во многих языках программирования массивы, элементы которых имеют строковые индексы, называются *ассоциативными массивами* или *хэшами*. В PHP же такие массивы никакого особого названия не имеют — просто "массивы".

Мы можем присвоить любому элементу массива другой массив (или, как говорят опытные программисты, создать *вложенный* массив).

```
$arr[6] = [1 => "n1", "n2", 10 => "n10"];
```

После этого можно получить доступ к любому элементу вложенного массива, указав после имени массива оба индекса последовательно, причем каждый индекс должен быть в квадратных скобках:

```
$str = $arr[6][2];
```

И переменная `$str` получит в качестве значения строку, содержащуюся во втором элементе вложенного массива, — `n2`.

Для удаления ненужного элемента массива или всего массива сразу мы можем воспользоваться уже знакомой функцией `unset`.

```
unset($arr[6]);
unset($digits);
```

Первое выражение приведенного ранее сценария удаляет элемент с индексом 6 массива `$arr` (этот элемент содержит вложенный массив). Второе же выражение удаляет массив `$digits` целиком.

Осталось сказать, что функция `gettype` возвращает для массива строку `array`.

Цикл просмотра

Ранее мы изучили три разновидности циклов, предоставляемых PHP. Но в нем имеется еще один цикл, специально предназначенный для работы с массивами. Это так называемый *цикл просмотра*, позволяющий выполнить какие-то действия над всеми элементами массива.

Цикл просмотра создается с помощью оператора `foreach`, поэтому часто называется "циклом foreach":

```
foreach (<имя массива> as <переменная-индекс> => <переменная-значение>)
  <тело цикла>
```

Здесь в скобках сначала указывается имя массива, с элементами которого нужно выполнить заданные в *теле цикла* действия. Далее, после ключевого слова `as`, указывается имя переменной, в которую будет занесено значение индекса очередного элемента массива, а после знака `=>` — имя переменной, куда будет занесено значение этого элемента.

Вот пример цикла просмотра:

```
foreach ($days as $index => $day) {
  echo "$days[" . $index . "] = " . $day . "\r\n";
}
```

Этот небольшой сценарий построчно (обратим внимание на специальный символ `\r\n`, задающий переход на новую строку) выведет на экран значения всех элементов массива `$days` вместе с их индексами.

Существует сокращенный вариант цикла просмотра, который записывается так:

```
foreach (<имя массива> as <переменная-значение>)
  <тело цикла>
```

Он используется, если значения индексов элементов массива не нужны.

```
foreach ($days as $day) {
  echo $day . "\r\n";
}
```

Константы

PHP предоставляет еще одну замечательную возможность, о которой нужно непременно поговорить. Это *константы* — числовые, строковые или логические значения, которым присвоено определенное имя.

Константы создаются с помощью встроенной функции `define`, имеющей такой формат вызова:

```
define(<значение константы>, <имя константы>);
```

Первым аргументом этой функции должно быть значение константы в строковом, числовом или логическом формате. Второй аргумент — это имя константы обязательно в строковом формате. Имена констант не предваряются знаком доллара и чувствительны к регистру символов, которыми набраны. Функция `define` не возвращает результата.

Давайте для примера создадим какую-либо константу:

```
define("PHP", "PLATFORM");
```

Это выражение создаст константу `PLATFORM`, имеющую строковое значение `PHP`. Обратим внимание на то, что имя константы набрано большими буквами — это своего рода стандарт де-факто в PHP.

Создав константу, мы можем использовать ее где-либо в наших сценариях:

```
echo "Мы работаем с " . PLATFORM;
```

И в этом случае имя константы не предваряется знаком доллара.

От переменных константы отличаются тем, что их значение всегда постоянно и не может быть изменено. Так, если мы напишем выражение

```
PLATFORM = "ASP";
```

то получим сообщение об ошибке.

PHP также предоставляет небольшой набор встроенных констант, определенных в нем самом. Например, константа `__FILE__` содержит путь и имя файла PHP, в котором хранится выполняемый в данный момент сценарий, в строковом виде.

С помощью констант можно создать набор неких постоянных значений, используемых в сценариях PHP, и сохранить его в одном месте. Если же потом потребуется изменить одно из значений, то сделать это потребуется только в том выражении, где создается соответствующая константа, а не выполнять правку всех сценариев, где оно используется.

Комментарии

Очень часто при написании сценариев возникает потребность поместить в код сценария какие-либо примечания для себя или коллег по работе. Для этого используются особые выражения языка PHP, называемые *комментариями*. Комментарии пропускаются обработчиком PHP, поэтому в них можно писать все что угодно.

Для вставки комментариев в код ActionScript предусмотрены три *оператора комментария*: `//`, `#` и `/*...*/`. Первый и второй позволяют вставить однострочный комментарий в конец любого выражения:

```
// Это однострочный комментарий
$a = $b + $c; # Это тоже однострочный комментарий
```

Заметим, что комментарий ставится после точки с запятой, обозначающей конец выражения.

Оператор `/*...*/` позволяет вставить в код сценария комментарий любого размера:

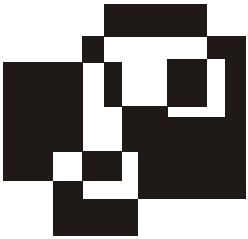
```
/*
В этом выражении мы складываем содержимое двух переменных
и помещаем результат в третью
*/
$a = $b + $c;
```

Разумеется, простой код комментировать не стоит — и так ясно, что мы складываем значения двух переменных и сумму помещаем в третью. А вот более сложный код прокомментировать полезно. Иначе мы сами со временем забудем, что он делает.

Что дальше?

Не будем слишком много распространяться о PHP. Тех основных возможностей, что мы здесь изучили, нам должно хватить на первое время. Если же нам от PHP понадобится что-то еще, мы изучим это по ходу дела.

А теперь — вперед! Хватит теории, настала пора заняться практикой. В следующей главе мы начнем создавать наши первые активные серверные Web-страницы на языке PHP. И поможет нам в этом несколько подзабытый Dreamweaver MX 2004.



Глава 8

Простейшие серверные Web-страницы. Вывод данных

Все, теоретический курс и различные подготовительные действия, не связанные с Web-дизайном, закончились. У нас есть база данных, мы вкратце "пробежались" по языку PHP и готовы взяться за работу.

Давайте приступим к созданию нашего нового сайта с использованием самых современных интернет-технологий. Создадим папку для файлов локальной копии этого сайта, назовем ее Site2, скопируем в нее главную страницу нашего сайта default.htm — мы договорились, что с небольшими изменениями используем ее и в новом сайте. После этого запустим Dreamweaver.

Когда Dreamweaver загрузится, откроем страницу default.htm нашего нового сайта и исправим интернет-адреса гиперссылок `файлы` и `статьи`. Они должны указывать на одну и ту же Web-страницу — Categories.php, — которая будет содержать список категорий, соответственно, файлов и статей. Этой страницей мы займемся в первую очередь.

Но сначала, как обычно, — небольшие подготовительные действия.

Подготовка к созданию серверных страниц

Давайте устроим небольшую ревизию — проверим, что нам нужно для создания активных серверных Web-страниц в среде Dreamweaver. А нужно не так уж и много, и почти все из этого у нас уже есть.

1. Полностью настроенный и работающий Web-сервер. Мы уже установили на свой компьютер Web-сервер Apache (процесс его установки и настройки описан в *приложении 1*).
2. Полностью настроенный и работающий сервер данных. У нас уже есть на компьютере сервер MySQL, с которым мы уже работали, когда изучали *главу 5*, — создавали нашу базу данных. (Процесс его установки и настройки описан в *приложении 2*.)

3. Полностью настроенный, работающий и интегрированный с Web-сервером обработчик PHP. И он у нас уже есть — мы установили его в *главе 7*. (Процесс его установки и настройки описан в *приложении 3*.)
4. Правильно зарегистрированный в Dreamweaver Web-сайт (будущий, разумеется, но зарегистрировать его нужно уже сейчас).

Первые три пункта списка у нас уже есть. А вот четвертый?..

Как регистрировать сайт в Dreamweaver, мы знаем — уже проделывали эту операцию в *главе 4*, когда публиковали свой первый сайт, содержащий только статические Web-страницы. Но дело в том, что сайты с активными серверными страницами нужно регистрировать особым образом, внося в Dreamweaver некоторые дополнительные данные. Какие — мы сейчас узнаем.

Итак, выбираем пункт **Manage Sites** в меню **Site**, нажимаем кнопку **New** в появившемся на экране диалоговом окне **Manage Sites** (см. рис. 4.1) и в появившемся на экране небольшом меню выбираем пункт **Site**. На экране появляется другое диалоговое окно — **Site Definition** (см. рис. 4.2). Переключаемся на категорию **Local Info** и вводим сведения о локальной копии нашего нового сайта; они будут такими же, как и у старого сайта, только имя папки и, соответственно, имя сайта будет не Site1, а Site2. Далее последовательно переключаемся в категории **Remote Info** и **Site Map Layout** и вводим данные в них; они будут теми же, что и для старого сайта, без всяких изменений. Все это нам уже знакомо.

А вот теперь мы готовы к тому, чтобы ввести дополнительные данные, нужные Dreamweaver для правильной поддержки выбранной нами технологии создания серверных страниц. Выбираем категорию **Testing Server** диалогового окна **Site Definition** (рис. 8.1).

С помощью раскрывающегося списка **Server model** задается сама технология, используемая нами для написания серверных страниц. Поскольку мы будем использовать PHP в "связке" с MySQL, то нам надо будет выбрать здесь пункт **PHP MySQL**.

На этом, в принципе, можно закончить и нажать кнопку **OK**. Но если мы так сделаем, то не сможем воспользоваться замечательной особенностью Dreamweaver — возможностью открывать серверные Web-страницы прямо в его среде (*"живой" просмотр*). Тем более что для использования "живого" просмотра нам придется сделать совсем немного.

Но как работает "живой" просмотр? Очень просто. Давайте предположим, что нам нужно проверить какую-либо серверную страницу в работе, и перечислим все действия, которые нужно для этого выполнить.

1. Скопировать все файлы этой Web-страницы (саму страницу, изображения, таблицы стилей и пр.) в корневую папку Web-сервера или одну из его виртуальных папок. Возможно, перед этим нам придется сохранить эту страницу, если мы исправили ее в Dreamweaver, но еще не сохранили.

2. Открыть Web-обозреватель.
3. Ввести в его строке адреса имя Web-сервера и имя файла нужной нам страницы.

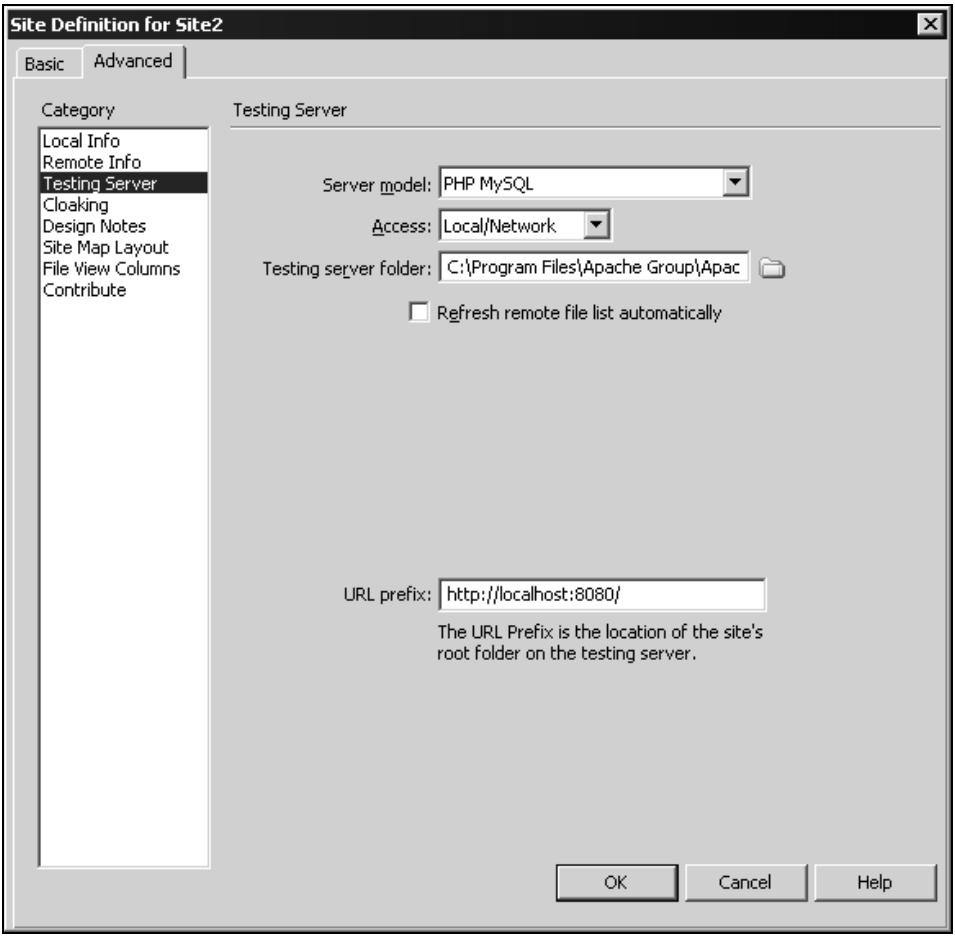


Рис. 8.1. Диалоговое окно **Site Definition** (категория **Testing Server**)

Сделать это, конечно, несложно. Но давайте представим, что нам придется выполнять эти действия каждую минуту, чтобы увидеть результат каждого исправления страницы. Так вот, Dreamweaver может сам сделать эти три действия вместо нас и откроет серверную страницу прямо в своей среде. Нам даже не понадобится запускать Web-обозреватель!

Такой возможностью грех не воспользоваться. Так что давайте дадим Dreamweaver все необходимые данные, чтобы он смог запустить "живой" просмотр.

Сначала в раскрывающемся списке **Access** выберем способ копирования тестируемых серверных страниц в папку Web-сервера (назовем эту папку *тестовой*, чтобы потом не путаться). Поскольку Web-сервер установлен на нашем же компьютере, выберем пункт **Local/Network** — копирование в локальную папку или по сети.

Теперь останется проверить, подставил ли Dreamweaver в поле ввода **Testing server folder** путь к корневой папке нашего сайта, который мы задали в категории **Remote Info**. Именно корневую папку нашего Web-сервера мы и используем в качестве тестовой. Также нужно проследить, чтобы Dreamweaver подставил в поле ввода **URL prefix** интернет-адрес нашего Web-сервера.

Задав нужные параметры, нажмем кнопку **OK** диалогового окна **Site Definition**, а потом кнопку **Done** диалогового окна **Manage Sites**. Если у нас открыта панель **Files**, в ней будут выведены все файлы нашего нового сайта Site2 (в настоящий момент там будет только главная страница default.htm). А Dreamweaver узнает, что мы собираемся работать с серверными страницами.

Регистрация базы данных в Dreamweaver

Наш следующий шаг — *регистрация базы данных* в Dreamweaver. Это нужно для того, чтобы Dreamweaver выяснил структуру базы данных, с которой мы собираемся работать.

Перед регистрацией базы данных нужно запустить сам сервер данных MySQL и Web-сервер Apache. Дело в том, что для выяснения структуры базы данных Dreamweaver помещает в тестовую папку Web-сервера особые серверные страницы PHP, а мы уже знаем, что для их выполнения нужен Web-сервер. Ну, а о сервере данных даже речи нет — он однозначно нужен!

И последний шаг перед регистрацией базы данных — создание первой серверной страницы. Увы, таково требование Dreamweaver, и с этим ничего не поделаешь. Так что выберем пункт **New** в меню **File**, в списке **Category** диалогового окна **New Document** выберем пункт **Dynamic Page**, а в правом списке — пункт **PHP**. После этого нажмем кнопку **Create** — и активная серверная страница будет создана. Сохраним ее, не закрывая, в корневой папке сайта Site2 под именем Categories.php (и используем потом для вывода списка категорий).

Для регистрации базы данных в Dreamweaver нам понадобится панель **Databases** (рис. 8.2). Чтобы вывести ее на экран, включим пункт-выключатель **Databases** в меню **Window** или нажмем комбинацию клавиш <Ctrl>+<Shift>+<F10>.

Бóльшую часть этой панели занимает список уже зарегистрированных баз данных. Но, поскольку мы еще не зарегистрировали ни одной базы, в панели **Databases** появится список, перечисляющий все шаги, которые мы должны выполнить для успешной регистрации баз данных в Dreamweaver. Мы эти шаги уже выполнили, так что перейдем сразу к процессу регистрации.

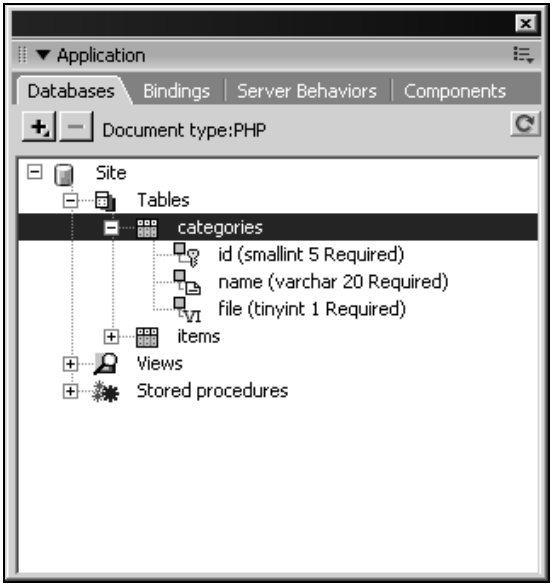


Рис. 8.2. Панель **Databases**

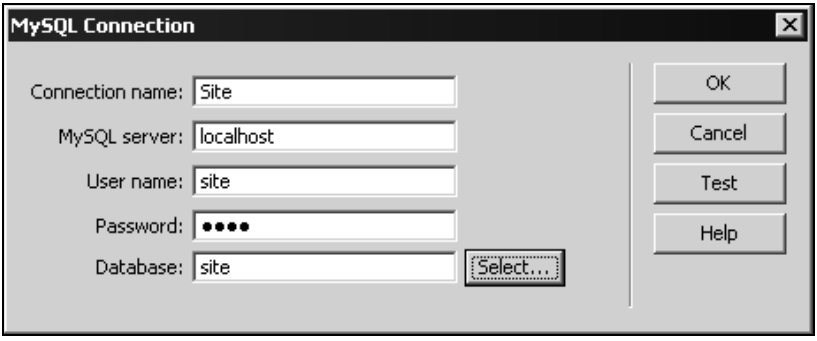


Рис. 8.3. Диалоговое окно **MySQL Connection**

Нажмем кнопку со знаком "плюс", расположенную над списком, и выберем в появившемся на экране небольшом меню единственный пункт **MySQL Connection**. На экране появится диалоговое окно **MySQL Connection**, показанное на рис. 8.3.

В поле ввода **Connection name** вводим имя соединения с нашей базой данных; под этим именем Dreamweaver будет хранить набор данных, необходимых для подключения к ней. Чтобы не ломать голову, введем туда то же имя, что мы дали для базы данных, — `Site`.

Поле ввода **MySQL server** служит для задания интернет-адреса сервера данных. Поскольку сервер установлен на нашем же компьютере, введем туда `localhost`.

В поля ввода **User name** и **Password** заносятся соответственно имя пользователя и пароль для подключения к базе данных. Вспоминаем, какие имя и пароль мы дали нашему пользователю — `site` и `site` соответственно.

Осталось только занести в поле ввода **Database** имя самой базы данных — `site`. Можно также щелкнуть кнопку **Select**, расположенную правее этого поля. На экране появится небольшое диалоговое окно **Select Database** (рис. 8.4), в списке **Select database** которого будет находиться единственная база данных, к которой имеет доступ пользователь `site`. Выберем ее и нажмем кнопку **OK**.

Рис. 8.4. Диалоговое окно **Select Database**

Чтобы проверить, правильно ли мы ввели все эти данные, нажмем кнопку **Test** диалогового окна **MySQL Connection**. В ответ Dreamweaver попытается подключиться к базе данных и выведет соответствующее окно-предупреждение. Если подключение прошло нормально, нажмем кнопку **OK**. Если же попытка подключения оказалась неудачной, нужно исправить введенные данные и повторить ее. Также нужно проверить, запущены ли сервер данных и Web-сервер.

Все, зарегистрированная нами база данных присутствует в списке панели **Databases** в виде "корня" иерархического списка (см. рис. 8.2), имя которого совпадает с именем самой базы данных. Этот "корень" разветвляется на три "дерева": **Tables**, **Views** и **Stored procedures**. Нас в данный момент больше интересует "дерево" **Tables**, содержащее имена всех созданных в базе данных `site` таблиц. Каждая таблица также представляет собой "дерево", развернув которое, мы увидим список имеющихся в этой таблице полей.

Чтобы изменить параметры регистрации базы данных, нужно выделить нужный "корень" и выбрать пункт **Edit Connection** в контекстном меню, по-

являющемся по щелчку правой кнопкой мыши. На экране отобразится уже знакомое нам диалоговое окно **MySQL Connection**, в котором мы сможем изменить все сведения о базе данных, кроме имени соединения.

Чтобы удалить ненужную базу данных из списка панели **Databases**, нужно выделить соответствующий "корень" и нажать кнопку со знаком "минус". На экране появится окно-предупреждение, в котором нужно нажать кнопку **Да** (Yes).

А теперь давайте посмотрим на список файлов локальной копии сайта, отображающийся в панели **Files**. (Если этой панели нет на экране, нужно включить пункт-выключатель **Files** в меню **Window** или нажать клавишу <F8>.) Видно, что в корневой папке Dreamweaver создал папку Connections, а в ней — файл Site.php. Название этого файла совпадает с именем соединения с базой данных, значит, этот файл содержит некий код PHP, устанавливающий это соединение.

Давайте посмотрим на этот код. Откроем файл Site.php в среде Dreamweaver и переключимся в режим отображения HTML-кода. В окне документа мы увидим следующее:

```
<?php
# FileName="Connection_php_mysql.htm"
# Type="MYSQL"
# HTTP="true"
$hostname_Site = "localhost";
$database_Site = "site";
$username_Site = "site";
$password_Site = "site";
$Site = mysql_pconnect($hostname_Site, $username_Site, $password_Site) or
↳trigger_error(mysql_error(),E_USER_ERROR);
?>
```

Если исключить тег `<?php...?>` и комментарии, содержащие служебные данные Dreamweaver, то останутся всего пять выражений, которые нам нужно рассмотреть. Эти выражения выделены полужирным шрифтом.

Первые четыре выражения задают значения переменных, содержащих, соответственно, интернет-адрес сервера данных, имя базы данных, имя пользователя и его пароль. Все это мы задали в диалоговом окне **MySQL Connection** (см. рис. 8.3).

А вот на пятом выражении нужно остановиться особо. В нем используются две встроенные функции PHP, с которыми мы еще не знакомы.

Первая функция — `mysql_pconnect` — устанавливает постоянное соединение с сервером данных. (О постоянных соединениях см. *главу 1*.) В качестве

аргументов она принимает интернет-адрес сервера данных, имя и пароль пользователя. (Заметим, что имя базы данных здесь еще не фигурирует.) Результатом этой функции будет особое ненулевое число (*идентификатор*), однозначно идентифицирующее установленное соединение и присваиваемое переменной `$Site` (опять же — имя этой переменной совпадает с именем зарегистрированного соединения).

Вторая функция — `trigger_error` — выводит на экран (конечно, не непосредственно на экран, а на Web-страницу) сообщение об ошибке. Первым аргументом этой функции должно быть само сообщение об ошибке в строковом виде (которое в нашем случае возвращает третья по счету функция `mysql_error`), а вторым — числовой код ошибки (почти всегда используется значение константы `E_USER_ERROR`).

А теперь давайте выясним, как вычисляется это пятое выражение из приведенного ранее сценария.

Предположим, что PHP удалось установить соединение с MySQL, и функция `mysql_pconnect` вернула корректный идентификатор и присвоила его переменной `$Site`. Тогда результат вычисления

```
$Site = mysql_pconnect($hostname_Site, $username_Site, $password_Site)
```

будет равен этому самому идентификатору.

Далее PHP встретит в пятом выражении оператор логического ИЛИ `or` и предположит, что это логическое выражение. Значит, возвращенный идентификатор нужно преобразовать в логическую величину. Этой величиной будет `true`, так как все ненулевые числа преобразуются в `true`. А поскольку оператор `or` вернет `true`, если один из его аргументов равен `true`, то вычислять выражение дальше не нужно, и функция `trigger_error` не будет выполнена.

Теперь давайте рассмотрим худший вариант — PHP не смог установить соединение с MySQL, и функция `mysql_pconnect` вернула 0 (она всегда возвращает 0 в таких случаях). 0 будет преобразован в `false`, PHP продолжит выполнение выражения, так как результат оператора `or` в таком случае не ясен, и выполнит функцию `trigger_error`. Посетитель сайта получит сообщение об ошибке.

Такой вот сценарий создал для нас Dreamweaver. Только не нужно удалять файл Site.php, в котором он хранится, иначе нам придется регистрировать базу данных в Dreamweaver заново.

Создание простейших серверных страниц

Вот теперь можно приступать к созданию в Dreamweaver серверных страниц. Все подготовительные операции мы уже выполнили.

Первая страница, которую мы создадим, будет содержать список категорий. Это страница Categories.php, которую мы уже, собственно, создали. Она, правда, не хранит никакого содержимого — только необходимые структуры HTML, созданные Dreamweaver. Нам нужно наполнить ее данными.

Итак, откроем в Dreamweaver страницу Categories.php, если она еще не открыта, дадим ей название, введем какой-либо поясняющий текст и создадим в ней таблицу из одной строки и одного столбца. В эту таблицу и будет помещен список категорий. А названия категорий станут гиперссылками, указывающими на страницы со списками файлов и статей.

Создание набора записей

Список категорий мы извлечем из таблицы `categories` нашей базы данных — это понятно. А для извлечения его мы напишем запрос SQL, вернее, заставим Dreamweaver написать его за нас — он горазд на такие фокусы. Сервер данных выполнит этот запрос и вернет нам ответ, содержащий нужные нам записи, — так называемый *набор записей* (по-английски — recordset).

Поскольку мы собираемся прибегнуть к помощи Dreamweaver, то давайте скажем ему, какой набор записей нам нужен. Для этого воспользуемся панелью **Bindings**, показанной на рис. 8.5. Чтобы вызвать ее на экран, включим пункт-выключатель **Bindings** в меню **Window** или нажмем комбинацию клавиш <Ctrl>+<F10>.

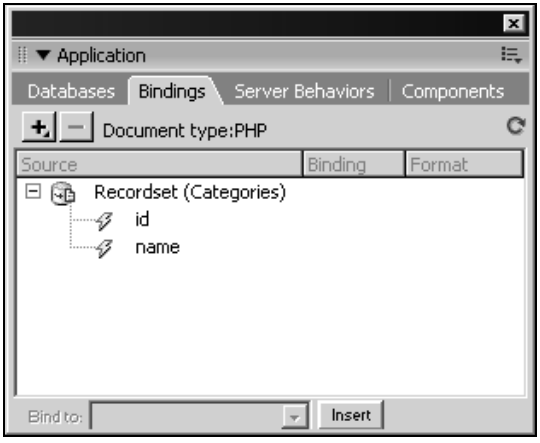


Рис. 8.5. Панель **Bindings**

Эта панель также состоит, в основном, из списка уже созданных к данному времени наборов записей. Изначально она, однако, содержит только нравоучительный текст, перечисляющий нерадивым Web-программистам шаги,

которые они должны пройти перед тем, как смогут создать свой первый набор записей. Но мы-то их уже прошли! Поэтому сразу же приступим к делу.

Нажмем кнопку со знаком "плюс", находящуюся на этой панели, и выберем в появившемся на экране меню пункт **Recordset (Query)**. На экране появится диалоговое окно **Recordset** (рис. 8.6).

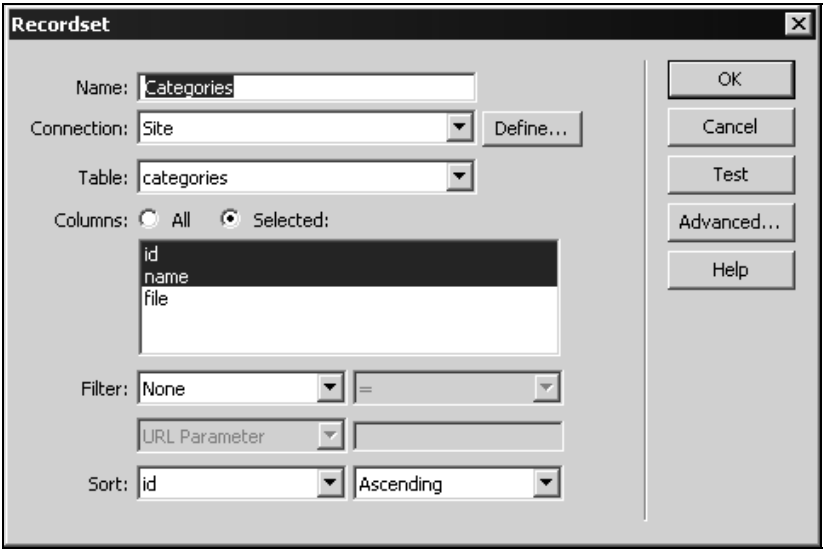


Рис. 8.6. Диалоговое окно **Recordset**

В поле ввода **Name** заносится имя создаваемого набора записей. Дадим ему имя `Categories` — так будет сразу понятно, зачем он нужен.

В раскрывающемся списке **Connection** выберем зарегистрированную нами ранее базу данных `Site`.

В раскрывающемся списке **Table** выберем нужную нам таблицу базы данных. Это таблица `categories`.

Группа переключателей **Columns** позволит нам выбрать поля таблицы, которые будут включены в набор записей. Мы можем включить переключатель **All** (впрочем, он включен по умолчанию), чтобы занести в набор записей все поля. Но давайте подумаем: нам нужны только поля `id` и `name`, а поле `file` фактически не нужно. Поэтому включим переключатель **Selected**, нажмем клавишу <Ctrl> и, удерживая ее нажатой, выберем нужные нам поля в списке, расположенном ниже.

В раскрывающемся списке **Sort** выбирается поле, по которому будут сортироваться записи набора, а в расположенном правее раскрывающемся списке — порядок сортировки (пункт **Ascending** — по возрастанию, а пункт **Descending** — по убыванию). Давайте выберем поле `id` и порядок сортировки

по возрастанию, чтобы записи в наборе выстроились в том порядке, в котором мы их ввели. (Можно, конечно, отсортировать их по полю `name`, но получится неаккуратно — категория `Прочее`, место которой в конце списка, окажется в его середине.)

Чтобы проверить, правильно ли введены все данные, нажмем кнопку **Test**. После этого на экране должно появиться диалоговое окно **Test SQL Statement**, показанное на рис. 8.7. В нем можно просмотреть полученный набор записей; чтобы закрыть окно, достаточно нажать кнопку **OK**. Вот только с русским языком у окна **Test SQL Statement** не все в порядке.

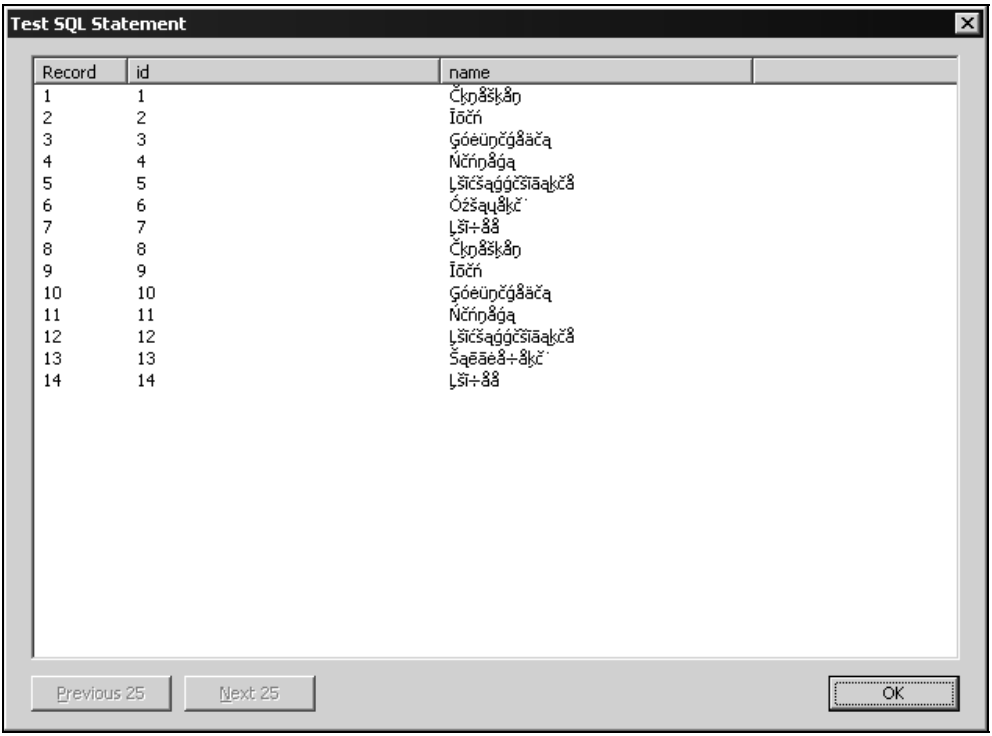


Рис. 8.7. Диалоговое окно **Test SQL Statement**

Если же параметры набора записей введены неправильно, Dreamweaver выведет на экран соответствующее окно-предупреждение. Нам придется снова проверить, правильно ли мы все ввели.

Закончив с параметрами набора записей, можно закрыть диалоговое окно **Recordset**, нажав кнопку **OK**. После этого в списке панели **Bindings** появится небольшое "деревце" с "корнем"-набором записей и "ветвями"-полями.

Если нам понадобится исправить параметры набора записей, нам будет нужно дважды щелкнуть по соответствующему "корню" списка. На экране

появится диалоговое окно **Recordset**, в котором мы сможем изменить нужные параметры. Чтобы удалить ненужный набор записей, необходимо выделить соответствующий "корень" и нажать кнопку со знаком "минус".

Теперь давайте сохраним нашу страницу Categories.php, выбрав пункт **Save** в меню **File** или нажав комбинацию клавиш <Ctrl>+<S>. Все, набор записей мы создали.

Dreamweaver создал за нас сценарий PHP, подключающийся к базе данных, выполняющий запрос SQL и получающий от сервера данных набор записей. А данные для создания SQL-запроса мы ввели в том самом диалоговом окне **Recordset**. (Собственно, об этом догадаться нетрудно.)

Создание самой серверной страницы

Закончив с набором записей, давайте перейдем к созданию самой серверной страницы, выводящей список категорий.

Итак, страница Categories.php у нас открыта. Проверим также, открыта ли панель **Bindings** — она нам сейчас понадобится. Расположим окно документа, в котором открыта страница, и панель так, чтобы они не перекрывали друг друга, а если и перекрывали, то так, чтобы созданная нами на странице таблица была видна. Вот теперь можно приступить к работе.

Выбираем в иерархическим списке панели **Bindings** "ветвь", соответствующую полю `name`, — содержимое именно этого поля будет выводиться на нашей странице. После чего перетаскиваем это поле на страницу и "бросаем" в единственную ячейку таблицы. Результат данного действия показан на рис. 8.8.

Мы только что создали *динамический текст* — так в терминологии Dreamweaver называется сценарий PHP, выводящий какие-либо данные. В нашем случае динамический текст выводит содержимое поля `name` набора записей `categories`.

Теперь самое время посмотреть, как все это будет работать в реальности. Воспользуемся режимом "живого" просмотра Dreamweaver, для чего включим кнопку-выключатель **Live Data View** (), расположенную в инструментарии документа. В ответ Dreamweaver поместит нашу страницу Categories.php в тестовую папку Web-сервера и запустит ее на выполнение. И мы увидим то, что показано на рис. 8.9.

Грандиозно! Она работает! Вот только почему-то вывела всего одну запись. А нам нужны все.

Чтобы наша страница выводила все записи, нам будет нужно создать на ней *повторяющуюся область* — особый элемент страницы, который будет выводиться столько раз, сколько записей находится в нашем наборе `categories`. Эта область будет включать единственную строку таблицы (тег `<TR>` с его содержимым); это значит, что содержимое поля `name` каждой записи набора будет выводиться в отдельной строке таблицы.

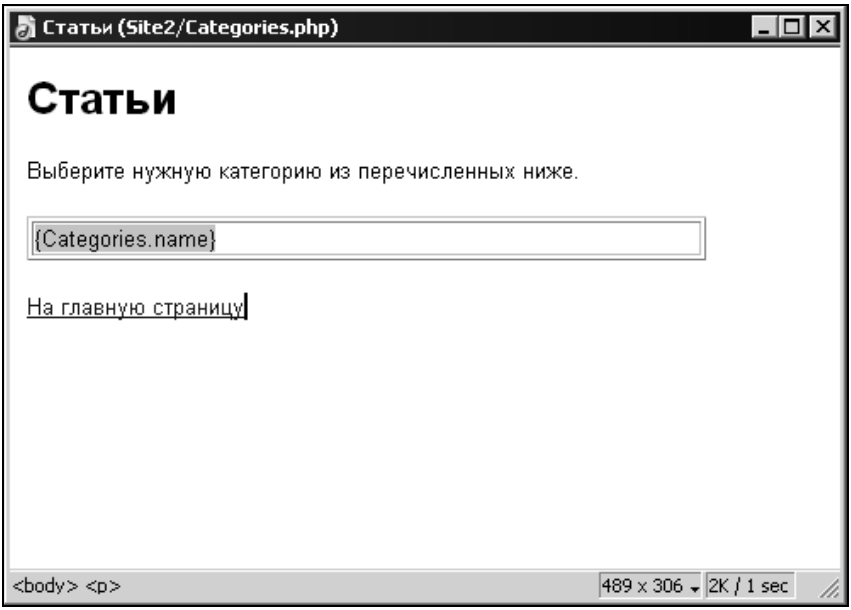


Рис. 8.8. Динамический текст, выводящий содержимое поля `name`

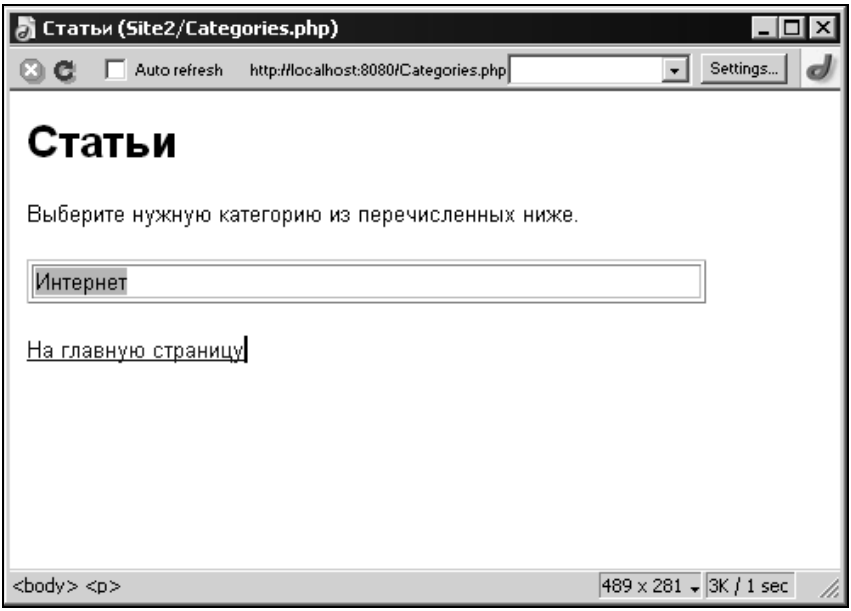


Рис. 8.9. Результат выполнения серверной страницы Categories.php

Рис. 8.10. Панель **Server Behaviors**

Поместим текстовый курсор куда-либо в ячейку таблицы и щелкнем в секции тегов, что в строке статуса окна документа, кнопку **<tr>**. Так мы выделим строку таблицы. После этого включим в меню **Window** пункт-выключатель **Server Behaviors** или нажмем комбинацию клавиш <Ctrl>+<F9>. На экране появится панель **Server Behaviors** (рис. 8.10).

Почти всю эту панель занимает список серверных поведений, созданных Dreamweaver на данной серверной Web-странице. *Серверными поведениями* (server behaviors) называются сценарии PHP, которые создает для нас Dreamweaver в коде серверной Web-страницы. В данный момент на странице Categories.php созданы два уже знакомых нам серверных поведения: **Recordset** (создание набора данных) и **Dynamic Text** (динамический текст). В скобках после названия серверного поведения указано имя набора записей, к которому оно относится.

Для создания повторяющейся области нам будет нужно добавить на страницу еще одно серверное поведение. Для этого нажмем кнопку со знаком "плюс" и в появившемся на экране меню выберем пункт **Repeat Region**. На экране появится диалоговое окно **Repeat Region**, показанное на рис. 8.11.

Рис. 8.11. Диалоговое окно **Repeat Region**

В раскрывающемся списке **Recordset** выбирается нужный нам набор записей — это, конечно, `Categories`. Содержимое создаваемой нами повторяющейся области будет выведено столько раз, сколько записей имеется в этом наборе.

Если включен верхний переключатель группы **Show**, то содержимое набора записей будет выводиться на страницу по частям, как бы постранично. Количество отображаемых одновременно записей (размер такой "страницы") задается в поле ввода **Records at a Time**. "Постраничный" вывод записей оправдан в том случае, когда записей очень много и, если бы они все выводились на одной Web-странице, посетитель сайта ждал бы слишком долго окончания ее загрузки. Но это не наш случай, и мы можем со спокойной совестью включить переключатель **All records**, предписывающий выводить все записи набора на одной странице.

Замечание

Если используется "постраничный" вывод записей набора, то нужно будет позаботиться о гиперссылках, выполняющих переход от "страницы" к "странице". Для этого Dreamweaver предоставляет серверные поведения **Move To First Page** (перемещение на первую "страницу"), **Move To Previous Page** (на предыдущую), **Move To Next Page** (на следующую) и **Move To Last Page** (на последнюю). Чтобы создать их, нужно нажать кнопку со знаком "плюс" панели **Server Behaviors** и выбрать в подменю **Recordset Paging** появившегося на экране меню соответствующий пункт.

Задав все нужные данные, можно нажать кнопку **OK**. Созданная нами повторяющаяся область будет показана в окне документа вот так — см. рис. 8.12. А в списке панели **Server Behaviors** появится новый пункт — **Repeat Region** (повторяющаяся область).

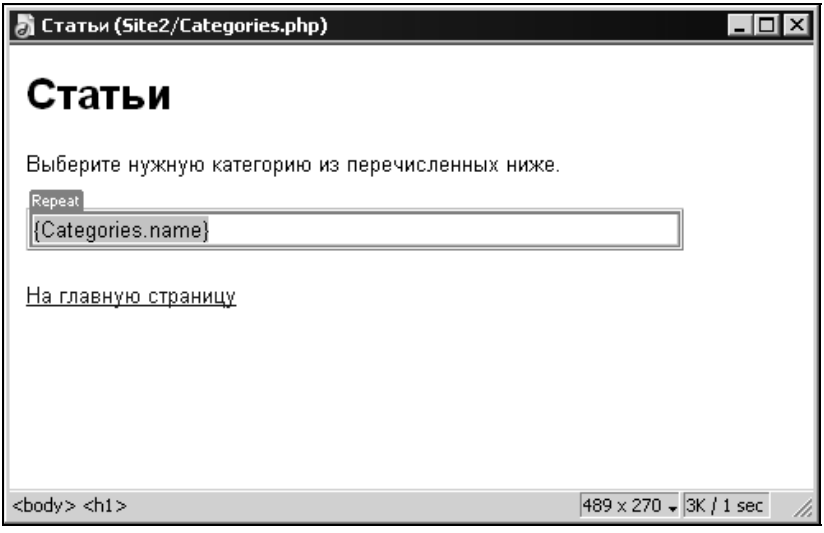


Рис. 8.12. Страница Categories.php с повторяющейся областью

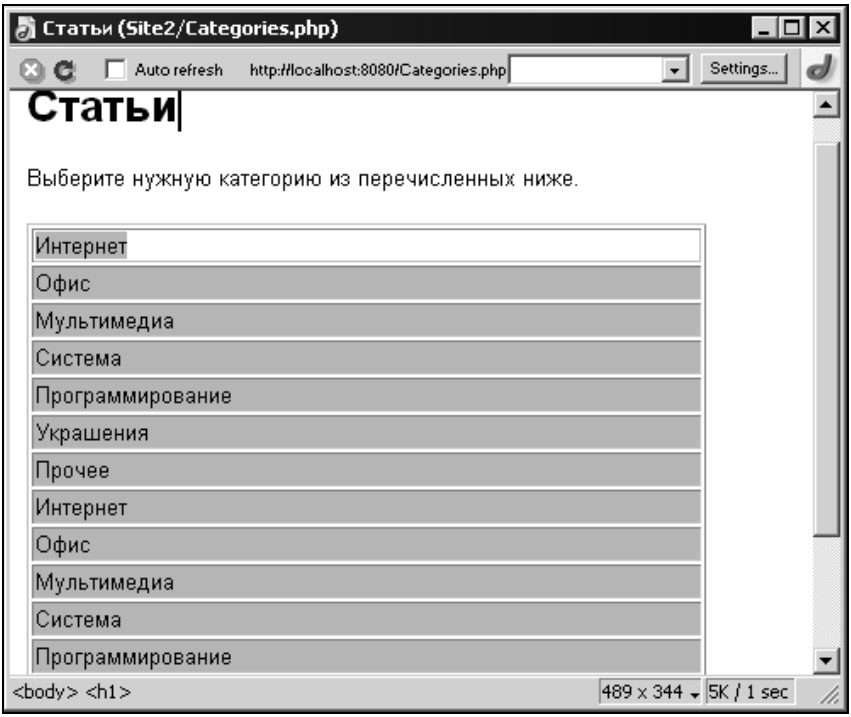


Рис. 8.13. Результат выполнения серверной страницы Categories.php с повторяющейся областью

Если же мы включим "живой" просмотр, то увидим, что наша серверная Web-страница работает так как надо, выводя все записи набора (рис. 8.13). (Вообще-то, она работает не совсем так как надо, но об этом мы поговорим позже.)

Вот так мы создали нашу первую активную серверную Web-страницу! И в этом нам помог Dreamweaver.

Осталось сказать немного. Чтобы изменить параметры какого-либо серверного поведения, нужно дважды щелкнуть по соответствующему пункту списка панели **Server Behaviors**. На экране появится диалоговое окно, в котором мы сможем изменить эти параметры. Так, если дважды щелкнуть по пункту **Recordset**, появится диалоговое окно **Recordset** (см. рис. 8.6), а если проделать то же самое с пунктом **Repeat Region**, то мы увидим диалоговое окно **Repeat Region** (см. рис. 8.11). А для удаления ненужного серверного поведения необходимо выделить соответствующий ему пункт и нажать кнопку со значком "минус".

Теперь давайте сохраним страницу Categories.php и переключимся в режим отображения HTML-кода Dreamweaver. Настала пора посмотреть, что за PHP-сценарии насоздавал Dreamweaver за нашей спиной.

Разбор сценариев PHP, используемых для вывода записей

Разбор кода PHP, созданного Dreamweaver, мы выполним по частям. Так будет проще понять, что он делает и к какому серверному поведению относится.

Первый сценарий PHP совсем короткий и состоит из одного-единственного выражения:

```
<?php require_once('Connections/Site.php'); ?>
```

Здесь мы сразу столкнулись с не знакомым нам оператором PHP — `require_once`. Этот оператор принимает один аргумент (заметим, что указывается он в скобках, как аргумент функции) — строку, содержащую путь к файлу PHP. Строка эта, как мы видим, взята в одинарные кавычки, что, в принципе, допустимо.

Все сценарии PHP, сохраненные в файле, имя которого передано оператору `require_once`, выполняются. Но выполнены они будут только один раз; если же в другом сценарии также встретится оператор `require_once` с тем же аргументом, он будет проигнорирован.

Что касается файла Site.php, хранящегося в папке Connections, то он содержит сценарий, устанавливающий постоянное соединение с сервером данных под заданным нами именем и паролем. Мы уже разбирали этот сценарий, когда говорили о регистрации базы данных в Dreamweaver.

По поводу первого сценария осталось сказать только то, что он вставляется в начало каждой серверной страницы PHP. Хотя он и не соответствует ни одному серверному поведению, но необходим для работы серверных поведений, работающих с базой данных.

Следующий сценарий соответствует серверному поведению **Recordset**. Вот он:

```
<?php
mysql_select_db($database_Site, $Site);
$query_Categories = "SELECT id, name FROM categories ORDER BY id ASC";
$Categories = mysql_query($query_Categories, $Site) or
↳die(mysql_error());
$row_Categories = mysql_fetch_assoc($Categories);
$totalRows_Categories = mysql_num_rows($Categories);
?>
```

Первое выражение использует встроенную функцию `mysql_select_db`, которая выполняет подключение к базе данных. В качестве аргументов она принимает имя базы данных в строковом формате (переменная `$database_Site`, объявленная в файле Site.php) и идентификатор соединения с сервером данных (переменная `$Site`, также объявленная в файле Site.php).

Следующее выражение совсем простое — оно присваивает переменной `$query_Categories` код SQL-запроса. Этот запрос построил сам Dreamweaver, основываясь на данных, которые мы ввели в диалоговом окне **Recordset** (см. рис. 8.6).

Третье выражение вызывает встроенную функцию `mysql_query`, которая выполняет SQL-запрос и возвращает в переменную `$Categories` идентификатор набора записей. (Обратим внимание, что имя переменной совпадает с именем набора записей, введенным нами в диалоговое окно **Recordset**.) В качестве аргументов эта функция принимает код SQL-запроса (переменная `$query_Categories`) и идентификатор соединения с сервером данных.

Если функция `mysql_query` не сможет выполнить запрос, она вернет 0, и тогда выполнится функция `die`. Эта функция прерывает выполнение всех сценариев PHP на данной странице и выводит сообщение об ошибке, переданное ей в качестве аргумента. А само это сообщение об ошибке в строковом виде вернет функция `mysql_error`.

Четвертое выражение опять содержит не знакомую нам встроенную функцию — `mysql_fetch_assoc`. Она принимает в качестве аргумента идентификатор набора записей (переменная `$Categories`), извлекает данные из первой записи этого набора и подготавливает для обработки следующую — вторую — запись.

В качестве результата функция `mysql_fetch_assoc` возвращает массив (и помещает его в переменную `$row_Categories`). Элементы этого массива — значения полей обработанной записи набора; они имеют индексы, представляющие собой имена соответствующих полей в строковом виде. То есть в нашем случае массив `$row_Categories` будет иметь такие элементы:

- `$row_Categories["name"]` — значение поля `name`;
- `$row_Categories["id"]` — значение поля `id`.

Именно из этого массива будут потом извлекаться значения полей всех записей набора `Categories`.

Последнее выражение тоже совсем простое. Встроенная функция `mysql_num_rows` принимает в качестве аргумента идентификатор набора записей и возвращает полное количество записей в этом наборе (и помещает его в переменную `$totalRows_Categories`).

Замечание

Собственно, данное выражение здесь и не нужно. Ни в одном сценарии, имеющемся на странице Categories.php, количество записей не используется. Dreamweaver вставил это выражение в код PHP в расчете на то, что мы создадим серверные поведения, которые могут использовать количество записей для каких-то целей (например, те же серверные поведения, выполняющие "листание"). Если же мы не собираемся создавать таких серверных поведений, то последнее выражение данного сценария можно удалить.

Обратим внимание, что оба приведенных ранее сценария PHP помещаются перед любым HTML-кодом.

Код HTML, задающий секцию заголовка Web-страницы, мы пропустим — там для нас нет ничего интересного. Обратимся сразу к HTML-коду, создающему нашу таблицу, вместе со сценариями PHP, которые выводят в нее данные из набора записей `Categories`. Вот этот код (все сценарии PHP выделены полужирным шрифтом):

```
<TABLE WIDTH="400"  BORDER="1" CELLSPACING="2" CELLPADDING="1">
  <?php do { ?>
  <TR>
    <TD><?php echo $row_Categories['name']; ?></TD>
  </TR>
  <?php } while ($row_Categories = mysql_fetch_assoc($Categories)); ?>
</TABLE>
```

Здесь мы видим нашу таблицу из одной строки и одной ячейки. Код HTML, создающий строку (тег `<TR>` вместе с содержимым), представляет собой тело цикла с постусловием (о циклах с постусловием см. *главу 7*). Это значит, что строка таблицы будет повторена в HTML-коде окончательной (выводимой посетителю сайта) Web-страницы столько раз, сколько выполнится тело этого цикла.

В качестве условия цикла используется вот такое знакомое нам выражение:

```
$row_Categories = mysql_fetch_assoc($Categories));
```

Мы уже знаем, что оно извлекает данные из текущей записи набора данных и подготавливает для обработки следующую запись. И, если это выражение будет выполняться последовательно много раз (а оно и будет так выполняться, ведь это условие цикла, вычисляемое после каждого выполнения его тела), оно, в конце концов, извлечет данные из всех записей набора `Categories`.

Когда будет достигнут конец набора записей (то есть все записи уже обработаны), это выражение вернет значение `false` — сообщение о том, что не может извлечь данные. Условие тотчас станет ложным, перестанет выполняться, и цикл завершится.

Подобный цикл соответствует серверному поведению **Repeat Region** (повторяющаяся область). Впрочем, догадаться об этом нетрудно.

А сам вывод значения поля `name` набора записей `Categories` выполняет сценарий

```
<?php echo $row_Categories['name']; ?>
```

Ну, здесь все совсем просто. Из массива `$row_Categories` извлекается элемент с индексом `name` (имя поля, содержащего названия категорий, в строковом

виде) и выводится на Web-страницу прекрасно знакомым нам оператором `echo`. Нетрудно догадаться, что этот сценарий соответствует серверному поведению **Dynamic Text** (динамический текст).

Осталось только рассмотреть последний, совсем простой сценарий, помещенный Dreamweaver в самом конце, после всего HTML-кода. Вот он:

```
<?php
mysql_free_result($Categories);
?>
```

Здесь используется еще одна не знакомая нам встроенная функция PHP — `mysql_free_result`. Она удаляет из памяти набор записей, идентификатор которого передан ей в качестве единственного аргумента.

Внимание

Набор записей занимает память серверного компьютера, причем чем больше он содержит записей, тем больше памяти требуется для его хранения. Поэтому после завершения работы с набором записей его всегда нужно удалять.

Да, рассмотренные нами сценарии PHP используют множество не знакомых нам встроенных функций и приемов. Но на самом деле в них нет ничего сложного. Ознакомившись с принципами их написания и получив немного практики, мы сможем писать их вручную, без помощи Dreamweaver.

А пока давайте посмотрим на нашу, казалось бы, вполне готовую страницу Categories.php и подумаем, что же она делает не так.

Передача данных между серверными страницами

Правильно! Она выводит список всех категорий — и статей, и файлов — несмотря на то, по какой гиперссылке страницы default.htm мы щелкнули. А нам это совсем не нужно!..

Нам нужно как-то отделить категории файлов от категорий статей, а для этого нужно знать, по какой гиперссылке щелкнул посетитель сайта. Короче говоря, нам нужно срочно придумать способ передачи данных от одной серверной Web-страницы к другой. Вот этим мы сейчас и займемся.

Метод передачи данных GET

К счастью, придумывать нам ничего не надо. Все уже придумано до нас. Это особый *метод передачи данных* по Сети от одной Web-страницы к другой, называемый *методом* `GET`. Сейчас мы его рассмотрим.

Из *главы 1* мы помним, что Web-обозреватель для того, чтобы получить от Web-сервера нужный ему файл, отправляет этому серверу клиентский запрос, включающий в себя интернет-адрес необходимого файла. Так вот, в случае метода `GET` данные передаются как часть этого адреса.

Передаваемые методом `GET` данные должны быть представлены в виде набора пар *<имя аргумента>=<значение аргумента>*. (Просматривается аналогия с переменными, не так ли?) Этот набор пар ставится в самый конец интернет-адреса и отделяется от него вопросительным знаком. Сами же пары должны быть отделены друг от друга знаком "коммерческое и" (&).

Получив такой интернет-адрес, Web-сервер отделяет имя файла, в котором сохранена серверная Web-страница, от передаваемых с ним данных и передает эти данные обработчику PHP. Сценарии, имеющиеся в серверной странице, могут запросить переданные данные, обратившись к особому встроенному (объявленному в самом PHP) массиву `$_GET`. Этот массив содержит элементы, являющиеся значениями переданных методом `GET` аргументов и имеющие индексы — имена этих аргументов в строковом виде.

Все это может показаться очень сложным, поэтому давайте рассмотрим использование метода `GET` на нашем примере. Чтобы отделить категории файлов от категорий статей, введем аргумент `file`. Значение 1 будет обозначать категории файлов, а значение 0 — категории статей. Тогда гиперссылка, указывающая на страницу со списком категорий файлов, будет иметь вид:

```
Categories.php?file=1
```

а гиперссылка, указывающая на список статей —

```
Categories.php?file=0
```

Видно, что использовать для вывода списка категорий мы будем одну и ту же серверную страницу — Categories.php. Просто сценарии этой страницы, в зависимости от значения переданного нами аргумента `file`, будут извлекать из таблицы `categories` базы данных `site` разные наборы записей.

Чтобы обратиться к аргументу `file`, наши сценарии должны использовать выражение вида

```
$is_file = $_GET["file"];
```

после выполнения которого в переменную `$is_file` будет помещено значение переданного аргумента `file`.

Конечно, вручную ничего такого нам писать не придется — все за нас сделает Dreamweaver. Он изменит уже созданные в странице Categories.php сценарии так, чтобы они могли обрабатывать передаваемые методом `GET` данные. Но, разумеется, нам тоже придется выполнить некоторые телодвижения.

Создание Web-страниц, передающих данные друг другу

Сначала откроем в Dreamweaver главную страницу нашего сайта default.htm и исправим гиперссылки, указывающие на списки категорий файлов и статей. Гиперссылка, указывающая на список категорий файлов, должна иметь такой интернет-адрес:

```
Categories.php?file=1
```

а гиперссылка, указывающая на список категорий статей, — такой:

```
Categories.php?file=0
```

Сохраним исправленную страницу default.htm и закроем ее.

Теперь настала пора внести изменения в страницу списка категорий Categories.php. Откроем ее, если она еще не открыта. Далее найдем в панели **Server Behaviors** пункт **Recordset**, соответствующий набору записей `Categories`, и дважды щелкнем по нему. На экране появится уже знакомое нам диалоговое окно **Recordset** (см. рис. 8.6).

Внимание

Если окно **Recordset** имеет вид, отличный от показанного на рис. 8.6, значит, оно переключилось в расширенный режим. Чтобы вернуть его в обычный режим, нужно нажать кнопку **Simple**.

Итак, нам нужно отобрать записи, основываясь на значении аргумента `file`, переданного от страницы default.htm методом `GET`. Для этого нам будет нужно добавить в запрос SQL, извлекающий данные из базы, критерий фильтрации записей. Точнее, нам нужно дать знать об этом Dreamweaver.

Нам нужно фильтровать записи по полю `file`. Поэтому выберем в раскрывающемся списке **Filter**, задающем поле, по которому выполняется фильтрация, пункт **file**. Раскрывающийся список справа задает оператор сравнения; выберем пункт =.

Ниже списка **Filter** находится другой раскрывающийся список, в котором выбирается вид значения, с которым будет сравниваться значение выбранного нами поля `file`. Поскольку нам нужно сравнивать значение этого поля с аргументом, переданным методом `GET`, то выберем в этом списке пункт **URL Parameter**. Теперь осталось только ввести в расположенное правее поле ввода имя аргумента — `file` — и все.

Из *главы 7* нам известно, что в PHP логическому значению `true` соответствует любое ненулевое числовое значение, а значению `false` — 0. Можно сказать, что MySQL придерживается таких же правил. (А точнее, он хранит логические значения `true` и `false` в виде чисел 1 и 0 соответственно.)

Введенные нами данные показаны на рис. 8.14. Проверим, правильно ли мы их ввели, и исправим ошибки, если они есть.

Рис. 8.14. Фрагмент диалогового окна **Recordset** с элементами управления, задающими критерий фильтрации записей

Теперь, в принципе, можно нажать кнопку **OK**. Но давайте еще немного поработаем с набором записей `Categories`. Давайте сделаем так, чтобы по умолчанию, если аргумент `file` вообще не был передан, страница Categories.php выводила список категорий статей.

Нажмем кнопку **Advanced**. Диалоговое окно **Recordset** изменит свой вид — см. рис. 8.15. Это так называемый расширенный режим, в котором мы можем задать дополнительные параметры серверного поведения **Recordset**. В частности, именно здесь можно задать значение по умолчанию для аргумента `file`.

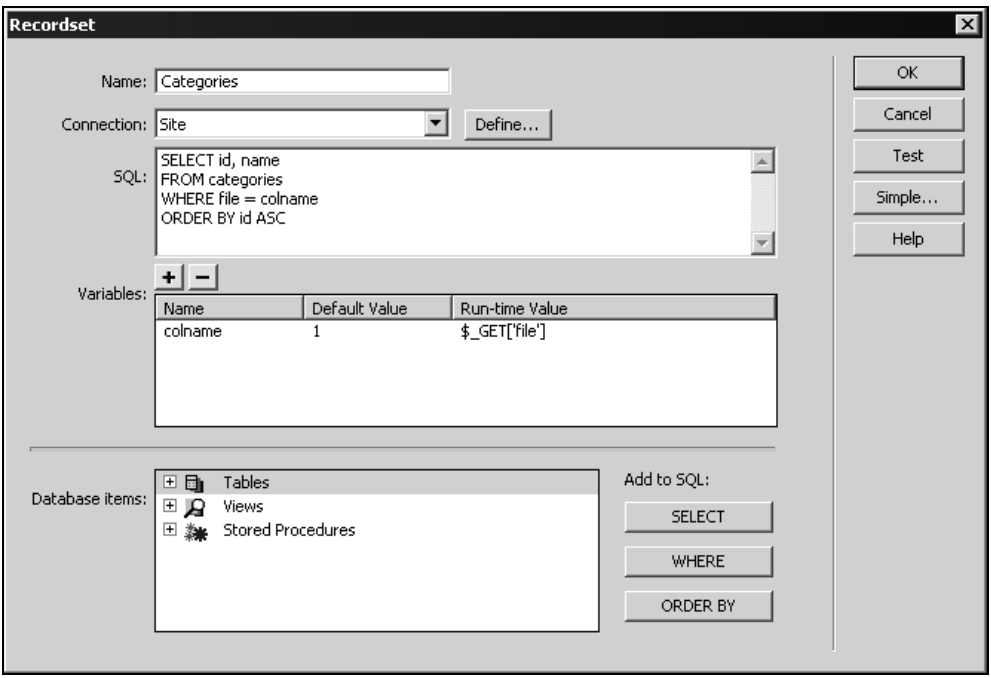


Рис. 8.15. Диалоговое окно **Recordset** (расширенный режим)

Раскрывающиеся списки **Name** и **Connection** нам уже знакомы. В области редактирования **SQL** отображается код SQL-запроса, сформированного Dreamweaver на основе введенных нами данных. Вот он:

```
SELECT id, name FROM categories WHERE file = colname ORDER BY id ASC
```

Здесь все ясно: из таблицы `categories` извлекаются поля `id` и `name` записей, значения поля `file` которых равно `colname`. Извлеченные записи сортируются по полю `id`, причем сортировка идет по возрастанию.

Но что такое `colname`? Это *переменная SQL-запроса*, которой Dreamweaver при выполнении этого запроса присваивает нужное значение (в нашем случае — значение аргумента `file`). Dreamweaver создал эту переменную, чтобы облегчить себе работу.

Все переменные SQL-запроса, созданные нами или Dreamweaver, отображаются в списке **Variables**. Этот список состоит из трех колонок: **Name** (имя переменной), **Default Value** (значение по умолчанию) и **Run-time Value** (реальное значение, которое будет подставлено во время выполнения). Выше списка находятся кнопки со знаками "плюс" и "минус"; первая создает новую переменную, вторая удаляет выделенную в списке.

Нам нужно задать новое значение по умолчанию для переменной `colname`. Щелкнем мышью по значению, что стоит в колонке **Default Value** (там стоит 1 — значение по умолчанию, подставленное самим Dreamweaver), введем туда 0 и нажмем клавишу <Enter>. Все!

Теперь давайте переключимся в обычный режим диалогового окна **Recordset**, для чего достаточно нажать кнопку **Simple**. И давайте проверим созданный нами набор данных в действии. Нажмем кнопку **Test**, введем в единственное поле ввода появившегося на экране диалогового окна **Please Provide a Test Value** (рис. 8.16) 0 или 1, нажмем кнопку **OK** этого окна и посмотрим на результат. Результат этот будет выведен в диалоговом окне **Test SQL Statement** (см. рис. 8.7).

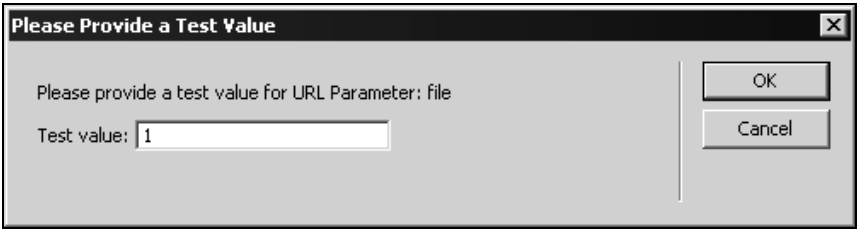


Рис. 8.16. Диалоговое окно **Please Provide a Test Value**

Закроем все диалоговые окна, последовательно нажимая кнопки **OK**. И проверим в действии готовую страницу Categories.php, включив "живой" просмотр. Поскольку аргумент `file` передан этой странице не был, она выведет список категорий статей.

А теперь можно проверить готовые серверные страницы в "боевых" условиях. Сохраним страницу Categories.php и перепишем все содержимое сайта Site2 в корневую папку Web-сервера (как это сделать, было описано в *главе 4*). После этого запустим Web-обозреватель и наберем в строке адреса **http://localhost:8080/**. Когда будет открыта главная страница нашего сайта, последовательно щелкнем по гиперссылкам `файлы` и `статьи` и посмотрим, что получится.

Если мы сделали все правильно, то наша первая серверная Web-страница Categories.php должна правильно обрабатывать разные значения аргумента `file`, выводя списки категорий файлов или статей. Правда, заголовок у нее останется все тем же — `Статьи`, — но это мы со временем исправим.

Разбор PHP-кода, обеспечивающего обработку принятых данных

Но сначала давайте посмотрим, какие изменения внес Dreamweaver в созданные им ранее сценарии PHP.

Собственно, изменениям подвергся один-единственный сценарий, соответствующий поведению **Recordset**. (И это правильно, ведь мы изменяли параметры только набора записей.) Теперь он выглядит так (измененные фрагменты выделены полужирным шрифтом):

```
<?php
$colname_Categories = "0";
if (isset($_GET['file'])) {
  $colname_Categories = (get_magic_quotes_gpc()) ? $_GET['file'] :
  ↳addslashes($_GET['file']);
}
mysql_select_db($database_Site, $Site);
$query_Categories = sprintf("SELECT id, name FROM categories WHERE
↳file = %s ORDER BY id ASC", $colname_Categories);
$Categories = mysql_query($query_Categories, $Site) or
↳die(mysql_error());
$row_Categories = mysql_fetch_assoc($Categories);
$totalRows_Categories = mysql_num_rows($Categories);
?>
```

Первое же выражение объявляет переменную `$colname_Categories` и присваивает ей строковое значение `"0"`. Эта переменная соответствует переменной SQL-запроса `colname`, а `"0"` — это ее значение по умолчанию.

Далее идет условное выражение, извлекающее значение аргумента `file` и присваивающее его переменной `$colname_Categories`. Его стоит рассмотреть подробнее.

Встроенная функция `isset` возвращает `true`, если переданная ей в качестве параметра переменная (или элемент массива, как в нашем случае) объявлена, и `false` в противном случае. Эта функция используется в условии для проверки, существует ли элемент массива `$_GET` с индексом `file`, или, говоря другими словами, передали ли данной странице аргумент `file`.

Если аргумент `file` был передан, то его значение нужно присвоить переменной `$colname_Categories` (если же нет, то эта переменная так и будет содержать значение по умолчанию). Это выполняет выражение

```
$colname_Categories = (get_magic_quotes_gpc()) ? $_GET['file'] :
addslashes($_GET['file']);
```

Здесь возникает проблема, которую Dreamweaver пытается разрешить. Дело в том, что в строковых значениях PHP недопустимы некоторые символы (например, двойные кавычки). А если аргумент `file` будет иметь значение, содержащее такие недопустимые символы, возникнет ошибка. Чтобы в строковых значениях можно было использовать недопустимые символы, их нужно предварить знаком "обратный слеш" (\). Этим и занимается функция `addslashes`.

Но есть и другая проблема. Обработчик PHP может быть настроен таким образом, чтобы при получении данных, переданных методом `GET`, автоматически предварять все недопустимые символы "обратным слешем" (так называемый "параметр magic_quotes_gpc"). Выходит, что нужно еще и проверять, включен этот параметр или отключен; если же он включен, то функцию `addslashes` вызывать не нужно. За это "отвечает" функция, возвращающая логическое значение `true`, если параметр `magic_quotes_gpc` включен, и `false` в противном случае.

Вот поэтому выражение, извлекающее значение аргумента `file` и присваивающее его переменной `$colname_Categories`, такое сложное. Dreamweaver просто старается обойти все "подводные камни", что, надо сказать, у него получается.

Последнее выражение, которое нам нужно рассмотреть, выполняет присвоение кода SQL-запроса переменной `$query_Categories`. Давайте посмотрим на него еще раз:

```
$query_Categories = sprintf("SELECT id, name FROM categories WHERE
file = %s ORDER BY id ASC", $colname_Categories);
```

Здесь используется очередная не знакомая нам встроенная функция `sprintf`. Она занимается тем, что берет строковое значение, переданное ей первым аргументом, ищет в нем так называемые *шаблоны* и заменяет их зна-

чениями своих последующих аргументов: первый шаблон — второй аргумент, второй шаблон — третий аргумент и т. д. Шаблоны имеют вид знака процента (`%`), за которым следует буква, обозначающая тип данных значения, которое должно быть подставлено вместо шаблона.

Посмотрим на первый аргумент этой функции — код запроса SQL. В нем используется всего один шаблон — `%s`, обозначающий строковое значение. Это значение будет взято из второго аргумента функции `sprintf` — переменной `$colname_Categories` — и подставлено вместо шаблона.

Остальной PHP-код этого сценария нам знаком. Так что мы не будем его рассматривать.

Более сложные серверные страницы

Ну все! Основные понятия мы изучили. Теперь пора приниматься за что-нибудь более сложное.

Дел у нас впереди еще много. Страница со списком категорий всегда выводит один и тот же заголовок — `Статьи`, независимо от того, категории статей или файлов она выводит. То же самое с названием (содержимое тега `<TITLE>`). Да какие там заголовки с названиями — мы ведь еще не создали страницу списка статей или файлов, относящихся к выбранной посетителем категории! За работу!..

Написание сценариев PHP вручную

Начнем мы с того, что заставим страницу Categories.php выводить соответствующий заголовок в зависимости от того, категории статей или файлов она содержит. Для этого мы напишем сценарий PHP, который это делает, вручную — Dreamweaver нам в таком случае не помощник.

Итак, откроем страницу Categories.php, если она еще не открыта, и переключимся в режим отображения HTML-кода. Наш первый написанный вручную сценарий мы вставим сразу после сценария, соответствующего серверному поведению **Recordset** (именно там объявлена переменная `$colname_Categories`, которую мы используем), и перед HTML-кодом.

Наш сценарий будет иметь такой вид:

```
<?php
switch ($colname_Categories) {
  case "0":
    $scat = "Статьи";
    break;
  case "1":
```

```
    $scat = "Файлы";
    break;
}
?>
```

В нем нет ничего сложного. Мы использовали выражение выбора для проверки значения переменной `$colname_Categories` и присвоения соответствующего значение переменной `$scat`. А уж значение переменной `$scat` и будет выводиться на окончательную Web-страницу.

Итак, впишем приведенный ранее сценарий сразу же после сценария, соответствующего серверному поведению **Recordset**. Теперь найдем тег `<TITLE>`, содержащий название Web-страницы. Ага, вот он:

```
<TITLE>Статьи</TITLE>
```

и немного его подправим:

```
<TITLE><?php echo $scat; ?></TITLE>
```

чтобы наша страничка имела название, соответствующее случаю.

Осталось найти тег `<H1>`, выводящий на страницу заголовок первого уровня:

```
<H1>Статьи</H1>
```

и исправить его:

```
<H1><?php echo $scat; ?></H1>
```

Если мы теперь переключимся в режим отображения Web-страницы, то увидим, что Dreamweaver вывел вместо заголовка `Статьи` вот такой значок — . Он обозначает сценарий PHP, вставленный в коде HTML.

Обратим внимание, что все сценарии, что находятся перед и после кода HTML, в окне документа при этом никак не отображаются. Чтобы их увидеть, нам будет нужно переключиться в режим отображения HTML-кода.

Теперь сохраним нашу страницу, перепишем все файлы сайта в корневую папку Web-сервера, запустим Web-обозреватель, наберем в строке адреса **http://localhost:8080/** и щелкнем по одной из гиперссылок — `файлы` или `статьи`. Ура — наш сценарий работает!

Тут возникает вопрос: а почему бы не вставить PHP-код, присваивающий значение переменной `$scat`, прямо в сценарий, соответствующий серверному поведению **Recordset**? Это, конечно, можно сделать, но в таком случае мы потеряем возможность работать с этим серверным поведением с помощью панели **Server Behaviors**, потому что Dreamweaver уже не будет считать этот сценарий "своим". Нам придется тогда править код PHP вручную.

Теперь нам нужно подготовить почву для создания серверной Web-страницы, выводящей список статей или файлов, относящихся к данной категории. (Она, как мы решили еще в *главе 5*, будет иметь имя Items.php.)

Для этого нам будет нужно преобразовать каждое название категории в гиперссылку и передать с ее помощью странице Items.php какой-то аргумент. Этот аргумент будет содержать значение, равное значению поля `id` соответствующей записи набора `Categories` (вот нам и пригодилось это поле!). Назовем его `catid`.

Какой вид должен иметь интернет-адрес такой гиперссылки, понятно:

```
Items.php?catid=<значение поля id>
```

Как создавать гиперссылки, мы тоже знаем — об этом говорилось в *главе 3*. Итак, переключаемся в режим отображения Web-страницы, выделяем динамический текст, выводящий название категории, щелчком мыши и вводим в раскрывающийся список **Link** редактора свойств вот что:

```
Items.php?catid=
```

Да-да, именно так! После этого переключаемся в режим отображения кода HTML и выводим на экран панель **Bindings** — проще всего это сделать, нажав комбинацию клавиш <Ctrl>+<F10>.

В панели **Bindings** мы раскрываем "дерево", соответствующее набору записей `Categories`, выбираем щелчком мыши "ветвь" **id**, перетаскиваем ее в значение атрибута `HREF` только что созданной нами гиперссылки и "бросаем" сразу же после знака равенства. После этого HTML-код этой гиперссылки должен принять такой вид:

```
<A HREF="Items.php?catid=<?php echo $row_Categories['id']; ?>">
↳<?php echo $row_Categories['name']; ?></A>
```

Одновременный вывод значений из нескольких полей

Ну что ж, гиперссылка у нас готова. Осталось создать саму страницу Items.php. Выбираем пункт **New** в меню **File**, в списке **Category** диалогового окна **New Document** выбираем пункт **Dynamic Page**, а в правом списке — пункт **PHP**. После этого нажимаем кнопку **Create** — и новая серверная страница готова. Вводим в нее какой-либо поясняющий текст и сохраняем, не закрывая, в корневой папке сайта Site2 под именем Items.php.

Теперь создаем набор записей. Выводим на экран панель **Bindings**, щелкаем по кнопке со знаком "плюс" и выбираем в появившемся меню пункт **Recordset (Query)**. В диалоговом окне **Recordset** задаем такие параметры:

- имя набора данных (поле ввода **Name**) — `Items`;
- имя соединения (раскрывающийся список **Connection**) — `Site`;
- имя таблицы (раскрывающийся список **Table**) — `items`;
- избранные поля таблицы (переключатель **Selected** в группе **Columns**);

- набор записей включает поля `author`, `name`, `added` и `href` (список **Columns**);
- фильтрация по полю `catid` (раскрывающийся список **Filter**);
- значение поля `catid` должно быть равно (пункт = раскрывающегося списка, который находится правее списка **Filter**);
- значению аргумента, переданного методом `GET` (пункт **URL Parameter** раскрывающегося списка, что ниже списка **Filter**);
- имя которого — `catid` (поле ввода правее и ниже списка **Filter**);
- сортировка по полю `added` (раскрывающийся список **Sort**);
- по убыванию значения этого поля (пункт **Descending** раскрывающегося списка, находящегося правее списка **Sort**).

Теперь нажмем кнопку **OK** и полюбуемся на созданный Dreamweaver набор записей — он отображается в виде "дерева" в списке панели **Bindings**.

Далее создадим на странице Items.php таблицу. Она будет иметь две строки (верхняя — строка заголовка), четыре столбца и ширину 100 %. Столбцы этой таблицы будут содержать следующие данные:

- первый — имя автора статьи или разработчика файла;
- второй — название статьи или файла;
- третий — дату добавления;
- четвертый — графическую гиперссылку на статью или файл. Для нее мы используем графическое изображение, хранящееся в файле Arrow.gif.

Дадим этим столбцам заголовки, введя их в соответствующие ячейки первой строки таблицы. Сохраним страницу и продолжим.

Нам уже известно, как создать динамический текст, выводящий данные из какого-либо поля набора записей. Для этого достаточно перетащить в нужное место Web-страницы соответствующую "ветвь" иерархического списка панели **Bindings**. Перетащим "ветвь" **author** в первую ячейку второй строки таблицы, "ветвь" **name** — во вторую, а "ветвь" **added** — в третью. "Ветвь" **href** пока оставим в покое.

Теперь поместим в четвертую ячейку второй строки таблицы графическое изображение из файла Arrow.gif, выделим его щелчком мыши и превратим в гиперссылку. В раскрывающийся список **Link** редактора свойств введем любой пришедший в голову интернет-адрес — все равно мы его потом удалим. Переключимся в режим отображения HTML-кода, найдем код, описывающий только что созданную гиперссылку, и удалим все содержимое атрибута `HREF` тега `<A>`, оставив сам этот атрибут и сам этот тег. Из панели **Bindings** перетащим в кавычки, в которых заключается значение атрибута `HREF`, "ветвь" **href** и полюбуемся на результат:

```
<A HREF="<?php echo $row_Items['href']; ?>"><IMG SRC="Arrow.gif"></A>
```

(Тег `IMG` может содержать другие атрибуты, но они здесь не приведены.)

Ну и, разумеется, нужно создать повторяющуюся область, чтобы наша страница вывела все записи набора `Items`. Поставим текстовый курсор в любую ячейку второй строки таблицы и щелкнем кнопку **<tr>** в секции тегов строки статуса, чтобы выделить всю эту строку. Нажмем комбинацию клавиш <Ctrl>+<F9>, чтобы вывести на экран панель **Server Behaviors**, нажмем кнопку со знаком "плюс" этой панели и в появившемся на экране меню выберем пункт **Repeat Region**.

В появившемся на экране диалоговом окне **Repeat Region** задаем следующие параметры повторяющейся области:

- имя набора данных (раскрывающийся список **Recordset**) — `Items`;
- вывод всех записей набора (переключатель **All records** группы **Show**).

Потом нажмем кнопку **OK** и сохраним страницу.

Теперь проверим, что мы сотворили? Перепишем все файлы сайта в корневую папку Web-сервера, откроем Web-обозреватель и наберем в строке адреса **http://localhost:8080/**. Когда Web-обозреватель откроет главную страницу нашего сайта, проследуем по гиперссылкам до любой страницы списка статей или файлов. Если мы все сделали правильно, то должны увидеть вот что — см. рис. 8.17.

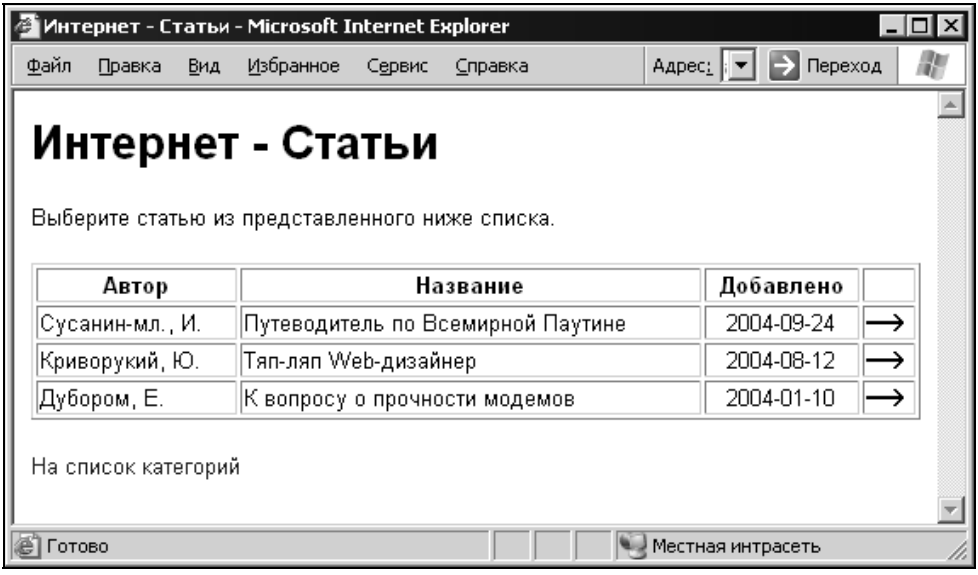


Рис. 8.17. Страница списка статей или файлов Items.php

Правда, особого повода радоваться нет. Во-первых, значения даты выводятся во внутреннем формате PHP — "год-месяц-число", а для нас более привычен порядок "число.месяц.год". Во-вторых, и заголовок на странице, и ее

название опять "оторваны от действительности". В-третьих, было бы замечательно, если бы гиперссылка внизу указывала на страницу со списком категорий, соответственно, статей или файлов.

И начнем мы с заголовка и названия нашей страницы.

Страницы с несколькими наборами данных

Чтобы подставить в текст страницы название категории, статьи (или файлы) из которой на ней сейчас отображаются, нам нужно откуда-то его извлечь. А извлечь мы его можем только из таблицы `categories` базы данных `site`. Это значит, что нам нужно создать на странице Items.php еще один набор записей, который будет содержать всего одну запись — ту, значение поля `id` которой равно значению атрибута `catid`, переданного методом `GET`.

Нажмем комбинацию клавиш <Ctrl>+<F10>, чтобы вывести на экран панель **Bindings**, щелкнем по кнопке со знаком "плюс" и выберем в появившемся меню пункт **Recordset (Query)**. В диалоговом окне **Recordset** зададим такие параметры:

- имя набора данных (поле ввода **Name**) — `Category`;
- имя соединения (раскрывающийся список **Connection**) — `Site`;
- имя таблицы (раскрывающийся список **Table**) — `categories`;
- избранные поля таблицы (переключатель **Selected** в группе **Columns**);
- набор записей включает поля `name` и `file` (список **Columns**);
- фильтрация по полю `id` (раскрывающийся список **Filter**);
- значение поля `id` должно быть равно (пункт = раскрывающегося списка, который находится правее списка **Filter**);
- значению аргумента, переданного методом `GET` (пункт **URL Parameter** раскрывающегося списка, что ниже списка **Filter**);
- имя которого — `catid` (поле ввода правее и ниже списка **Filter**);
- без сортировки (пункт **None** раскрывающегося списка **Sort**).

Чтобы создать набор записей, нажмем кнопку **OK**. В иерархическом списке панели **Bindings** появится новое "дерево".

А дальше совсем просто. Переключаемся в режим отображения HTML-кода и ищем тег `<TITLE>`, задающий название страницы. Вот он:

```
<TITLE>Интернет - Статьи</TITLE>
```

Удаляем все его содержимое, выделяем "ветвь" **name** в "дереве" **Recordset (Category)** списка панели **Bindings**, перетаскиваем ее и "бросаем" прямо между открывающим и закрывающим тегами `<TITLE>`. Должно получиться вот что:

```
<TITLE><?php echo $row_Category['name']; ?></TITLE>
```

То же самое проделываем с тегом `<H1>`, создающим заголовок на странице.

Теперь давайте заставим после названия категории отображаться в кавычках слово `статьи` или `файлы`. Для этого мы вставим в код страницы вот такой, уже знакомый нам сценарий:

```
<?php
switch ($row_Category['file']) {
  case "0":
    $scat = "статьи";
    break;
  case "1":
    $scat = "файлы";
    break;
}
?>
```

поместив его сразу после сценария, соответствующего серверному поведению **Recordset**. Этот сценарий извлечет значение из поля `file` набора записей `Category` и, в зависимости от его значения, присвоит переменной `$scat` строку `статьи` или `файлы`.

Осталось только дополнить сценарий, который мы вставили в содержимое тегов `<TITLE>` и `<H1>`. Он должен выглядеть так:

```
<TITLE><?php echo $row_Category['name'] . " (" . $scat . ")"; ?></TITLE>
```

(это сценарий для тега `<TITLE>`; его "коллега" для тега `<H1>` будет таким же).

Последнее, что мы сделаем, — это исправим находящуюся внизу страницы гиперссылку, которая указывает на страницу списка категорий. Сделаем так, чтобы при щелчке по ней, посетитель попадал со списка файлов на список категорий файлов, а со списка статей — на список категорий статей. Для этого нам нужно дать ее вот такой интернет-адрес:

```
Categories.php?file=<1 — для файлов, 0 — для статей>
```

А значение для аргумента `file` мы можем взять из поля `file` набора записей `Category`.

Итак, выделим строку `На список категорий` и наберем в раскрывающемся списке **Link** редактора свойств

```
Categories.php?file=
```

Потом переключимся в режим отображения HTML-кода, найдем код только что созданной гиперссылки и в значении атрибута `HREF` тега `<A>` сразу после знака равенства допишем вручную то, что далее выделено полужирным шрифтом:

```
Categories.php?file=<?php echo $row_Category['file']; ?>
```

В конце концов, мы уже имеем достаточный опыт программирования на PHP, чтобы дописать какие-то мелочи вручную! Можно, конечно, перетащить туда "ветвь" **file** "дерева" **Category** из панели **Bindings**, но это неинтересно...

Вот и все! Теперь можно опубликовать наш сайт на локальном Web-сервере и испытать в Web-обозревателе. Попробуйте сделать это без подсказки — пора привыкать к самостоятельности.

Правильный вывод значений даты

Теперь можно заняться значениями даты. Заставим их выводиться не так, как хочет PHP, а как нам привычно, — в формате "число.месяц.год".

PHP поддерживает встроенную функцию `date`, которая принимает два аргумента. Первый аргумент — это строковое значение, описывающее формат, в котором должна выводиться дата, а второй — это само значение даты, которое нужно вывести, в числовом формате. Возвращает эта функция строку, содержащую отформатированное значение даты.

Давайте посмотрим на сценарий, соответствующий динамическому тексту, который выводит значение даты:

```
<?php echo $row_Items['added']; ?>
```

Дополним его функцией `date`:

```
<?php echo date("d.m.Y", strtotime($row_Items['added'])); ?>
```

Здесь первым аргументом мы передали в функцию строку "d.m.Y", которая описывает нужный нам формат "число.месяц.год". Эта строка содержит шаблоны, вместо которых PHP подставит значения числа, номера месяца и года. Всего этих шаблонов три:

- "d" — вывод числа двумя цифрами. Если число состоит из одной цифры, оно будет выведено с нулем. Так, для 2-го мы получим на выходе 02, а для 21-го — 21;
- "m" — вывод номера месяца, опять же, двумя цифрами. Если номер месяца состоит из одной цифры, он будет выведен с нулем;
- "Y" — вывод года четырьмя цифрами.

И еще. Поскольку функция `date` принимает второй аргумент в числовом формате, а в массиве `$row_Items` оно хранится в строковом, нам придется вызвать еще и встроенную функцию `strtotime`, преобразующую строковое значение даты в числовое.

Вписываем дополненный сценарий в HTML-код страницы на место старого. Сохраняем страницу и проверяем ее в Web-обозревателе. Вот теперь значения даты выводятся нормально.

Условный вывод элементов Web-страницы

Всем наша страница Items.php хороша. За одним только исключением — если набор не будет содержать ни одной записи (в выбранной посетителем категории нет ни одного файла или статьи), таблица все равно будет выведена, причем очень некрасиво. Давайте что-то с этим делать.

Напрашивается такое решение — проверяем, есть ли в наборе записи:

```
if ($row_Items = mysql_fetch_assoc($Items)) {
```

и выводим таблицу, содержащую список. Если же набор записей пуст:

```
} else {
```

выводим предупреждение вида: `Извините, администратор не удосужился заполнить эту категорию статьями (файлами).`

Именно так мы бы и сделали, если бы писали код нашей Web-страницы вручную. Но, поскольку мы используем замечательный пакет Dreamweaver, то давайте воспользуемся его услугами. Тем более что он может нам в этом помочь, предоставляя возможность создавать так называемые *необязательные области* страницы, выводимые при выполнении какого-либо условия.

Итак, давайте вставим прямо под таблицей, выше гиперссылки, указывающей на список категорий, коротенький абзац `Список пуст.`. После этого выделим строку `Выберите статью из представленного ниже списка.` и таблицу и создадим из всего этого необязательную область, отображаемую, если набор записей не пуст.

Привычным уже движением нажмем комбинацию клавиш <Ctrl>+<F9>, чтобы вывести на экран панель **Server Behaviors**. В этой панели нажмем кнопку со знаком "плюс", выберем в появившемся на экране меню пункт **Show Region**, а в появившемся следом подменю — пункт **Show If Recordset Is Not Empty**. На экране появится диалоговое окно **Show If Recordset Is Not Empty** (рис. 8.18).

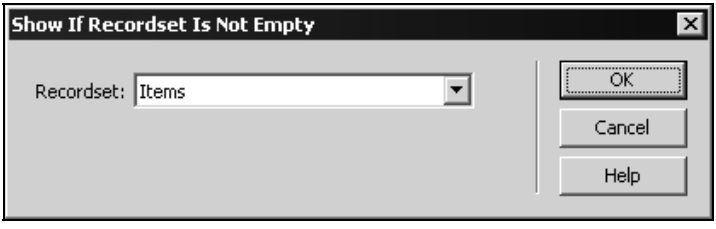


Рис. 8.18. Диалоговое окно **Show If Recordset Is Not Empty**

Единственное, что нам будет нужно сделать в этом окне, — это выбрать в раскрывающемся списке **Recordset** наш набор записей `Items`. И можно нажимать кнопку **OK**. Созданная нами необязательная область показана на рис. 8.19.

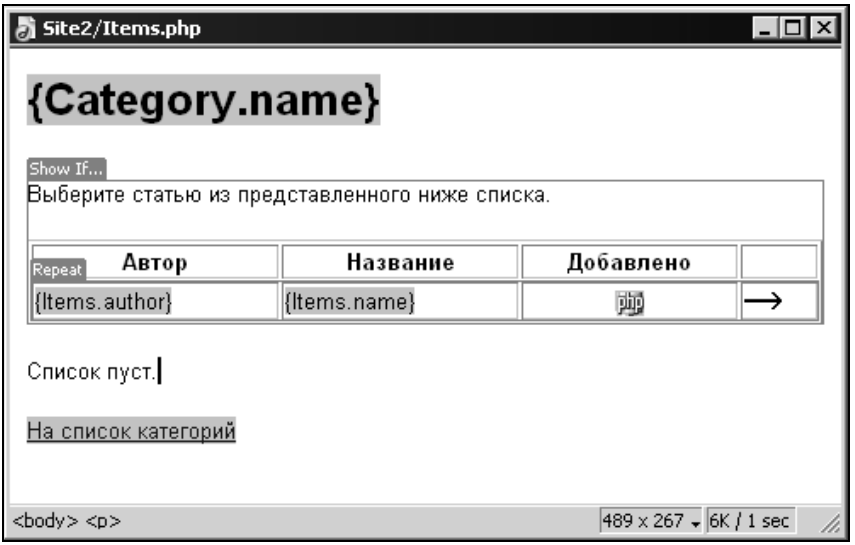


Рис. 8.19. Страница Items.php с необязательной областью

Теперь выделим абзац `Список пуст.`. Мы превратим его в необязательную область, отображаемую, если набор записей, наоборот, не пуст.

Нажмем кнопку со знаком "плюс" панели **Server Behaviors**, выберем в появившемся на экране меню пункт **Show Region**, а в появившемся следом подменю — пункт **Show If Recordset Is Empty**. На экране появится диалоговое окно **Show If Recordset Is Empty**, очень похожее на уже знакомое нам окно **Show If Recordset Is Not Empty**. Выберем в раскрывающемся списке **Recordset** этого окна все тот же набор записей `Items` и нажмем кнопку **OK**.

Если мы теперь опубликуем наш сайт на локальном Web-сервере и проверим его в Web-обозревателе, на страницах, отображающих содержимое "пустых" категорий, будет выводиться текст `Список пуст.`. И никаких корявых таблиц из одной строки!

Если мы теперь переключимся в режим отображения HTML-кода нашей страницы и найдем сценарий, соответствующий серверному поведению **Show If Recordset Is Empty**:

```
<?php if ($totalRows_Items == 0) { // Show if recordset empty ?>
<P>Список пуст.</P>
<?php } // Show if recordset empty ?>
```

то удивимся — ведь Dreamweaver почти угадал наши мысли! Правда созданный им сценарий проверяет на равенство нулю значение переменной `$totalRows_Items`, которая была объявлена в сценарии, соответствующем серверному поведению **Recordset**:

```
$totalRows_Items = mysql_num_rows($Items);
```

Когда мы разбирали этот сценарий, то еще подумали, что это выражение лишнее — ведь переменная `$totalRows_Items` тогда нигде не использовалась. Оказывается, она все-таки пригодилась нам!

Что касается сценария, соответствующего серверному поведению **Show If Recordset Is Not Empty**, то в нем проверяется, больше ли значение переменной нуля:

```
<?php if ($totalRows_Items > 0) { // Show if recordset not empty ?>
<P>Выберите статью из представленного ниже списка.</P>
. . .
<?php } // Show if recordset not empty ?>
```

Вот и все о необязательных областях.

Вывод сведений о наборе записей

Последнее, что мы сделаем со страницей Items.php, — это заставим ее показывать количество статей или файлов в выбранной категории, то есть количество записей в наборе. (Вообще-то, это необязательно, но пусть будет.)

Количество записей в наборе будет отображаться в отдельной строке, находящейся под таблицей со списком статей (файлов). Очевидно, что эта строка должна попадать в созданную нами ранее необязательную область, выводимую на экран, если набор содержит записи. Поэтому нам нужно как-то "втиснуть" эту строку в необязательную область. И проще всего сделать это в режиме отображения HTML-кода.

Итак, переключимся в режим отображения HTML-кода и найдем место, где кончается первая необязательная область, выводимая, если набор содержит записи. Вот код HTML, описывающий это место:

```
  <?php } while ($row_Items = mysql_fetch_assoc($Items)); ?>
</TABLE>
<?php } // Show if recordset not empty ?>
<?php if ($totalRows_Items == 0) { // Show if recordset empty ?>
```

(Правда, тут больше PHP, чем HTML.) Вставляем сразу после тега `</TABLE>` тег `<P>`, содержащий текст `Всего записей:` (+обязательный пробел!). После этого код будет выглядеть так (вставленный фрагмент выделен полужирным шрифтом):

```
  <?php } while ($row_Items = mysql_fetch_assoc($Items)); ?>
</TABLE>
<P>Всего записей: </P>
<?php } // Show if recordset not empty ?>
<?php if ($totalRows_Items == 0) { // Show if recordset empty ?>
```

Теперь у нас есть текстовый абзац, входящий в ту же необязательную область, что и таблица со списком. Осталось вставить в него сценарий PHP, выводящий количество записей в наборе. И для этого мы опять воспользуемся услугами Dreamweaver.

Переключимся вновь в режим отображения Web-страницы и поставим текстовый курсор в конце только что созданного абзаца, сразу после пробела. Выведем на экран панель **Server Behaviors**, нажмем на ней кнопку со знаком "плюс", выберем в появившемся на экране меню пункт **Display Record Count**, а в появившемся следом подменю — пункт **Display Total Records**. На экране появится небольшое диалоговое окно **Display Total Records** (рис. 8.20).

Рис. 8.20. Диалоговое окно **Display Total Records**

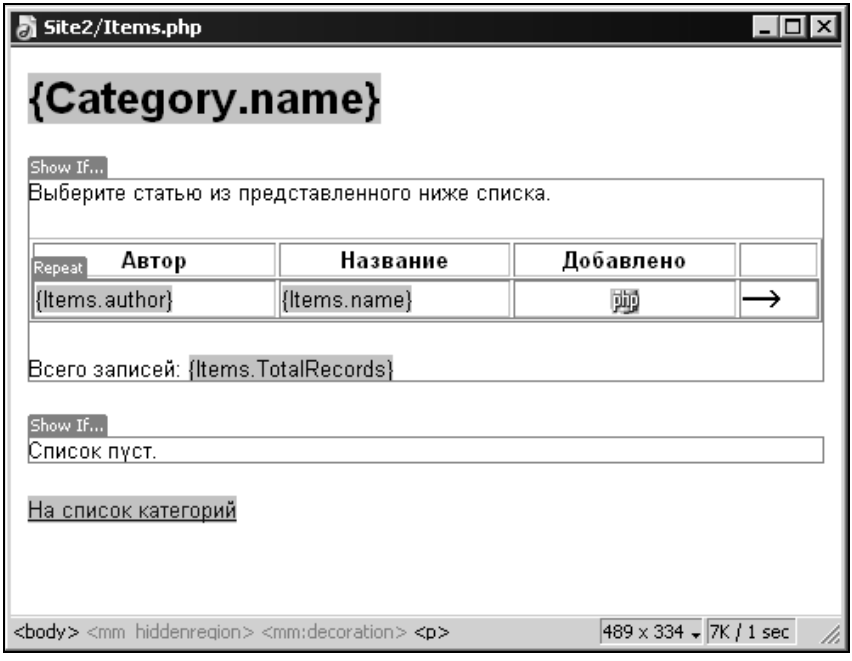


Рис. 8.21. Страница Items.php с необязательной областью и строкой количества записей в наборе

В раскрывающемся списке **Recordset** выбираем набор записей `Items`, количество записей в котором нам нужно показывать на странице. И нажмем кнопку **OK**. Теперь наша серверная Web-страница Items.php будет выглядеть так, как показано на рис. 8.21.

Осталось только проверить готовую страницу в действии.

Если мы посмотрим на сценарий PHP, соответствующий только что созданному серверному поведению **Display Total Records**, то увидим, что он очень прост:

```
<P>Всего записей: <?php echo $totalRows_Items ?></P>
```

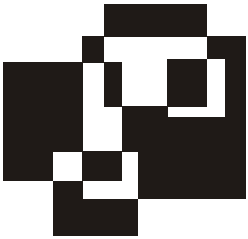
Здесь содержимое переменной `$totalRows_Items`, объявленной в сценарии, соответствующем серверному поведению **Recordset**, выводится на экран. Эта переменная снова нам пригодилась!

Что дальше?

Мы только что сделали простейший Web-сайт, использующий активные серверные Web-страницы. И, в отличие от предыдущего, "статичного" сайта, мы обошлись всего тремя страницами — default.htm, Categories.php и Items.php.

На этом можно считать, что простейшие серверные страницы в среде Dreamweaver мы создавать научились. Теперь очередь — за сложными страницами и сложными сценариями, реализующими ввод и правку данных в базе `site`. Нам ведь понадобится заносить в список новые статьи и файлы, исправлять ошибки и удалять те из них, что прекратили свое существование в Сети. А иначе наш сайт очень быстро потеряет актуальность, а за ней — и всех посетителей...

Созданию серверных страниц, предназначенных для добавления, изменения и удаления данных, будет посвящена следующая глава. Не закрываем Dreamweaver, не останавливаем ни MySQL, ни Apache, ведь наш простейший сайт еще далеко не готов.



Глава 9

Реализация ввода и правки данных

Что ж, с выводом данных из базы на Web-страницу мы разобрались в *главе 8*. Настала пора выяснить, каким образом с помощью HTML и PHP можно реализовать ввод новых данных в базу и правку уже существующих. И создать, наконец, страницы для ввода данных в базу `site`, чтобы не пользоваться для этого программой-клиентом данных.

Хоть в конце *главы 8* Web-страницы, реализующие ввод и правку данных, и были названы сложными, но ничего особо сложного в них нет. Нужно просто уяснить некоторые принципы, на основе которых они строятся, и изучить серверные поведения Dreamweaver, которые используются для их создания. А также нужно знать средства языка HTML, с помощью которых реализуется сам ввод данных посетителем.

Сначала, как обычно, — небольшой теоретический курс. В нем мы как раз и изучим основные принципы приема данных от посетителя и передачи их серверной программе. Так что можно пока оставить компьютер в покое.

Как реализуется ввод и передача данных

Вообще, чтобы создать Web-страницы, принимающие данные от посетителя и обрабатывающие их (в нашем случае — заносящие их в базу данных), нужно решить три проблемы:

- собственно, принять эти данные;
- закодировать их на стороне Web-обозревателя;
- передать их в закодированном виде серверной программе.

Раскодирование принятых данных выполняется самим обработчиком PHP без нашего участия, так что нам этим заниматься не придется.

Замечание

Поскольку технология PHP предназначена именно для создания серверных программ, она имеет встроенные средства для раскодирования принятых данных. Если же серверная программа пишется с использованием других технологий (например, на языках C++ или Pascal), то их раскодированием придется заниматься самому программисту.

Давайте же рассмотрим, как решаются эти проблемы с помощью HTML и PHP.

Ввод данных. Формы

Как реализуется ввод данных в обычных приложениях Windows? Очень просто — с помощью окон и элементов управления: полей ввода, списков, флажков, переключателей и кнопок. На экране появляется окно, содержащее элементы управления, в которые нам будет нужно ввести какие-то данные. Введя их, мы нажимаем кнопку **OK**, и программа приступает к обработке введенных данных. Все это мы уже наблюдали на примере того же Dreamweaver.

А как же в случае Web-страниц? Точно так же! Web-обозреватель выводит страницу, содержащую набор элементов управления, точно таких же, какие используются в приложениях Windows. И мы вводим в них свои данные.

Как элементы управления Windows-программы обязательно должны находиться в окне, так и элементы управления Web-страницы должны быть помещены в форму. *Форма* — это особый элемент страницы, своего рода вместилище для элементов управления, данные из которых должны быть посланы Web-серверу. Пример такой формы можно видеть на рис. 9.1; эта форма с двумя полями ввода используется для задания имени и пароля пользователя перед входом на защищенный сайт.

Рис. 9.1. Пример формы

Внимание

Все элементы управления, используемые для ввода данных, которые должны быть отправлены Web-серверу, обязательно должны находиться в форме.

Сам процесс отправки данных начинается после того, как посетитель нажмет особую кнопку. Эта кнопка носит название **Отправить** (Submit — в англоязычных программах) и обязательно должна присутствовать в форме.

Также в форме может присутствовать кнопка **Сброс** (Reset), обнуляющая введенные посетителем данные. Как правило, обе эти кнопки располагаются в самом низу формы.

Форма создается с помощью особого парного тега `<FORM>`, внутри которого помещаются теги, создающие элементы управления.

```
<FORM ACTION="givemedata.php">
. . .
</FORM>
```

Обязательный атрибут `ACTION` этого тега задает имя серверной программы, которая должна получить введенные данные.

Большинство элементов управления создается с помощью одинарного тега `<INPUT>`. Этот тег имеет обязательные атрибуты `TYPE` и `NAME`. Первый атрибут задает тип элемента управления — поле ввода, флажок или кнопка, — а второй — его уникальное имя.

Давайте рассмотрим HTML-код, создающий небольшую форму, наподобие той, что показана на рис. 9.1:

```
<FORM ACTION="givemedata.php">
  <P>Имя: <INPUT TYPE="text" NAME="name"></P>
  <P>Пароль: <INPUT TYPE="text" NAME="password"></P>
  <P><INPUT TYPE="submit" NAME="submit"></P>
</FORM>
```

Первая строка этого кода нам уже знакома — она создает форму, которая отправит введенные данные серверной странице givemedata.php. Все это мы уже знаем.

Вторая и третья строки создают поля ввода для задания, соответственно, имени пользователя и его пароля. Видно, что теги `<INPUT>`, создающие их, имеют атрибуты `TYPE` со значениями `text`, то есть поле ввода.

Четвертая строка создает кнопку **Отправить**. Здесь атрибут `TYPE` тега `<INPUT>` уже должен иметь значение `submit`.

Для задания имен элементов управления может также использоваться тег `ID`:

```
<INPUT TYPE="text" ID="name">
```

Но, поскольку он поддерживается не всеми Web-обозревателями, на практике в теге `<INPUT>` обычно используют оба атрибута — и `NAME`, и `ID`. Вот так:

```
<INPUT TYPE="text" NAME="name" ID="name">
```

Все теги HTML, используемые для создания разных элементов управления, мы рассмотрим далее в этой главе. Сейчас же давайте выясним, каким образом решается вторая проблема.

Кодирование данных

Прежде чем данные будут отправлены Web-серверу и через него серверной программе, они должны быть особым образом закодированы. Кодирование данных выполняется формой, точнее, Web-обозревателем на основе значений особых атрибутов тега `<FORM>`. Давайте выясним, как именно кодируются данные и какие атрибуты за это "отвечают".

Итак, можно сказать, что введенные посетителем данные кодируются формой. Форма извлекает из элементов управления введенные данные и имена, заданные атрибутами `NAME` и `ID` тега `<INPUT>`, и объединяет их в пары вида `<имя элемента управления>=<значение>`. Например, на основе данных, введенных в нашу форму, Web-обозреватель создаст вот такой набор:

```
name=admin
password=superuser
submit=1
```

Так что серверная программа сможет узнать, какие данные в какой элемент управления были введены.

Что касается кнопки **Отправить**, то она всегда получает значение, равное 1, при ее нажатии. На практике это значение никогда не используется — серверной программе и так ясно, что кнопка **Отправить** была нажата, иначе эта программа не получила бы данные от формы.

После этого форма выполняет преобразование символов. Так, символ пробела заменяется символом `+`. Все символы, не являющиеся цифрами, латинскими буквами и знаками подчеркивания, преобразуются в последовательности вида `%NN`, где `NN` — шестнадцатеричный код символа.

А уже после этого форма выполняет кодирование данных согласно заданному набору правил — *методу кодирования*. Метод кодирования задается с помощью особого атрибута `ENCTYPE` тега `<FORM>`.

```
<FORM ACTION="givemedata.php"
↳ENCTYPE="application/x-www-form-urlencoded">
. . .
</FORM>
```

По умолчанию (если атрибут `ENCTYPE` не указан) выполняется кодирование данных с помощью метода `application/x-www-form-urlencoded`. Этот метод применяется в подавляющем большинстве случаев. Большинством Web-обозревателей также поддерживаются методы кодирования `multipart/form-data` и `text/plain`, но они используются значительно реже.

Замечание

Метод кодирования `multipart/form-data` используется, если нужно отправить серверной программе файлы; он обеспечивает соответствующее такому

случаю преобразование двоичных данных. А метод `text/plain` представляет данные в виде обычного текста, что может быть полезно, если данные формы будут отправляться по электронной почте (иногда используется и такой способ передачи данных).

Хотелось бы отметить, что кодирование данных с помощью одного из перечисленных ранее методов применяется не всегда. Мы поговорим об этом, когда рассмотрим решение третьей проблемы.

Передача данных

Для передачи данных по Сети форма (разумеется, не сама форма, а обрабатывающий ее Web-обозреватель) может использовать один из двух методов передачи данных. Это уже знакомый нам метод `GET` и пока еще не известный метод `POST`. Рассмотрим их по порядку.

Метод `GET` мы изучили в *главе 8*, когда разбирали способ передать данные от одной Web-страницы другой. Передаваемые этим методом данные представляются в виде набора пар *<имя элемента управления>=<значение>*, который ставится в самый конец интернет-адреса и отделяется от него вопросительным знаком. Сами же пары разделяются друг от друга знаком "коммерческое и" (&).

Давайте возьмем набор данных, сформированный Web-обозревателем на основе введенных в нашу форму данных:

```
name=admin
password=superuser
submit=1
```

и отправим его серверной программе методом `GET`:

```
givemedata.php?name=admin&password=superuser&submit=1
```

Никакого кодирования данных в этом случае не применяется. Это значит, что атрибут `ENCTYPE` тега `<FORM>` в случае отправки данных методом `GET` можно опустить — он все равно не будет обрабатываться.

Простота и наглядность представления данных — единственное преимущество метода `GET`. Поэтому значительно упрощается отладка серверных программ: поскольку передаваемый Web-серверу адрес всегда отображается в строке адреса Web-обозревателя, мы всегда сможем увидеть, что именно было передано. (Понятно, что конфиденциальные данные таким методом не передашь — их увидят все, кто стоит у нас за спиной.)

Недостатков же у метода `GET` два, и оба очень серьезные. Первый: с его помощью невозможно передавать большие объемы данных. Это происходит из-за ограничения, накладываемого стандартами на длину интернет-адреса: не более 256 символов. Вычтем отсюда длину интернет-адреса самой сер-

верной программы — и получим максимально допустимый размер наших данных. Второй недостаток метода GET — обратная сторона его достоинства. Данные, пересылаемые им, открыты для всеобщего обозрения и могут быть легко прочитаны в строке адреса Web-обозревателя.

Метод GET используется, если пересылаемые серверной программе данные заведомо невелики и не являются секретными. В основном, это внутренние данные Web-сайта: номера категорий (как на нашем сайте — см. *главу 8*), различные коды, ключи и пр. Если же нам нужно передавать объемные либо конфиденциальные данные, мы используем второй метод передачи, называемый POST.

Метод POST передает данные серверной программе уже не в виде части интернет-адреса, а в так называемых дополнительных данных клиентского запроса. Поскольку размер этих дополнительных данных не ограничен (по крайней мере, он может быть очень велик), мы можем передать все что угодно, в каких угодно количествах. Способом POST можно передать Web-серверу даже файл.

В случае передачи данных методом POST кодирование данных выполняется. И атрибут ENCTYPE тега <FORM> всегда обрабатывается, если, конечно, он присутствует в HTML-коде. (Если же атрибут ENCTYPE опущен, используется метод кодирования данных по умолчанию — application/x-www-form-urlencoded.)

Достоинства у метода POST два: отсутствие ограничения на объем передаваемых данных (как мы уже выяснили) и их "невидимость". Недостатки: сложность раскодирования данных и трудность отладки. Но, поскольку раскодированием данных за нас занимается обработчик PHP, а трудностей настоящие программисты не боятся (так ведь?), этими недостатками можно пренебречь.

Методом POST передаются объемные данные, которые должны быть занесены в базу данных сайта, или секретные сведения (имена и пароли). Поэтому на наших Web-страницах, выполняющих ввод и правку данных, мы будем использовать именно метод POST.

Замечание

Говорят, что комитет W^{3}C намерен со временем заставить Web-дизайнеров и Web-программистов вообще отказаться от метода GET и использовать только метод POST. Пока что метод GET просто объявлен не рекомендованным (deprecated) для использования во вновь создаваемых сайтах. Однако все выпускаемые в настоящее время Web-обозреватели поддерживают метод GET, поэтому он все еще в деле.

Ну, вот и все! Теория закончилась, и, как обычно, наступает время практики. Пора браться за Dreamweaver.

Административные и пользовательские Web-страницы

Но сначала давайте пока отвлечемся от ввода и правки данных и подумаем о другом. О чем? О безопасности и разграничении доступа.

Безопасность нашей базы данных мы обеспечили еще в *главе 6*, когда ее, собственно, и создали. Пользователь `site`, от имени которого наши серверные страницы подключаются к базе, имеет доступ к таблицам `categories` и `items` на чтение и запись данных, но создавать таблицы не может. Этого должно быть достаточно, чтобы защитить нашу базу. В конце концов, у нас есть еще пользователь `root`, который имеет права на все (права администратора сервера).

Но дело в том, что далеко не все посетители нашего сайта должны иметь доступ к базе данных на запись. Обычные посетители, которые просто просматривают страницы со списками статей и файлов, не должны иметь возможность что-то в них вписать. Права на запись должен иметь только администратор сайта, то есть мы, поскольку именно нам предстоит пополнять списки и исправлять в них ошибки (администрировать сайт).

Так как же разграничить полномочия обычных "бесправных" пользователей (как говорят профессиональные программисты, *гостей*) и администраторов сайта? Очень просто. Нужно создать два набора Web-страниц разного назначения, так называемые *административные* и *пользовательские* страницы.

Пользовательские Web-страницы должны обеспечивать только просмотр данных гостями с максимальным для них комфортом. Эти страницы мы уже создали в *главе 8* — Categories.php и Items.php. Что касается страниц административных, то они должны предоставлять возможность добавления и правки данных также с максимальным комфортом. Вот эти страницы нам еще нужно создать.

Давайте создадим в корневой папке нашего сайта папку Admin (как это сделать, описано в *главе 4*). В эту папку мы поместим все административные страницы, чтобы отделить их от пользовательских.

Для работы со списком категорий мы создадим в папке Admin страницу Categories_admin.php. Эта страница будет аналогичная уже созданной странице Categories.php, за тем исключением, что таблица в ней будет содержать три столбца. Первый столбец — это, как и на странице Categories.php, название категории, а остальные два будут содержать гиперссылки `Изменить` и `Удалить`, указывающие на страницы, соответственно, правки данной записи и ее удаления. Также нам будет нужно добавить в эту таблицу еще одну строку, не входящую в повторяющуюся область; последняя ячейка этой строки будет содержать гиперссылку `Добавить`, указывающую на страницу добавления новой записи. Примерный вид страницы Categories_admin.php приведен на рис. 9.2.

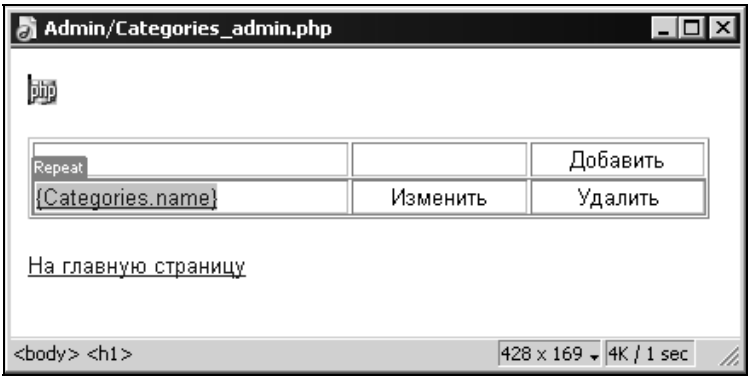


Рис. 9.2. Вид Web-страницы Categories_admin.php в окне документа Dreamweaver

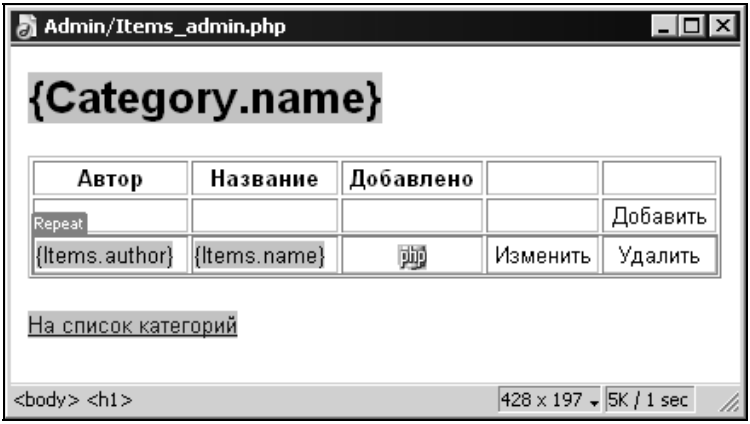


Рис. 9.3. Вид Web-страницы Items_admin.php в окне документа Dreamweaver

Ну, а для работы со списком статей (файлов) выбранной категории мы создадим Web-страницу Items_admin.php, также в папке Admin. Эта страница ничем не будет отличаться от уже созданной нами Items.php, за несколькими исключениями. Во-первых, мы добавим в набор записей `Items` еще одно поле — `id`. Во-вторых, мы уберем столбец `Открыть`, а вместо него добавим столбцы `Изменить` и `Удалить`, по аналогии со страницей Categories_admin.php. В-третьих, мы добавим в таблицу еще одну строку, не входящую в повторяющуюся область, и поместим в ее последнюю ячейку гиперссылку `Добавить`. В-четвертых, удалим обе необязательные области, причем вторую, отображаемую при пустом наборе записей, — вместе с содержимым. В-пятых, мы можем удалить и строку со сведениями о количестве записей в наборе (хотя можем и не удалять). В результате должно получиться вот что — см. рис. 9.3.

Обратим внимание, что столбца `Интернет-адрес` таблица на Web-странице Items_admin.php не содержит. Дело в том, что интернет-адрес может быть

весьма длинным, и тогда наша таблица перестанет помещаться в окно Web-обозревателя, так что лучше на странице Items_admin.php его не отображать. Если же нам будет нужно его увидеть, мы всегда сможем перейти на страницу правки записи, где будут отображены значения всех ее полей.

Разумеется, созданные нами административные страницы должны содержать гиперссылки, указывающие друг на друга, а не на их пользовательских "коллег". Вообще, административные страницы не должны ссылаться на пользовательские, и наоборот.

Создав административные страницы списков категорий и статей (файлов), можно приступать к созданию страниц для их добавления, правки и удаления. Этим мы сейчас и займемся.

Создание простых серверных Web-страниц для ввода и правки данных

Сначала мы создадим набор административных Web-страниц, предназначенных для добавления, правки и удаления категорий — они проще. Страницами для добавления, правки и удаления статей и файлов мы займемся потом.

Простая страница для добавления записи

И начнем мы с Web-страницы, предназначенной для добавления новой категории. Пусть она называется Category_add.php. С ее помощью мы сможем ввести название для новой категории, а также задать к статьям или файлам она относится.

Итак, создадим новую серверную страницу PHP и сохраним под именем Category_add.php в папке Admin. Дадим ей название `Добавление категории`, введем такой же заголовок и какой-либо поясняющий текст. И создадим первую в нашей карьере интернет-программиста форму.

Чтобы создать форму, выберем пункт **Form** подменю **Form** меню **Insert**. Созданная нами пока еще пустая форма будет выглядеть так, как показано на рис. 9.4.

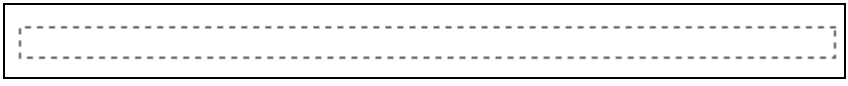


Рис. 9.4. Пустая форма на Web-странице Category_add.php

Поставим в эту форму текстовый курсор и посмотрим на редактор свойств. Он должен выглядеть так, как показано на рис. 9.5.

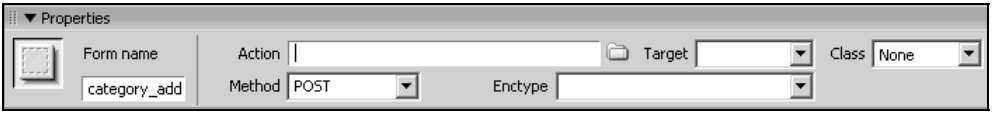


Рис. 9.5. Вид редактора свойств при выделенной форме

Здесь нас интересует только поле ввода **Form name**, в котором задается уникальное в пределах Web-страницы имя формы. Дадим нашей форме "говорящее" имя `category_add` — так нам будет проще разобраться в своих собственных страницах. Вводим это имя и нажимаем клавишу <Enter>, чтобы Dreamweaver внес соответствующие изменения в код HTML.

Замечание

Остальные элементы управления редактора свойств, показанного на рис. 9.5, задают значения соответствующих атрибутов тега `<FORM>`. Мы не будем их вводить — это сделает сам Dreamweaver, когда будет создавать серверное поведение для добавления новой записи.

Теперь снова поставим текстовый курсор в форму, введем текст `Название:` и поставим пробел. После этого пробела мы создадим поле ввода, предназначенное для ввода названия новой категории.

Поле ввода создается выбором пункта **Text Field** подменю **Form** меню **Insert**. Созданное поле ввода тотчас появится в том месте, где стоит текстовый курсор, то есть сразу после надписи `Название:` (+пробел). Щелкнем по нему мышью, чтобы выделить, и посмотрим на редактор свойств (рис. 9.6).

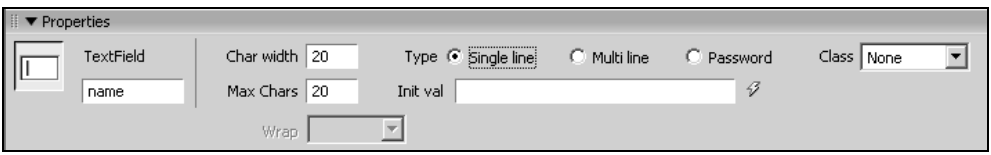


Рис. 9.6. Вид редактора свойств при выделенном поле ввода

В редакторе свойств мы введем следующие параметры только что созданного поля ввода:

- имя (поле ввода **TextField**) — `name`;
- ширина в символах (поле ввода **Char width**) — 20 (поле `name` таблицы `categories` имеет размер 20 символов);
- максимальное количество символов, которое можно ввести в это поле ввода (поле ввода **Max Chars**), — 20;
- тип поля ввода (набор переключателей **Type**) — обычное поле ввода в одну строку (переключатель **Single line**);
- значение по умолчанию (поле ввода **Init val**) — ничего.

Замечание

Кроме однострочных полей ввода, средствами HTML можно создавать области редактирования (переключатель **Multi line** группы **Type** в редакторе свойств) и особые поля ввода пароля (переключатель **Password**), которые мы рассмотрим в *главе 11*.

Введя эти данные, поставим текстовый курсор сразу после только что созданного поля ввода и нажмем клавишу <Enter>, чтобы создать другой абзац. В этом абзаце мы наберем текст `Файл:` и поставим пробел. А теперь немного подумаем.

Поле `file` таблицы `categories` имеет логический тип. Это значит, что оно может принимать значения `true` или `false`, "включено" или "выключено". Какой элемент нам для этого предпочесть?

Можно, конечно, создать обычный или раскрывающийся список с пунктами "Да" и "Нет" либо набор из двух таких же переключателей. Но список и набор переключателей займет много места на Web-странице, а раскрывающийся список не очень удобен — чтобы просмотреть все его пункты, его придется раскрыть. Так что нашим выбором будет флажок — с его помощью можно будет задать логическое значение более наглядно.

Флажок создается выбором пункта **Checkbox** все того же подменю **Form** меню **Insert**. Когда он появится после надписи `Файл:` (+пробел), щелкнем по нему мышью, чтобы выделить, и обратимся к редактору свойств (рис. 9.7).

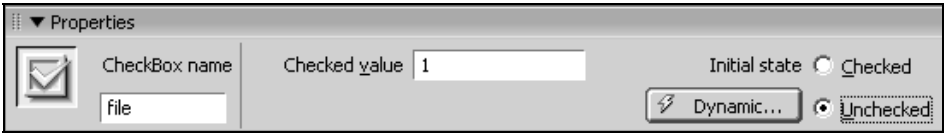


Рис. 9.7. Вид редактора свойств при выделенном флажке

В редакторе свойств нам будет нужно ввести вот такие параметры нашего флажка:

- имя (поле ввода **CheckBox name**) — `file`;
- значение, соответствующее включенному состоянию (поле ввода **Checked value**) — 1 (в принципе, можно ввести любое значение);
- начальное состояние (набор переключателей **Initial state**) — отключенное (переключатель **Unchecked**; переключатель **Checked** задает включенное состояние).

Если наш флажок будет включен, форма создаст вот такую пару:

```
file=1
```

Если же он не будет включен, то форма вообще его проигнорирует (как будто его нет) и не создаст на его основе никаких данных.

Закончив ввод, поставим текстовый курсор сразу после нашего флажка и нажмем клавишу <Enter>, чтобы создать еще один, третий абзац. В этом абзаце мы создадим последний элемент управления, без которого не обходится ни одна форма, — кнопку **Отправить**.

Чтобы создать кнопку, выберем пункт **Button** подменю **Form** меню **Insert**. Когда она появится на форме, выделим ее щелчком мыши и прибегнем к помощи редактора свойств (рис. 9.8), чтобы ввести ее параметры.

Рис. 9.8. Вид редактора свойств при выделенной кнопке

А параметры нашей кнопки будут такими:

- имя (поле ввода **Button name**) — `submit`;
- надпись на кнопке (поле ввода **Label**) — `Добавить`;
- действие, выполняемое кнопкой (набор переключателей **Action**), — отправка данных (переключатель **Submit form**).

Замечание

Чтобы создать кнопку сброса формы (Сброс), нужно включить переключатель **Reset form** группы **Action** (рис. 9.8). Также имеется возможность создать кнопку, не выполняющую никакого предусмотренного в HTML действия; для этого нужно включить переключатель **None**.

Все, наша форма готова — см. рис. 9.9. Теперь можно сохранить страницу и приступить к созданию серверного поведения, выполняющего добавление записи.

Для создания серверного поведения нам понадобится незаменимая панель **Server Behaviors**. Выведем ее на экран, нажмем кнопку со знаком "плюс" и выберем в появившемся на экране меню пункт **Insert Record**. На экране появится диалоговое окно **Insert Record** (рис. 9.10).

В раскрывающемся списке **Submit values from** выбирается имя формы, из которой будет производиться выборка данных для новой записи. В нашем случае это форма `category_add`.

Раскрывающийся список **Connection** уже знаком нам по другим окнам. Он задает соединение с базой данных; в нашем случае — `Site`.

Таблица, в которую должна быть добавлена запись, задается с помощью раскрывающегося списка **Insert table**. Выберем в нем таблицу `categories`.

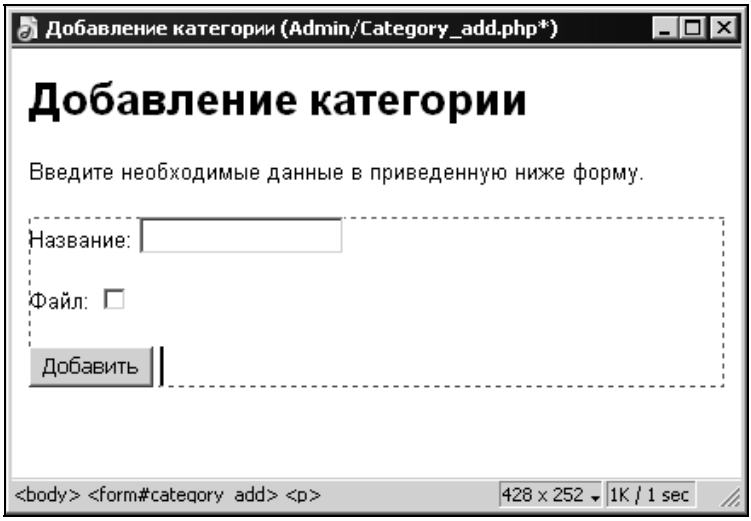


Рис. 9.9. Страница Category_add.php с готовой формой

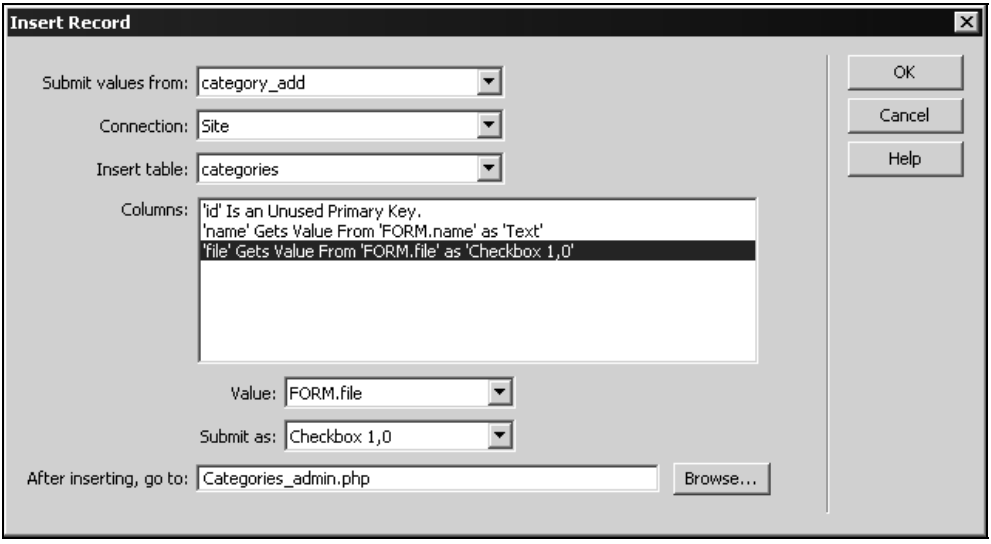


Рис. 9.10. Диалоговое окно **Insert Record**

Большой список **Columns** перечисляет все поля выбранной таблицы и соответствующие им элементы управления формы. Мы можем выбрать в нем любое поле и задать для него элемент управления и тип данных с помощью раскрывающихся списков **Value** и **Submit as** соответственно.

Давайте поочередно выбирать в списке **Columns** поля таблицы `categories` и задавать для них параметры. Итак, значение поля `id` не будет браться из

формы (оно проставляется самим сервером данных, поскольку имеет тип счетчика), поэтому мы выбираем пункт **None** в раскрывающемся списке **Value**. Значение поля `name` будет взято из поля ввода `name` формы (выберем пункт **FORM.name** в списке **Value**), а его тип текстовый (пункт **Text** списка **Submit as**). Поле `file` получит значение от флажка `file` (пункт **FORM.file** списка **Value**) логического типа.

Стоп-стоп! В списке **Submit as** присутствуют целых три флажка, задающие логический тип данных: **Checkbox Y,N**, **Checkbox 1,0** и **Checkbox -1,0**. Какой выбрать? Давайте подумаем. MySQL хранит логические данные в числовых полях "длиной" в одну цифру, а PHP все ненулевые числа считает как `true`, а ноль — как `false`. Значит, выбираем пункт **Checkbox 1,0** списка **Submit as**.

Что касается поля ввода **After inserting, go to**, то оно задает Web-страницу, на которую будет выполнен переход, если новая запись будет создана. Введем в это поле имя нашей административной страницы списка категорий — Categories_admin.php. Также можно щелкнуть по кнопке **Browse**, расположенной справа, и выбрать нужный файл в диалоговом окне открытия файла Dreamweaver.

Введя все нужные данные, можно нажать кнопку **OK**. Готовая форма с привязанным к ней серверным поведением **Insert record** показана на рис. 9.11.

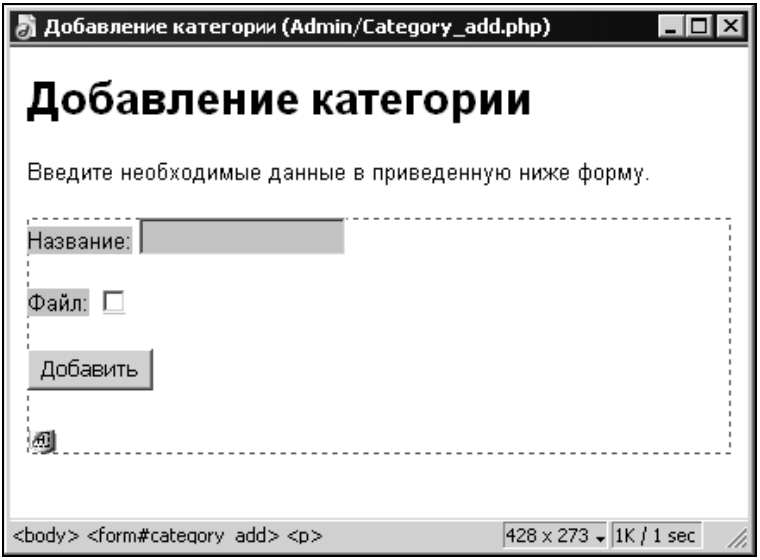


Рис. 9.11. Страница Category_add.php с готовой формой, к которой было привязано серверное поведение **Insert record**

Внизу формы хорошо заметен странный значок . Это так называемое *скрытое поле*, используемое для хранения в форме данных, которые не нужно

показывать посетителю. Dreamweaver создал это поле сам для своих собственных нужд.

Осталось только открыть страницу Categories_admin.php и превратить текст `Добавить`, находящийся в таблице, в гиперссылку, указывающую на страницу Category_add.php. И после этого можно проверить созданные нами страницы в действии.

Поскольку мы еще не создали на странице default.htm ссылки на административные страницы, то, чтобы попасть на список категорий, нам будет нужно набрать в строке адреса Web-обозревателя **http://localhost:8080/Admin/Categories_admin.php?file=<*0 для статей или 1 для файлов*>**. Можно попробовать добавить пару новых категорий — все равно мы их потом удалим, как только создадим страницу Category_delete.php.

Наша страница для добавления записи имеет только два недостатка. Во-первых, при вводе записи возврат всегда происходит на список категорий статей. Во-вторых, нет гиперссылки для возврата на список категорий, а это плохой тон в Web-дизайне. Но мы исправим оба этих недостатка позже.

Разбор сценариев PHP, используемых для добавления записи

Что ж, настала пора посмотреть на созданный Dreamweaver код PHP. Он довольно объемный, так что рассматривать мы его будем по частям. Переключимся в режим отображения кода HTML и начнем.

Сначала давайте найдем вот такой фрагмент кода PHP в большом сценарии, находящемся перед кодом HTML и соответствующем серверному поведению **Insert record**:

```
$editFormAction = $_SERVER['PHP_SELF'];
if (isset($_SERVER['QUERY_STRING'])) {
  $editFormAction .= "?" . htmlentities($_SERVER['QUERY_STRING']);
}
```

Сначала переменной `$editFormAction` присваивается имя файла серверной Web-страницы, которая обрабатывается в данный момент, то есть страницы Category_add.php. Это имя извлекается из элемента с индексом `PHP_SELF` встроенного массива `$_SERVER`.

Далее проверяется, были ли переданы этой странице какие-то аргументы методом `GET` (они находятся в элементе с индексом `QUERY_STRING` все того же встроенного массива `$_SERVER`). Если они были переданы (данный элемент массива существует), они добавляются к имени страницы, хранящейся в переменной `$editFormAction`.

Что касается функции `htmlentities`, которая выполняет кодирование передаваемых методом `GET` данных (в частности, преобразование символов), то,

поскольку элемент с индексом PHP_SELF массива $_SERVER содержит еще не раскодированные данные, вызывать ее фактически не нужно. Почему она все же присутствует в коде PHP — непонятно; вероятно, это какая-то ошибка в Dreamweaver.

Пропустим остальной код PHP и HTML и обратимся к HTML-коду, создающему форму. Вот он:

```
<FORM ACTION="<?php echo $editFormAction; ?>" METHOD="POST"
↳NAME="category_add" ID="category_add">
  <P>Название:
    <INPUT NAME="name" TYPE="text" ID="name" SIZE="20" MAXLENGTH="20">
  </P>
  <P>Файл:
    <INPUT NAME="file" TYPE="checkbox" ID="file" VALUE="1">
  </P>
  <P>
    <INPUT NAME="submit" TYPE="submit" ID="submit" VALUE="Добавить">
  </P>
  <INPUT TYPE="hidden" NAME="MM_insert" VALUE="category_add">
</FORM>
```

Здесь нам, в принципе, все уже знакомо. Видно, что Dreamweaver сам задал метод передачи данных (POST) и серверную страницу, которая примет эти данные (значение переменной $editFormAction). Также мы видим код, создающий скрытое поле:

```
<INPUT TYPE="hidden" NAME="MM_insert" VALUE="category_add">
```

Атрибут TYPE тега <INPUT> в случае скрытого поля должен содержать значение hidden. Также мы видим, что имя (значение атрибута NAME) установлено в MM_insert, а содержащиеся в скрытом поле данные (значение атрибута VALUE) — в category_add. Это все создал Dreamweaver для своих нужд, а для каких — сейчас станет известно.

Вернемся к тегу <FORM>, задающему нашу форму, и посмотрим на значение атрибута ACTION. Если переменная содержит имя данной Web-страницы, то, выходит, данные будут отправлены этой же самой странице — Category_add.php. И выходит, что она должна их обработать.

Это подход Dreamweaver — данные обрабатывает та же самая Web-страница, на которой они вводятся. Серверное поведение **Insert record** содержит сценарий, который проверяет наличие в данных, переданных ей методом POST, аргумента MM_insert со значением category_add — содержимого того самого скрытого поля! Если аргумент MM_insert отсутствует, то выводится форма для ввода данных, а если присутствует, значит, данные уже были введены и их нужно обработать.

То есть страница Category_add.php, созданная Dreamweaver, работает так:

1. При первом открытии, поскольку странице вообще не были переданы никакие данные методом `POST`, будет выведена форма.
2. После ввода данных в форму посетитель нажимает кнопку **Добавить**, и данные передаются методом `POST` этой же странице.
3. Страница открывается во второй раз. Сценарий PHP снова проверяет, были ли ей переданы данные методом `POST`, и, поскольку они были переданы, обрабатывает их. Форма в этом случае не выводится, а выполняется переход на страницу списка категорий Categories_admin.php.

А теперь давайте подробно, строка за строкой, разберем сценарий обработки данных.

```
if ((isset($_POST["MM_insert"])) &&
↳($_POST["MM_insert"] == "category_add")) {
```

Здесь проверяется, был ли передан данной странице методом `POST` аргумент `MM_insert` со значением `category_add` — содержимое скрытого поля, созданного Dreamweaver в нашей форме. Если он был передан, выполняется весь последующий блок.

```
  $insertSQL = sprintf("INSERT INTO categories (name, file) VALUES (%s,
  ↳%s)", GetSQLValueString($_POST['name'], "text"),
  ↳GetSQLValueString(isset($_POST['file']) ? "true" : "",
  ↳"defined","1","0"));
```

Переменная `$insertSQL` получает значение — код запроса SQL добавления записи. Для его создания опять используется встроенная функция `sprintf`, выполняющая замену присутствующих в строке шаблонов реальными значениями.

Также в этом выражении используется не знакомая нам функция `GetSQLValueString`. Это не встроенная функция — ее объявил сам Dreamweaver в самом начале сценария, соответствующего серверному поведению **Insert record**. Она занимается тем, что преобразует переданное ей первым аргументом значение так, чтобы оно было воспринято MySQL, и возвращает его как результат. При этом она руководствуется значением второго аргумента, которое задает тип данных преобразуемого значения. Так, если вторым аргументом передана строка `text`, то первый аргумент должен быть обработан как значение строкового типа (взят в одинарные кавычки).

Если же вторым аргументом функции `GetSQLValueString` передана строка `defined`, эта функция поступит следующим образом. Если первый аргумент содержит непустую строку, она вернет значение, переданное необязательным третьим аргументом. Если же передать этой функции первым аргументом пустую строку, она вернет необязательный четвертый аргумент.

Давайте посмотрим вот на такое выражение:

```
GetSQLValueString(isset($_POST['file']) ? "true" : "",
↳"defined","1","0"));
```

Здесь первым аргументом функции `GetSQLValueString` стоит выражение, проверяющее, был ли передан форме аргумент `file` (то есть был ли наш флажок включен). Если он был передан, то первым аргументом функции `GetSQLValueString` будет строка `true` и функция эта вернет `1`. Если же аргумент `file` не был передан форме, то первым аргументом функции `GetSQLValueString` станет пустая строка (`""`) и функция вернет `0`. А уже возвращенное этой функцией значение будет подставлено в запрос SQK добавления записи.

```
mysql_select_db($database_Site, $Site);
$Result1 = mysql_query($insertSQL, $Site) or die(mysql_error());
```

Эти два уже знакомых нам выражения выбирают базу данных `site` и выполняют созданный ранее и помещенный в переменную `$insertSQL` запрос SQL. Результат выполнения запроса помещается в переменную `$Result1` и больше нигде не используется. (В таком случае непонятно, зачем его нужно где-то хранить.)

```
$insertGoTo = "Categories_admin.php";
```

А это выражение помещает в переменную `$insertGoTo` имя Web-страницы, на которую будет выполнен переход после добавления записи. Это административная страница списка категорий.

```
if (isset($_SERVER['QUERY_STRING'])) {
```

А это условное выражение Dreamweaver создал, что называется, на всякий случай. Оно проверяет, были ли переданы странице Category_add.php какие-то данные методом `GET`. Если же они были переданы, то выполняются два выражения, приведенные далее.

```
  $insertGoTo .= (strpos($insertGoTo, '?')) ? "&" : "?";
```

Это выражение проверяет, встретился ли в значении переменной `$insertGoTo` символ вопросительного знака (`?`). Если он встретился, то к значению этой переменной добавляется `&`, в противном случае — `?`. Это нужно для того, чтобы передать странице, чье имя содержится в переменной `$insertGoTo`, все данные, переданные странице Category_add.php методом `GET`.

(Встроенная функция `strpos` возвращает номер позиции символа, переданного вторым аргументом, в строке, переданной первым аргументом. Если же этот символ в строке не встретился, возвращается 0.)

```
  $insertGoTo .= $_SERVER['QUERY_STRING'];
}
```

А это выражение, собственно, добавляет к значению переменной `$insertGoTo` закодированные данные, переданные странице Category_add.php методом `GET`. То есть страница Category_add.php переправляет переданные ей данные странице Category_admin.php.

```
  header(sprintf("Location: %s", $insertGoTo));
}
```

Последнее выражение, собственно, выполняет переход на страницу, чье имя (со всеми передаваемыми методом `GET` аргументами) находится в переменной `$insertGoTo`. Для этого используется синтаксис вида:

```
header("Location: <имя файла Web-страницы>");
```

Встретив функцию `header`, обработчик PHP через Web-сервер посылает Web-обозревателю особый серверный ответ, содержащий команду перехода на другую страницу. Когда Web-обозреватель ее получит, он отправит Web-серверу запрос на Web-страницу, чье имя было указано в команде (в нашем случае — это страница Categories_admin.php).

Задание значений по умолчанию для элементов управления

Да, наша страница Category_add.php работает, но не совсем так, как надо. После добавления записи она всегда выполняет возврат на список категорий статей, а вдобавок не содержит гиперссылки для возврата на список категорий. К тому же, было бы неплохо заставить флажок `file` изначально включаться или отключаться, в зависимости от того, со списка категорий статей или файлов мы на эту страницу перешли.

Выход из этого положения очевиден — нужно передать странице Category_add.php методом `GET` аргумент `file`. Откроем страницу Categories_admin.php, переключимся в режим отображения кода HTML и найдем код, создающий гиперссылку `Добавить`. Дадим этой гиперссылке такой интернет-адрес (значение атрибута `HREF` тега `<A>`):

```
Category_add.php?file=<?php echo $colname_Categories; ?>
```

(Переменная `$colname_Categories` объявляется в сценарии, соответствующем серверному поведению **Recordset**, и содержит значение аргумента `file`, переданного странице Categories_admin.php методом `GET`.)

Сохраним исправленную страницу Categories_admin.php и закроем. Переключимся на страницу Category_add.php.

Сначала нам нужно будет добавить код PHP, который извлечет значение из аргумента `file` и поместит в какую-нибудь переменную. Можно написать его вручную, но, поскольку мы работаем в Dreamweaver, пусть он сделает это за нас.

Нажмем комбинацию клавиш <Ctrl>+<F10>, чтобы вывести на экран панель **Bindings**. В этой панели нажмем кнопку со знаком "плюс" и выберем в появившемся на экране меню пункт **URL Variable**. На экране появится небольшое диалоговое окно **URL Variable**, показанное на рис. 9.12.

Рис. 9.12. Диалоговое окно **URL Variable**

В единственное поле ввода **Name** этого окна мы введем имя нашего аргумента — `file`. И нажмем кнопку **OK**. После этого в списке панели **Bindings** появится новое "дерево" с "корнем" **URL** и единственной "ветвью" **file**.

Мы только что выполнили *регистрацию* нашего аргумента `file` в Dreamweaver. Фактически мы дали понять Dreamweaver, что будем использовать этот аргумент в своих сценариях.

Зарегистрировав аргумент, выделим щелчком мыши флажок `file`. В редакторе свойств (см. рис. 9.7) нажмем кнопку **Dynamic**. На экране появится диалоговое окно **Dynamic CheckBox** (рис. 9.13).

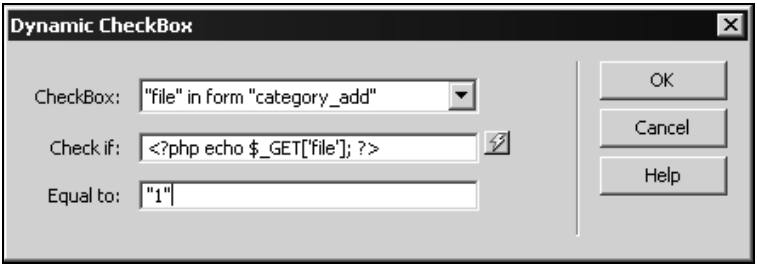


Рис. 9.13. Диалоговое окно **Dynamic CheckBox**

В раскрывающемся списке **CheckBox** выбирается флажок, которому мы хотим дать значение по умолчанию. Изначально там выбран наш флажок `file`, так что нам не придется делать это самим.

В поле ввода **Check if** задается параметр, на основе которого будет задаваться значение флажка по умолчанию. В нашем случае — это аргумент `file`, переданный методом `GET`. Мы можем ввести прямо в это поле ввода вот такой фрагмент кода PHP:

```
<?php $_GET['file'] ?>
```

А еще можно щелкнуть по кнопке со знаком молнии, расположенной справа от поля ввода **Check if**. После этого на экране появится диалоговое окно **Dynamic Data** (рис. 9.14).

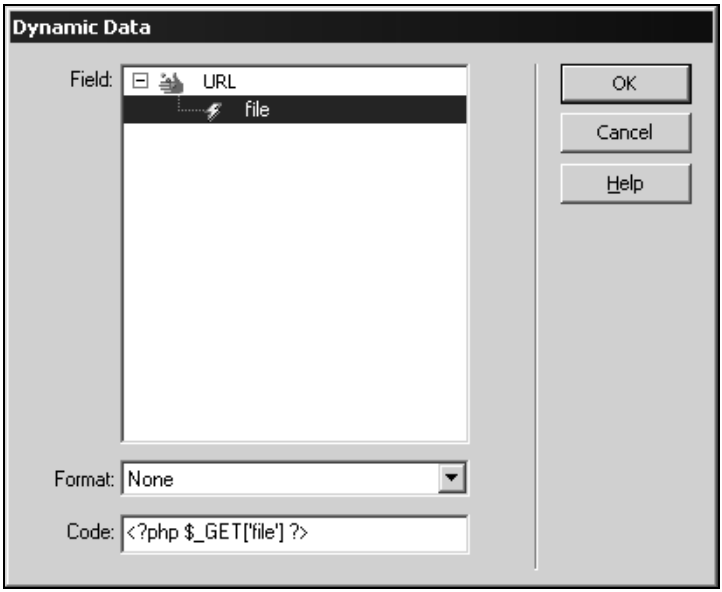


Рис. 9.14. Диалоговое окно **Dynamic Data**

В иерархическом списке **Field** этого окна выбирается сам параметр, на основе которого будет задаваться значение по умолчанию. Нам будет нужно развернуть в этом списке "дерево" с "корнем" **URL** и выбрать "ветвь" **file**, соответствующей нашему аргументу `file`. В поле ввода **Code** появится фрагмент кода PHP, извлекающий значение этого аргумента.

Что касается раскрывающегося списка **Format**, то он задает дополнительное форматирование значения выбранного в списке **Field** параметра. Обычно никакого дополнительного форматирования не нужно, поэтому мы можем со спокойной совестью выбрать в этом списке пункт **None**.

Выбрав нужный параметр, нажмем кнопку **OK** диалогового окна **Dynamic Data** и снова попадем в окно **Dynamic CheckBox**.

Последнее, что нам остается, — это занести в поле ввода **Equal to** значение, с которым будет сравниваться значение параметра, заданного в поле ввода **Check if**. Если они равны, флажок будет включен, в противном случае — выключен. Нам нужно ввести туда `1` — это числовой эквивалент значения `true`, представленный в виде строки, так как значения элементов массива `$_GET` представлены именно в строковом виде.

Вот теперь можно нажать кнопку **OK**, чтобы закрыть окно **Dynamic CheckBox** и заставить Dreamweaver создать нужный сценарий. Все, значение по умолчанию для флажка `file` задано.

Сразу же давайте создадим гиперссылку, указывающую на страницу списка категорий (административную, разумеется). Поставим текстовый курсор правее формы и нажмем клавишу <Enter>, чтобы создать пустой абзац. В этом пустом абзаце наберем текст `На список категорий` и превратим его в гиперссылку с таким вот интернет-адресом:

```
Categories_admin.php?file=<?php echo $_GET["file"]; ?>
```

Теперь можно сохранить готовую страницу и проверить ее в действии. А, вдоволь налюбовавшись на нее, давайте переключимся в режим отображения кода HTML и посмотрим, какой код PHP создал Dreamweaver, чтобы дать флажку `file` значение по умолчанию. Код на удивление невелик (он выделен полужирным шрифтом):

```
<INPUT <?php if (!(strcmp($_GET['file'],"1"))) {echo "checked";} ?>
↳NAME="file" TYPE="checkbox" ID="file" VALUE="1">
```

Встроенная функция `strcmp` выполняет весьма сложное сравнение строк, переданных в качестве ее аргументов. Полное описание этой функции приведено в справочном руководстве по PHP; нам же нужно знать, что она возвращает 0, если строки равны, и любое ненулевое значение, если они не равны. А, применив оператор логического НЕ `!`, мы получим значение `true` в случае равенства строк и значение `false` в случае их неравенства.

Если строки равны (условие в условном выражении вернуло результат `true`), то в тег `<INPUT>`, создающий флажок, подставляется атрибут `CHECKED`, который делает флажок включенным. Это так называемый *атрибут без значения*; он не имеет значения, достаточно только его присутствия в теге.

Все, больше никакого кода PHP Dreamweaver в страницу не добавил. (Созданный нами вручную код в интернет-адресе гиперссылки не в счет.) Так что закончим со страницей Category_add.php и перейдем к созданию страницы Category_edit.php, выполняющей правку данных выбранной категории.

Простая страница для правки записи

Создадим новую серверную страницу PHP и сохраним под именем Category_edit.php в папке Admin. Дадим ей название `Изменение категории`, введем такой же заголовок и какой-либо поясняющий текст. И создадим точно такую же форму, как и на странице Category_add.php, только назовем ее `category_edit`, а надпись на кнопке `submit` сделаем другую — `Изменить`.

А теперь остановимся и подумаем. Для правки записи нам будет нужно выполнить две задачи:

- извлечь из таблицы `categories` нужную запись и заполнить значениями ее полей элементы управления формы `category_edit`;

- после нажатия кнопки **Изменить** — извлечь исправленные значения из элементов управления формы `category_edit` и записать их в соответствующие поля этой же записи таблицы `categories`.

Но дело в том, что предусмотренное в Dreamweaver серверное поведение, выполняющее изменение записи (забегая вперед, скажем, что оно называется **Update Record**), выполняет только вторую часть этой работы — переносит исправленные значения в таблицу. Заполнить элементы управления формы старыми значениями выбранной для правки записи нам придется самим. А для этого необходимо создать набор записей, содержащий выбранную для правки запись.

Найти нужную запись не проблема. Достаточно передать странице методом `GET` аргумент `id`, содержащий значение поля `id` таблицы `categories`. Это поле содержит уникальные в пределах всей таблицы значения, так что нужную нам запись мы найдем без труда.

Кроме того, нам придется где-то сохранить значение поля `id` извлеченной записи. Серверное поведение **Update Record** потом использует его, чтобы найти запись, которую нужно изменить. Поэтому давайте вставим в нашу форму `category_edit` скрытое поле.

Поставим текстовый курсор сразу после флажка `file` — именно здесь будет находиться скрытое поле, хранящее значение поля `id` выбранной записи. Чтобы создать это скрытое поле, выберем пункт **Hidden Field** подменю **Form** меню **Insert**. Когда скрытое поле будет создано, выделим его щелчком мыши и посмотрим на редактор свойств (рис. 9.15).

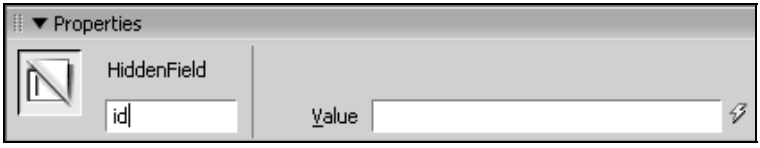


Рис. 9.15. Вид редактора свойств при выделенном скрытом поле

Единственный параметр, который нам здесь нужно задать, — это имя скрытого поля. Оно заносится в поле ввода **HiddenField**.

А вот теперь можно приступить к созданию набора записей. Нажмем комбинацию клавиш <Ctrl>+<F10>, чтобы вывести на экран панель **Bindings**, щелкнем по кнопке со знаком "плюс" и выберем в появившемся меню пункт **Recordset (Query)**. В диалоговом окне **Recordset** зададим такие параметры:

- имя набора данных (поле ввода **Name**) — `Category`;
- имя соединения (раскрывающийся список **Connection**) — `Site`;
- имя таблицы (раскрывающийся список **Table**) — `categories`;

- все поля таблицы (переключатель **All** в группе **Columns**);
- фильтрация по полю `id` (раскрывающийся список **Filter**);
- значение поля `id` должно быть равно (пункт = раскрывающегося списка, который находится правее списка **Filter**);
- значению аргумента, переданного методом `GET` (пункт **URL Parameter** раскрывающегося списка, что ниже списка **Filter**);
- имя которого — `id` (поле ввода правее и ниже списка **Filter**);
- без сортировки (пункт **None** раскрывающегося списка **Sort**).

Чтобы создать набор записей, нажмем кнопку **OK**. В иерархическом списке панели **Bindings** появится соответствующее только что созданному нами набору "дерево".

Следующая наша задача — установить для элементов управления формы `category_edit` значения по умолчанию. Эти значения будут браться из соответствующих полей только что созданного нами набора записей `Category`.

Щелкнем по полю ввода `name`, чтобы его выделить, и нажмем в редакторе свойств маленькую кнопку, имеющую знак молнии и расположенную правее поля ввода **Init val** (см. рис. 9.6). На экране появится уже знакомое нам диалоговое окно **Dynamic Data** (см. рис. 9.14).

В иерархическом списке **Field** этого окна развернем "дерево" **Recordset (Category)** и выберем "ветвь" **name**, соответствующую полю `name` набора записей `Category`. В поле ввода **Code** появится код PHP, извлекающий значение этого поля. После этого нажмем кнопку **OK**.

Далее выделим флажок `file` и нажмем кнопку **Dynamic** редактора свойств. В окне **Dynamic CheckBox** нажмем кнопку со знаком молнии, находящуюся правее поля ввода **Check if**. В иерархическом списке **Field** появившегося после этого на экране диалогового окна **Dynamic Data** выберем "ветвь" **file** "дерева" **Recordset (Category)** и нажмем кнопку **OK**. Снова оказавшись в окне **Dynamic CheckBox**, введем в поде ввода **Equal to** значение `1` и опять нажмем кнопку **OK**.

Напоследок выделим скрытое поле `id` и нажмем все ту же кнопку со знаком молнии, что находится в редакторе свойств правее поля ввода **Value** (см. рис. 9.15). В иерархическом списке **Field** окна **Recordset (Category)** развернем "дерево" **Recordset (Category)** и выберем "ветвь" **id**, после чего нажмем кнопку **OK**.

Вот теперь можно приступать к созданию серверного поведения **Update Record**. Выведем на экран панель **Server Behaviors**, нажмем на ней кнопку со знаком "плюс" и выберем в появившемся на экране меню пункт **Update Record**. На экране появится диалоговое окно **Update Record** (рис. 9.16).

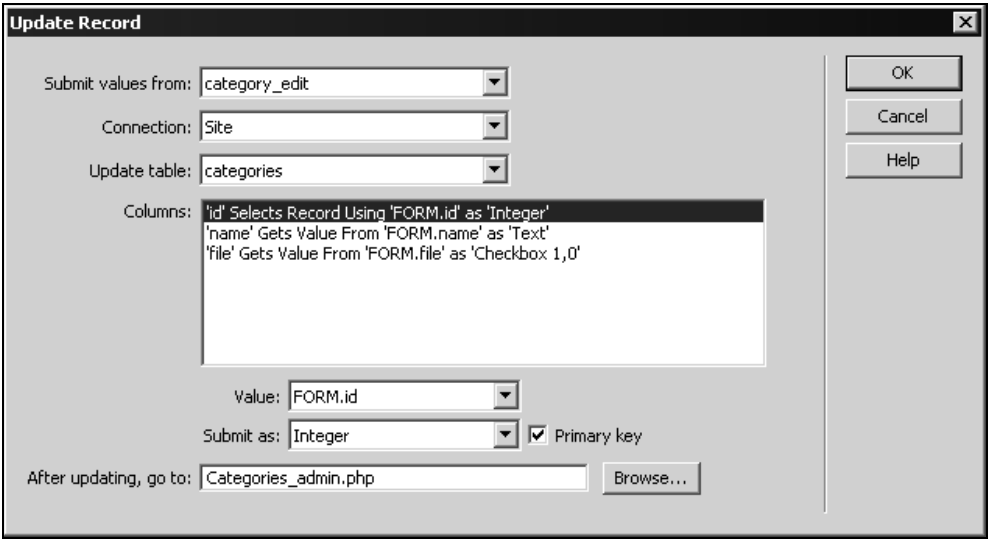


Рис. 9.16. Диалоговое окно **Update Record**

В раскрывающемся списке **Submit values from** выбирается имя формы, из которой будут выбираться исправленные данные для занесения в запись. В нашем случае это форма `category_edit`.

Раскрывающийся список **Connection**, как обычно, задает соединение с базой данных; в нашем случае — `Site`.

Таблица, в которой находится изменяемая запись, задается с помощью раскрывающегося списка **Update table**. Выберем в нем таблицу `categories`.

Большой список **Columns** перечисляет все поля выбранной таблицы и соответствующие им элементы управления формы. Мы можем выбрать в нем любое поле и задать для него элемент управления и тип данных с помощью раскрывающихся списков **Value** и **Submit as** соответственно.

Давайте поочередно выбирать в списке **Columns** поля таблицы `categories` и задавать для них параметры. Значение поля `name` будет взято из поля ввода `name` формы (выберем пункт **FORM.name** в списке **Value**), а его тип текстовый (пункт **Text** списка **Submit as**). Поле `file` получит значение от флажка `file`, тип же его будет логическим в числовой форме (пункт **Checkbox 1,0** списка **Submit as**).

Что касается поля `id`, то мы выберем для него в списке **Value** пункт **FORM.id**, в списке **Submit as** — пункт **Integer** (целочисленный тип данных). Кроме того, сценарий, который создаст Dreamweaver, должен выполнить поиск исправляемой записи по значению именно этого поля. Так вот, чтобы он это сделал, нам нужно обязательно включить флажок **Primary key**.

Поле ввода **After updating, go to** задает Web-страницу, на которую будет выполнен переход после изменения записи. Введем в это поле имя нашей административной страницы списка категорий — Categories_admin.php. Также можно щелкнуть по кнопке **Browse**, расположенной справа, и выбрать нужный файл в диалоговом окне открытия файла Dreamweaver.

Введя все нужные данные, можно нажать кнопку **OK**. После этого Dreamweaver создаст нужное нам серверное поведение **Update Record**.

Сохраним почти готовую страницу Category_edit.php и откроем административную страницу списка категорий Categories_admin.php. Найдем в таблице, содержащей список категорий, строку `Изменить` и превратим ее в гиперссылку с интернет-адресом

```
Category_edit.php?id=<?php echo $row_Categories['id']; ?>
```

(Массив `$row_Categories` содержит значения полей набора записей `Categories` страницы Categories_admin.php.)

Все? Отнюдь. Когда мы начнем эксплуатировать страницу Category_edit.php, то сразу же столкнемся с тем, что после изменения записи происходит возврат исключительно на список статей (значение аргумента `file`, передаваемого методом `GET` странице Categories_admin.php, равно 0, как мы сами задали в *главе 8*). Как это исправить?

Выход довольно прост. Когда мы создавали страницу Category_add.php, то столкнулись с тем, что эта страница при переходе на страницу "возвращала" ей все аргументы, переданные ей методом `GET`. Значит, если мы вместе с аргументом `id` передадим странице Category_edit.php аргумент `file`, то после изменения записи оба они "вернутся" странице Categories_admin.php. Незнакомый аргумент `id` эта страница проигнорирует, а `file` использует для вывода соответствующего списка категорий.

Сказано — сделано. Задаем гиперссылке `Изменить` новый интернет-адрес:

```
Category_edit.php?id=<?php echo $row_Categories['id']; ?>&
↳file=<?php echo $colname_Categories; ?>
```

(Переменная `$colname_Categories` содержит значение аргумента `file`, переданного странице Categories_admin.php методом `GET`.)

Осталось вернуться к странице Category_edit.php и создать в ней гиперссылку, указывающую на страницу списка категорий Categories_admin.php, чтобы мы смогли вернуться к ней, если почему-то передумаем править запись. Поставим текстовый курсор правее формы и нажмем клавишу <Enter>, чтобы создать пустой абзац. В этом пустом абзаце наберем текст `На список категорий` и превратим его в гиперссылку с таким вот интернет-адресом:

```
Categories_admin.php?file=<?php echo $_GET["file"]; ?>
```

Все, страница правки записи закончена! Можно проверить ее в действии.

Разбор сценариев PHP, используемых для правки записи

Собственно, разбирать тут особо нечего. Сценарий, соответствующий серверному поведению **Update Record**, практически ничем не отличается от сценария, соответствующего уже знакомому нам серверному поведению **Insert Record**. А мы его уже разобрали. Поэтому давайте рассмотрим только те фрагменты сценариев, которые от них отличаются.

Вот код HTML, создающий форму `category_edit`:

```
<FORM METHOD="POST" ACTION="<?php echo $editFormAction; ?>"
↳NAME="category_edit" ID="category_edit">
  <P>Название:
    <INPUT NAME="name" TYPE="text" ID="name" VALUE="<?php echo
    ↳$row_Category['name']; ?>" SIZE="20" MAXLENGTH="20">
  </P>
  <P>Файл:
    <INPUT <?php if (!(strcmp($row_Category['file'],"1"))) {echo
    ↳"checked";} ?> NAME="file" TYPE="checkbox" ID="file" VALUE="1">
    <INPUT NAME="id" TYPE="hidden" ID="id" VALUE="<?php echo
    ↳$row_Category['id']; ?>">
  </P>
  <P>
    <INPUT NAME="submit" TYPE="submit" ID="submit" VALUE="Изменить">
  </P>
  <INPUT TYPE="hidden" NAME="MM_update" VALUE="category_edit">
</FORM>
```

Здесь добавился еще один тег `<INPUT>` со значением атрибута `TYPE`, равным `hidden`, задающий скрытое поле `id`. Служебное скрытое поле, созданное самим Dreamweaver, уже имеет имя `MM_update` и несет в себе строку `category_edit` (значение атрибута `VALUE` тега `<INPUT>`). Соответственно, сценарий, изменяющий запись, проверяет на наличие в переданных этой странице с помощью метода `POST` данных элемента `MM_update` встроенного массива `$_POST`.

Также здесь добавились небольшие сценарии, помещающие в элементы управления значения соответствующих им полей таблицы `categories`. Они заносят эти значения во все те же атрибуты `VALUE` тегов `<INPUT>`. Мы не будем рассматривать эти сценарии — они очень просты и уже нам знакомы.

А вот фрагмент кода PHP, формирующий запрос SQL изменения записи:

```
$updateSQL = sprintf("UPDATE categories SET name=%s, file=%s WHERE
  ↳id=%s", GetSQLValueString($_POST['name'], "text"),
```

```
↳GetSQLValueString(isset($_POST['file']) ? "true" : "",
↳"defined","1","0"), GetSQLValueString($_POST['id'], "int"));
```

Видно, что значения поля ввода `name` и флажка `file` заносятся в соответствующие им поля, а значение скрытого поля `id` используется для поиска изменяемой записи.

Больше ничего интересного в созданных Dreamweaver сценариях PHP нет. Поэтому давайте оставим их в покое и приступим к созданию Web-страницы удаления записи Category_delete.php.

Простая страница для удаления записи

И в третий раз создадим новую серверную страницу PHP и сохраним под именем Category_delete.php в папке Admin. Дадим ей название `Удаление категории`, введем такой же заголовок и поясняющий текст. Напоследок создадим на этой странице пустую форму и назовем ее `category_delete`.

Для удаления записи нам, опять же, придется выполнить две задачи:

- извлечь из таблицы `categories` нужную запись и вывести значения ее полей на Web-страницу;
- после нажатия кнопки **Удалить** — удалить выбранную запись из таблицы `categories`.

Эти задачи заметно проще, чем в случае страницы Category_edit.php. Но повозиться тоже придется. И начнем мы с набора записей, содержащего одну-единственную — выбранную для удаления — запись. Он будет иметь такие же параметры, как и набор, что мы создали, когда работали над страницей Category_edit.php, и то же имя — `Category`.

Создав набор записей, поставим текстовый курсор в форму `category_delete`, наберем текст `Название:` и поставим пробел. Сразу же после пробела поставим динамический текст, отображающий значение поля `name` набора записей `Category`. Потом нажмем клавишу <Enter>, чтобы создать новый абзац, наберем в этом абзаце текст `Файл:` и поставим пробел. И сразу же после этого переключимся в режим отображения кода HTML, чтобы ввести вручную сценарий, выводящий значение поля `file`. Он будет совсем простым:

```
<?php if (!(strcmp($row_Category['file'],"1"))) { echo "+"; } ?>
```

Как видно, если поле `file` содержит значение `true`, будет выведен знак "плюс". Если же это поле содержит `false`, ничего выведено не будет. Просто и понятно.

Переключимся обратно в режим отображения Web-страницы и поставим курсор в конце второго абзаца. Поместим там скрытое поле, выбрав пункт **Hidden Field** подменю **Form** меню **Insert**. Дадим этому скрытому полю имя `id`.

Выделим скрытое поле `id` и нажмем кнопку со знаком молнии, находящуюся в редакторе свойств правее поля ввода **Value** (см. рис. 9.15). В иерархическом

списке **Field** окна **Recordset (Category)** развернем "дерево" **Recordset (Category)** и выберем "ветвь" **id**, после чего нажмем кнопку **OK**.

Далее поставим текстовый курсор после только что созданного скрытого поля и нажмем клавишу <Enter>, чтобы создать еще один абзац. В нем поместим кнопку с такими параметрами:

- имя (поле ввода **Button name** редактора свойств) — `submit`;
- надпись на кнопке (поле ввода **Label**) — `Удалить`;
- действие, выполняемое кнопкой (набор переключателей **Action**) — отправка данных (переключатель **Submit form**).

Сохраним страницу и приступим к созданию серверного поведения, реализующего удаление записи. Оно называется **Delete Record**.

Выведем на экран панель **Server Behaviors**, нажмем в ней кнопку со знаком "плюс" и выберем в появившемся на экране меню пункт **Delete Record**. На экране появится диалоговое окно **Delete Record** (рис. 9.17).

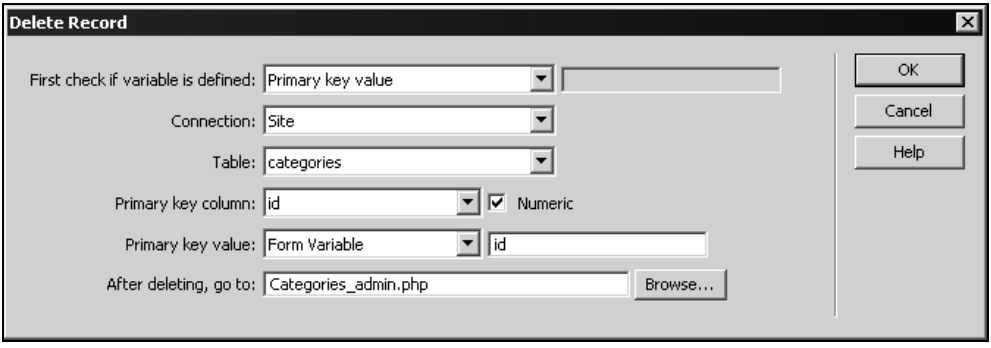


Рис. 9.17. Диалоговое окно **Delete Record**

Насколько удалось выяснить автору, в раскрывающемся списке **First check if variable is defined** этого окна можно выбрать любой пункт. По умолчанию там выбран пункт **Primary key value** — не будем его менять.

Раскрывающийся список **Connection**, опять же, задает соединение с базой данных; в нашем случае — `Site`.

Таблица, из которой нужно удалить запись, задается с помощью раскрывающегося списка **Table**. Выберем в нем нашу таблицу `categories`.

В раскрывающемся списке **Primary key column** выбирается поле этой таблицы, по которому будет производиться поиск удаляемой записи. В нашем случае — это поле `id`, поэтому выберем соответствующий пункт в этом списке. И включим флажок **Numeric**, поскольку это поле имеет числовой тип.

А раскрывающийся список **Primary key value** задает, откуда будет взято значение для поиска в поле, заданном в списке **Primary key column**. Поскольку

нам нужно будет выполнять поиск значения, содержащегося в скрытом поле `id`, выберем пункт **Form Parameter** — аргумент, переданный методом `POST`. И введем имя этого аргумента — `id` — в появившееся справа поле ввода.

Ну, а поле ввода **After deleting, go to** задает Web-страницу, на которую будет выполнен переход после изменения записи. Введем в него имя нашей административной страницы списка категорий — Categories_admin.php. Также можно щелкнуть по кнопке **Browse**, расположенной справа, и выбрать нужный файл в диалоговом окне открытия файла Dreamweaver.

Теперь нажмем кнопку **OK**, и серверное поведение **Delete Record** будет создано.

Сохраним страницу Category_delete.php и откроем административную страницу списка категорий Categories_admin.php. Найдем в таблице, содержащей список категорий, строку `Удалить` и превратим ее в гиперссылку с интернет-адресом

```
Category_delete.php?id=<?php echo $row_Categories['id']; ?>&
↳file=<?php echo $colname_Categories; ?>
```

Осталось вернуться к странице Category_delete.php и создать в ней гиперссылку, указывающую на страницу списка категорий Categories_admin.php. Поставим текстовый курсор правее формы и нажмем клавишу <Enter>, чтобы создать пустой абзац. В этом пустом абзаце наберем текст `На список категорий` и превратим его в гиперссылку с таким вот интернет-адресом:

```
Categories_admin.php?file=<?php echo $_GET["file"]; ?>
```

На этом все. Три страницы, реализующие добавление, правку и удаление записей, нами сделаны. Теперь очередь за аналогичными страницами, но предназначенными для добавления, изменения и удаления статей и файлов.

Более сложные серверные страницы для ввода и правки данных

Страницы, предназначенные для добавления, правки и удаления статей и файлов, будут несколько более сложными. И не только потому, что таблица `items` содержит больше полей, в которые придется вводить значения, но и потому, что страницы для работы с записями этой таблицы будут содержать списки.

Начнем мы со страницы Item_add.php, предназначенной для добавления статей и файлов. Создадим новую серверную страницу PHP и сохраним под именем Item_add.php в папке Admin. Дадим ей название `Добавление статьи (файла)`, введем такой же заголовок и какой-либо поясняющий текст. И создадим форму, имя которой будет `item_add`.

Таблица `items` нашей базы данных `site` имеет значительно больше полей, чем таблица `categories`. Поэтому над созданием элементов управления нам придется потрудиться. И не потому, что они какие-то там невероятно сложные — просто их много.

Итак, ставим в форму текстовый курсор, пишем текст `Автор:`, ставим пробел и создаем поле ввода с такими параметрами:

- имя (поле ввода **TextField**) — `author`;
- ширина в символах (поле ввода **Char width**) — 30 (поле `author` таблицы `items` имеет размер 30 символов);
- максимальное количество символов, которое можно ввести в это поле ввода (поле ввода **Max Chars**), — 30;
- тип поля ввода (набор переключателей **Type**) — обычное поле ввода в одну строку (переключатель **Single line**);
- значение по умолчанию (поле ввода **Init val**) — ничего.

Ставим текстовый курсор сразу же после только что созданного поля ввода и жмем клавишу <Enter>. В новом абзаце пишем `Название:`, ставим пробел и создаем второе поле ввода с параметрами:

- имя — `name`;
- ширина в символах — 60 (поле `name` таблицы `items` имеет размер 60 символов);
- максимальное количество введенных символов — 60;
- тип поля ввода — обычное поле ввода в одну строку;
- значение по умолчанию — ничего.

В следующем абзаце пишем `Добавлено:`, ставим пробел и создаем третье поле ввода. Его параметры таковы:

- имя — `added`;
- ширина в символах — 14 (пусть будет побольше);
- максимальное количество введенных символов — 10 (два символа числа, два символа номера месяца, четыре символа года и две точки);
- тип поля ввода — обычное поле ввода в одну строку.

А вот значение по умолчанию мы этому полю дадим — пусть в нем автоматически подставляется сегодняшняя дата. Нажмем кнопку со знаком молнии в редакторе свойств и введем в поле ввода **Code** диалогового окна **Dynamic Data** вот такой код PHP:

```
<?php echo date("Y-m-d"); ?>
```

Встроенная функция `date` нам знакома по *главе 8*. Она форматирует значение даты, переданное вторым аргументом, согласно строке с шаблонами,

переданной первым аргументом, и возвращает его в строковом виде. Если же второй аргумент не указан, эта функция вернет значение сегодняшней даты. Что нам и нужно.

Да, но почему мы выбрали какой-то странный формат представления даты — "год-месяц-число"? Дело в том, что это единственный формат даты, который "понимает" MySQL. Если же он получит дату в привычном нам формате "число.месяц.год", то запишет в поле `added` такое, что смотреть страшно будет. Увы — это ограничение самой программы MySQL, и нам придется это учитывать

Замечание

Вообще, эту проблему можно решить, и решение будет несложным. Каким? А его мы найдем самостоятельно. Подсказка №1: встроенная функция PHP по имени `substr` позволяет "вырезать" из строки фрагмент произвольной длины. Подсказка №2: оператор объединения строк `.`.

И еще один, четвертый, абзац. Вводим текст `Гиперссылка:`, пробел, вставляем новое поле ввода Его параметры:

- имя — `href`;
- ширина в символах — 60 (иначе оно не поместится на страницу);
- максимальное количество введенных символов — 255 (именно такой размер имеет поле `href` таблицы `items`);
- тип поля ввода — обычное поле ввода в одну строку;
- значение по умолчанию — ничего.

Терпение — осталось только два элемента управления! Создаем еще один абзац, пишем в нем `Категория:`, ставим пробел и создаем... А что создаем?

Здесь нам нужно предоставить выбор из нескольких позиций. Это можно сделать с помощью трех различных элементов управления: обычного списка, раскрывающегося списка и набора переключателей. Каждый из этих элементов управления имеет свои достоинства и недостатки, которые мы сейчас рассмотрим.

Обычный список позволяет отобразить одновременно достаточно много доступных для выбора позиций, но весьма громоздок. Он эффективен, если позиций, доступных для выбора, очень много — десятки, если не сотни. А это не наш случай.

Набор переключателей лучше всего использовать тогда, когда позиций не очень много — в пределах десяти, — и все они должны быть на виду. Если же позиций больше, то набор переключателей станет просто гигантским и уж точно не поместится ни на одной Web-странице.

Раскрывающийся список компактен и может содержать очень много позиций. Его, наверное, единственный недостаток — на виду только одна позиция,

выбранная в данный момент. Но этот недостаток не принципиален, так что давайте остановимся на раскрывающемся списке.

Чтобы создать раскрывающийся список, нужно выбрать пункт **List/Menu** подменю **Form** меню **Insert**. Когда список будет создан, щелкнем по нему и обратимся к редактору свойств (рис. 9.18), чтобы задать параметры этого списка.

Рис. 9.18. Вид редактора свойств при выделенном списке

Параметров мы пока зададим всего два:

- имя (поле ввода **List/Menu**) — `category`;
- тип (группа переключателей **Type**) — раскрывающийся список (переключатель **Menu**).

Замечание

Если включить переключатель **List** группы **Type**, будет создан обычный список.

Осталось поставить текстовый курсор сразу после только что созданного списка, нажать клавишу <Enter> и создать в новом абзаце кнопку. Ее параметры будут такими:

- имя — `submit`;
- надпись на кнопке — `Добавить`;
- действие, выполняемое кнопкой — отправка данных.

Сохраним страницу Item_add.php. И обратим внимание на список `category`.

Что он будет содержать? Список категорий. А откуда ему взять этот список? Из таблицы `categories`. Значит, нам нужно создать набор записей, содержащий категории статей или файлов, и заполнить значениями из этого набора созданный нами список.

Описывать в очередной раз создание набора записей не стоит — мы создали их уже предостаточно. Итак, наш набор записей будет содержать поля `id` и `name` из таблицы `categories` базы данных `site`. Фильтрация будет проводиться по значению поля `file` таблицы, а сравниваемое значение должно быть взято из аргумента `file`, переданного методом `GET` от страницы Items_admin.php. Сортироваться записи набора будут по полю `id`. А называться он будет `Categories`.

И еще. Давайте сделаем так, чтобы в списке `category` уже при открытии страницы Item_add.php была подставлена категория, статьи (файлы) которой были перечислены на странице Items_admin.php. Для этого нам придется передать от страницы методом `GET` еще один аргумент — `catid`, содержащий значение поля `id` таблицы `categories` для данной категории. Давайте сразу же выведем на экран панель **Bindings**, нажмем кнопку со знаком "плюс", выберем в появившемся на экране меню пункт **URL Variable**, введем в поле ввода **Name** диалогового окна **URL Variable** (см. рис. 9.12) имя создаваемого аргумента — `catid` — и нажмем кнопку **OK**.

Ну что, закончили? Выделяем в форме `item_add` список `category` и нажимаем в редакторе свойств кнопку **Dynamic** (см. рис. 9.18). На экране появится диалоговое окно **Dynamic List/Menu** (рис. 9.19).

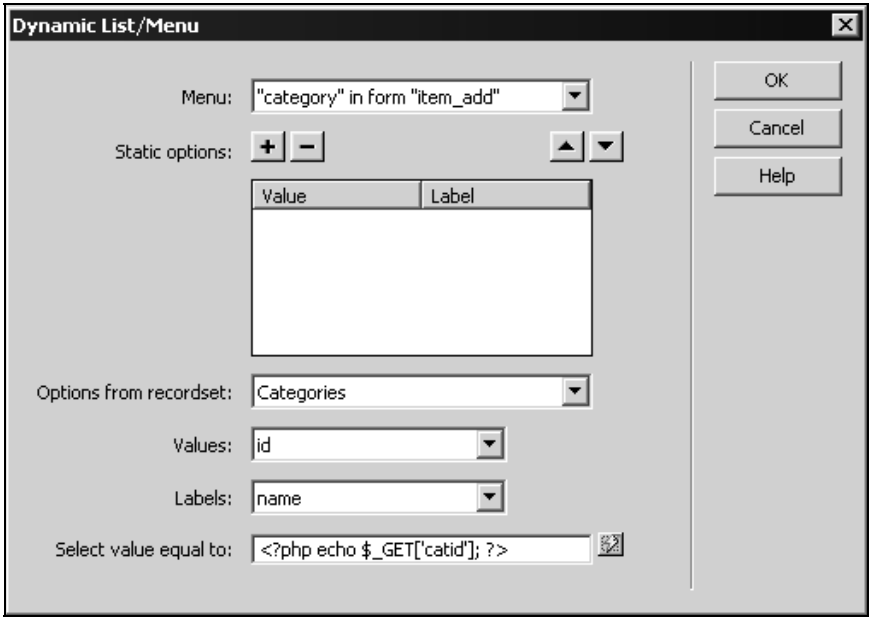


Рис. 9.19. Диалоговое окно **Dynamic List/Menu**

В раскрывающемся списке **Menu** этого окна задается список, который нам нужно заполнить пунктами. Там уже должен быть выбран наш единственный список `category`, так что не будем его трогать.

Список **Static options** с набором кнопок позволяет ввести в список некоторые пункты вручную. В данный момент это нам не нужно.

А вот и список **Options from recordset**. Он задает набор записей, из которого будут взяты записи, будущие пункты нашего списка. Выберем в нем пункт **Categories** — наш набор записей.

Раскрывающийся список **Values** задает поле набора записей, из которого будут взяты так называемые *значения пунктов списка*. Эти значения не отображаются на экране, но именно они отправляются серверной программе, которая должна получить данные из формы. Выберем в этом списке пункт **id** — это поле содержит уникальное значение, идентифицирующее категорию.

А раскрывающийся список **Labels** задает поле набора записей, из которого будут взяты названия пунктов списка, те самые, которые отображаются на экране. Разумеется, нам нужно выбрать пункт **name**.

В поле ввода **Select value equal to** заносится код PHP, возвращающий значение списка по умолчанию — значение того пункта списка, который должен быть выбран изначально. Туда можно ничего не вводить, а щелкнуть по кнопке со знаком молнии справа от этого поля, выбрать в иерархическом списке **Dynamic Data** нужный аргумент или поле и нажать кнопку **OK** этого окна. В нашем случае там нужно выбрать аргумент `catid`, передаваемый методом `GET`, — именно он содержит номер категории.

Задав все нужные данные, нажмем кнопку **OK**. А после этого можем приступать к созданию серверного поведения **Insert Record**, поскольку все подготовительные действия мы выполнили.

В диалоговом окне **Insert Record** (см. рис. 9.10) нам будет нужно выбрать:

- форма, из которой будут взяты данные для новой записи (раскрывающийся список **Submit values from**), — `item_add`;
- имя соединения с базой данных (раскрывающийся список **Connection**) — `Site`;
- таблица, в которую будет добавлена запись (раскрывающийся список **Insert table**), — `items`;
- страница, на которую будет выполнен переход после добавления записи (поле ввода **After inserting, go to**), — Items_admin.php.

Все? Отнюдь. Нам еще нужно поочередно выбрать в списке **Column** каждое поле таблицы `items`, которое там есть, и задать для него кое-какие параметры. Все они перечислены в табл. 9.1.

Таблица 9.1. *Параметры полей таблицы* `items` *для страницы Items_add.php*

Поле таблицы	Элемент управления формы (список Value)	Тип данных (список Submit as)
`id`	Нет (пункт **None**)	Целочисленный
`author`	`author`	Строковый (пункт **Text**)
`name`	`name`	Строковый
`added`	`added`	Дата (пункт **Date**)
`href`	`href`	Строковый
`catid`	`category`	Целочисленный (пункт **Integer**)

Закончив ввод данных, можно нажать кнопку **OK**. После этого серверное поведение **Insert Record** будет создано.

Теперь давайте создадим гиперссылку на страницу Items_admin.php. Поставим текстовый курсор правее формы и нажмем клавишу <Enter>, чтобы создать пустой абзац. В этом пустом абзаце наберем текст `На список статей (файлов)` и превратим его в гиперссылку с таким вот интернет-адресом:

```
Items_admin.php?catid=<?php echo $_GET["catid"]; ?>
```

Остается только открыть страницу Items_admin.php, переключиться в режим отображения кода HTML, найти код, создающий гиперссылку `Добавить`, и дать этой гиперссылке такой интернет-адрес:

```
Item_add.php?file=<?php echo $row_Category['file']; ?>&catid=<?php echo
↳$row_Category['id']; ?>
```

(Массив `$row_Category` содержит значения полей набор записей `Category`, созданного на этой странице. А сам набор `Category` содержит всего одну запись — выбранную категорию.)

Что касается страниц для правки и удаления статьи (файла), то они создаются аналогично страницам для правки и удаления категории. Этим страницам нам придется передавать методом `GET` целых три аргумента: `file` (для выбора нужных категорий), `id` (для выбора исправляемой или удаляемой статьи или файла) и `catid` (для корректного возврата на список статей или файлов). Также для вывода значения даты в единственном поддерживаемом MySQL формате "год-месяц-число" нужно будет использовать такое вот выражение PHP:

```
<?php echo date("Y-m-d", strtotime($row_Item['added'])); ?>
```

Впрочем, на страницу удаления статьи (файла) Item_delete.php дату можно выводить и в привычном для нас формате "число.месяц.год" — мы ведь не будем его там изменять, а значит, его не придется снова записывать в таблицу. Для этого мы используем уже знакомый нам сценарий

```
<?php echo date("d.m.Y", strtotime($row_Item['added'])); ?>
```

Еще на странице Item_delete.php нам понадобится вывести название категории, к которой относится удаляемая статья (файл). Для этого мы создадим набор записей, извлекающий единственную запись таблицы `categories`, значение поля `id` которой равно значению аргумента `catid`, переданного методом `GET`.

И не забудем создать гиперссылки `Изменить` и `Удалить` на странице Items_admin.php.

Здесь не приводится полное описание процесса создания страниц Item_edit.php и Item_delete.php — будем считать это нашим домашним заданием и постараемся выполнить его сами.

Последнее, что нам останется сделать, — это поместить на главную страницу нашего сайта default.htm гиперссылки, указывающие на административную страницу списка категорий Categories_admin.php. Ведь набирать в строке адреса Web-обозревателя адрес, указывающий на эту страницу, очень утомительно, а набрать адрес сайта и щелкнуть по гиперссылке намного проще и быстрее.

Частный случай ввода данных — поисковая машина

Каждый уважающий себя Web-сайт, созданный на основе серверных программ, имеет средства поиска данных — так называемую *поисковую машину*. Поисковая машина принимает от посетителя критерии поиска, ищет данные, удовлетворяющие этим критериям, и выводит результат на особую Web-страницу (*страницу результатов поиска*).

Реализация поисковой машины — это, можно сказать, частный случай ввода данных. В самом деле, посетитель сайта должен сначала ввести нужные ему критерии поиска, чтобы поисковая машина знала, что ей искать. Подход "принеси то, не знаю что" не работает ни в сказках, ни в реальной жизни...

Давайте для нашего простенького сайта используем один-единственный критерий поиска — ключевое слово, которое должно встретиться в названии статьи или файла. Сценарий PHP, который мы создадим (точнее, его создаст Dreamweaver), будет выполнять поиск записей таблицы `items`, поле `name` которых содержит это слово. Для ввода ключевого слова мы используем форму с полем ввода и кнопкой, которую поместим на главной странице нашего сайта default.htm.

Сказано — сделано. Открываем страницу default.htm, добавляем на нее новый абзац с поясняющим текстом и вставляем в него форму по имени `search`. В этой форме создаем поле ввода с именем `keyword` и кнопку отправки данных `submit` с надписью `Найти`.

А теперь — внимание! Нам придется задать параметры формы самим — Dreamweaver этого за нас не сделает. Итак, поставим текстовый курсор где-нибудь внутри формы, щелкнем кнопку **<form>** секции тегов, чтобы выделить форму, и обратимся к редактору свойств (см. рис. 9.5). Параметры формы `search` будут такие:

- имя формы (поле ввода **Form name**) — `search` (мы его, собственно, уже задали);
- серверная страница, которая должна получить данные из формы (поле ввода **Action**), — Result.php (мы создадим ее потом);
- метод передачи данных (раскрывающийся список **Method**) — GET (одноименный пункт этого списка).

Для передачи данных мы используем метод `GET` — обычно ключевые слова невелики и уж точно не являются секретными.

Сохраним переделанную страницу default.htm и закроем. И создадим страницу Result.php, в которой будут отображаться результаты поиска. Дадим ее название `Результаты поиска`, введем такой же заголовок и какой-либо поясняющий текст. И сохраним эту страницу в корневой папке сайта — ведь к ней должны иметь доступ обычные пользователи (гости).

Теперь нам нужно создать набор записей, который будет содержать записи из таблицы `items`, значение поля `name` которых содержит введенное нами в форму `search` ключевое слово. Назовем этот набор записей `Result`.

Теперь нам нужно задать критерий фильтрации записей. Фильтровать их будем по полю `name`, так что выбираем в раскрывающемся списке **Filter** пункт **name**. Значение поля `name` должно содержать введенное посетителем ключевое слово, поэтому выбираем в раскрывающемся списке справа пункт **contains**. (Этот пункт включает в запрос особый оператор сравнения `LIKE`, который возвращает `true`, если указанная слева от него строка содержит строку, которая указана справа.) Само ключевое слово будет браться из аргумента, переданного методом `GET`, поэтому выбираем в нижнем раскрывающемся списке пункт **URL Parameter**. А имя этого аргумента — `keyword` — вводим в поле ввода, расположенное справа и снизу.

Теперь остается задать только сортировку по полю `added` и по убыванию — набор записей `Result` готов. Но давайте его немного усложним. Пусть он выдает, кроме всего прочего, название категории, в которой находится статья или файл. Для этого нам придется связать в одном запросе SQL две таблицы (о связывании таблиц см. *главу 6*).

Щелкнем по кнопке **Advanced** диалогового окна **Recordset** — и оно переключится в расширенный режим (см. рис. 8.15). В поле ввода **SQL** мы видим код уже созданного Dreamweaver запроса. Вот он:

```
SELECT * FROM items WHERE name LIKE '%colname%' ORDER BY added DESC
```

Здесь `colname` — это переменная SQL-запроса, вместо которой будет подставлено значение аргумента `keyword`. А шаблон `%` обозначает любое количество любых символов (в том числе, и их отсутствие).

Увы — здесь Dreamweaver ничем нам помочь не сможет. Нам придется ввести код своего SQL-запроса в это поле вручную. Наш новый запрос будет выглядеть так:

```
SELECT items.*, categories.name AS cat, categories.file
↳FROM items, categories WHERE items.catid=categories.id AND
↳items.name LIKE '%colname%' ORDER BY items.added DESC
```

После этого можно нажать кнопку **OK**.

Теперь создадим на странице Result.php таблицу из пяти колонок. Первая колонка будет содержать имя автора, вторая — название, третья — дату добавления, четвертая — название категории, а пятая — гиперссылку, открывающую статью или файл. В ячейках первой строки таблицы напишем названия колонок, а в ячейках второй поместим динамический текст, выводящий значения соответствующих полей нашего набора записей. Далее создадим повторяющуюся область, включающую в себя вторую строку таблицы, — и дело сделано.

Сейчас, в принципе, можно сохранить страницу и проверить готовую поисковую машину в действии. Но давайте еще немного ее улучшим. Сделаем так, чтобы в четвертом столбце после названия категории в скобках выводилась строка `статьи` или `файлы`.

Переключимся в режим отображения кода HTML и найдем сценарий PHP, выводящий название категории. Вот он:

```
<?php echo $row_Result['cat']; ?>
```

Чтобы добавить к названию категории строку `статьи` или `файлы`, нам нужно проверить значение логического поля `file`. Если оно будет содержать `0` (элементы массива `$row_Result` всегда имеют строковый тип, а `0` — числовое представление логического значения `false`), нужно вывести статьи, если `1` — файлы. Поэтому дополним приведенный ранее сценарий вот так (добавленный код выделен полужирным шрифтом):

```
<?php echo $row_Result['cat'] . " (" . (($row_Result['file'] == "1") ?
↳"файлы" : "статьи") . ")"; ?>
```

Вот теперь действительно все! И готовую поисковую машину можно проверить в "боевых" условиях.

Конечно, Web-страница результатов поиска Result.php — далеко не идеал. Например, если поиск не дал результатов, таблица все равно выводится на экран. Но это дело поправимое — достаточно создать две необязательных области, как на странице Items.php (см. *главу 8*). Также можно превратить название категории в гиперссылку, указывающую на страницу Items.php со статьями (файлами), входящими в данную категорию. Вообще можно много чего сделать, главное — вовремя остановиться!

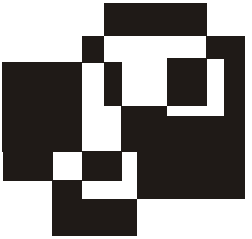
Что дальше?

Что ж, можно сказать, что наш сайт Site2 практически готов. Мы создали пользовательские страницы, выводящие информацию из базы данных. Мы создали административные страницы, позволяющие добавлять новые записи,

править существующие и удалять ненужные. Мы даже создали свою поисковую машину, чем не каждый сайт может похвастаться. Так чего же еще желать?

Да, мы знаем, как создать простейший Web-сайт на основе серверных страниц. Но именно простейший! Мы создали сайт, который поддерживает только самые основные функции: ввод-вывод данных и простейший поиск. На самом деле, мы не знаем еще очень многого.

"Холодный душ для команды" — как, кажется, говорят футболисты. Так вот, в следующей главе мы примем холодный, ледяной душ. Ведь мы совсем не позаботились о безопасности своего сайта!



Часть III

Решаем вопросы безопасности и целостности данных

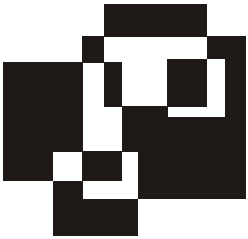
Глава 10. Введение в безопасность и целостность данных

Глава 11. Разграничение доступа

Глава 12. Поддержание ссылочной целостности данных

Безопасности сейчас уделяется очень много внимания. В компьютеры встраиваются считыватели папиллярных узоров пальцев, программы и Web-сайты ощетиниваются парольными защитами, файлы прячутся на скрытых зашифрованных дисках, а серверные комнаты охраняются немногословными ребятами в камуфляже и с дубинками у пояса. Что ж, это оправданно — охотники за ценной информацией были, есть и будут всегда, и пока есть спрос на их услуги, будут существовать разнообразные технологии защиты данных.

Настала пора и нам позаботиться о безопасности нашего сайта. Конечно, вряд ли на хранящуюся в базе данных `site` информацию покусится кто-то серьезный. Но обыкновенных вандалов, которых хлебом не корми, а дай покопаться в чужих данных, сбрасывать со счетов не стоит. Себе дороже...



Глава 10

Введение в безопасность и целостность данных

Вторая часть этой книги была весьма объемной. В ней мы узнали о базах данных, познакомились с сервером данных MySQL, освоили программирование сценариев PHP и создали в Dreamweaver с десяток серверных Web-страниц, составивших неплохо работающий Web-сайт. Фактически мы занимались серверным программированием, и эти занятия принесли определенные плоды.

Но чем нам заняться теперь? Публиковать наш сайт в Сети? Ведь, кажется, мы предусмотрели все, что нужно для его работы и сопровождения: не только пользовательские, но и административные страницы, служащие для пополнения сайта, и даже поиск. Чего еще желать?

Нет, в Сеть нашему сайту пока еще рано. И все из-за того, что мы совсем не подумали о двух вещах: безопасности и целостности данных. Вот о них-то и пойдет речь во всей *третьей части* этой книги — настолько все это важно.

В данной главе мы поговорим о том, что может подстерегать наши данные на "диких просторах" Интернета. И начнем с того, что закроем доступ к святая святых любого сайта, использующего базы данных, — его административным страницам — для всех, кто захочет без нашего ведома добраться до них.

Безопасность и разграничение доступа

Давайте запустим Web-сервер и откроем наш сайт в Web-обозревателе. Когда он выведет главную страницу сайта, посмотрим на нее критически. Что там не так?

Сразу же бросается в глаза то, что гиперссылка, указывающая на административную страницу списка категорий Categories_admin.php, находится прямо на главной странице и доступна всем. Это значит, что любой посетитель нашего сайта сможет щелкнуть по ней и попасть туда, куда ему попадать не положено. Что он там натворит, зависит только от его фантазии и испорченности.

А нам это совсем не нужно. Мы сами хотим править список категорий и добавлять новые статьи и файлы, не давая эту привилегию никому (ну, может быть, за исключением только проверенных людей, которым можно доверять). Сайт наш, и только мы имеем право его администрировать!

Так что делать? Как закрыть доступ к административным страницам для всех, кроме немногих избранных? Одним словом, как реализовать разграничение доступа?

Из *главы 1* мы уже знаем, как это делается в программах-серверах. Каждая такая программа имеет особый список, куда вносятся имена и пароли пользователей, имеющих право подключаться к этому серверу. При попытке подключения сервер запрашивает у посетителя имя и пароль и, получив их, проверяет, есть ли они в списке. Если есть — добро пожаловать, уважаемый пользователь, мы вам всегда рады! Если же нет, то все зависит от самого сервера и его настроек.

Можно сделать так, что пользователь, чьи имя и пароль не внесены в список, вообще не имеет права к нему подключаться. Так обычно делается на серверах данных и серверах FTP, хранящих "закрытые" файлы. В этом случае пользователю возвращается сообщение об ошибке подключения, и соединение, естественно, не устанавливается.

Часто делается по-другому: незарегистрированный пользователь все-таки может подключиться к серверу, но при этом получает самые минимальные ("гостевые") права. Это касается всех Web-серверов, очень многих серверов FTP, некоторых серверов данных (в частности, нашего старого знакомого MySQL) и пр.

Но все это детали, главное — сам подход: зарегистрированные в специальном списке пользователи, запрос имени и пароля. Реализовать его весьма просто, поэтому он используется и для закрытия доступа к административным страницам Web-сайтов.

Итак, давайте составим список того, что нам нужно сделать.

1. Создать в нашей базе данных `site` новую таблицу — список пользователей. Эта таблица будет включать поля, содержащие имена пользователей, их пароли и, возможно, сведения о правах доступа к различным страницам сайта.
2. При попытке какого-либо посетителя сайта открыть административную (или вообще "закрытую") страницу спросить у него имя и пароль.
3. Если имя и пароль, введенные посетителем, имеются в списке пользователей, то перенаправить его на главную страницу сайта, на которой сделать доступными гиперссылки на административные страницы.
4. Если же таких имени и (или) пароля нет в списке пользователей, то отправить посетителя на ту же главную страницу. Но гиперссылки, указы-

вающие на административные страницы, при этом должны оставаться скрытыми.

5. Когда пользователь закончит работу с административными страницами, предоставить ему возможность корректно выйти с сайта.
6. И не забыть создать еще один набор административных страниц, предназначенных для управления зарегистрированными пользователями. Этот набор будет включать четыре страницы: страница — список пользователей, страницы добавления, правки и удаления пользователя.

Зачем нужно реализовывать корректный выход с сайта? Дело в том, что нам придется хранить сведения о том, что данный посетитель сайта является зарегистрированным пользователем и что ему нужно будет открыть все "закрытые" Web-страницы. А после того как он закончит работу, эти сведения должны быть удалены из памяти компьютера — это, собственно, и называется выходом с сайта.

Давайте еще немного усложним нашу задачу. Пусть у нас будут два вида зарегистрированных пользователей: администраторы и ведущие. Администраторы будут иметь полные права на доступ ко всем административным страницам сайта: списку категорий, списку пользователей, а также спискам статей и файлов. Ведущие же пусть имеют доступ только к списку категорий, спискам статей и файлов, но не к списку пользователей. Для хранения указаний на то, является ли пользователь администратором или ведущим, мы создадим особое поле в таблице списка пользователей базы данных `site`.

Ну, вот и все о разграничении доступа. В *главе 11* мы займемся его практической реализацией.

Замечание

Существует также возможность реализовать разграничение доступа средствами Web-сервера. Практически все серьезные Web-серверы предоставляют возможность ограничить доступ к определенным папкам и файлам. О том, как это сделать, описано в документации по данному Web-серверу.

Целостность данных

Предположим, что мы уже разграничили доступ к страницам нашего сайта. Администраторы и ведущие, соответственно, администрируют и ведут, гости приходят и уходят, все замечательно работает, и все довольны. И вдруг случается страшный конфуз!..

Какой-то незадачливый ведущий заходит на страницу списка категорий (административную, разумеется, — чего он не видел на пользовательской!) и удаляет какую-то категорию. И все было бы прекрасно, если бы к этой категории не относилось несколько статей (файлов). И теперь к этим статьям (файлам) нет никакого доступа! Они "повисли в воздухе", "осиротели"!!!

Из *главы 6* мы знаем, что при наличии связи между двумя таблицами — первичной и вторичной — одной или более записям вторичной таблицы обязательно должна соответствовать одна запись первичной таблицы. В этом случае специалисты по базам данных говорят, что *ссылочная целостность* или просто *целостность* данных соблюдена. Если же во вторичной таблице присутствуют записи, которым не соответствует ни одной записи в таблице первичной, то ссылочная целостность нарушается.

Собственно, первичная таблица потому и называется первичной, что именно она создает эту самую ссылочную целостность. Запись в первичной таблице — это своего рода пропуск к записям вторичной таблицы; если же она удалена, то ссылающиеся на нее записи вторичной таблицы автоматически становятся недоступными.

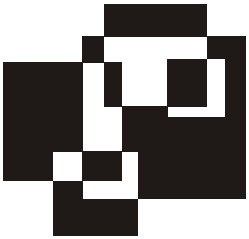
Нарушение ссылочной целостности — это настоящая трагедия для базы данных. Стройная структура связей рушится на глазах. Программы не могут получить доступ к целой группе записей. Посетители сайта в ярости, администратор — в панике. Что же делать?

Вообще-то, подавляющее большинство серверов данных сами следят за ссылочной целостностью и не позволяют пользователям ее нарушать. То есть удалить запись первичной таблицы, на которую ссылаются записи вторичной, нельзя — сервер откажется выполнять запрос на удаление записи и выдаст сообщение об ошибке. Но MySQL не таков — он все норовит переложить на плечи программиста. На наши плечи...

Самый простой выход, сразу приходящий на ум, — перед удалением записи из первичной таблицы проверять, есть ли ссылающиеся на нее записи из вторичной. Если нет, значит, запись можно удалить, если есть — соответственно, нельзя. Практически все разработчики серверных программ так и поступают; так же поступим и мы в *главе 12*, когда начнем реализовывать в своих страницах сохранение ссылочной целостности наших данных.

Что дальше?

А дальше — практические занятия! Не будем терять времени: наши данные открыты, и злоумышленники уже подбираются к ним. За работу!



Глава 11

Разграничение доступа

В *главе 10* мы перечислили все, что нам нужно для разграничения доступа, здесь же будет описано, как реализовать это для нашего сайта. А именно как закрыть от нежелательных посетителей административные Web-страницы. Давайте вспомним, что нам для этого необходимо.

Первым же делом нам будет нужно создать таблицу, где будет храниться список зарегистрированных пользователей — администраторов и ведущих. Администраторы будут иметь доступ ко всем административным страницам, ведущие — только к спискам категорий и статей с файлами. Соответственно, администраторы смогут править списки категорий, статей (файлов) и пользователей, а ведущие — только списки категорий и статей (файлов).

Следующий шаг — это создание страницы для входа на сайт. Эта страница будет содержать форму для ввода имени пользователя и пароля, а также сценарии PHP, выполняющие поиск введенных имени и пароля в таблице списка пользователей. Если такой пользователь есть в списке, ему будут открыты все страницы сайта, в противном случае он получит только права гостя (обычного посетителя).

Потом нам придется добавить в уже созданные в *главе 9* административные страницы особые сценарии PHP, которые будут проверять, зарегистрированный ли пользователь на них зашел и имеет ли он права на эти страницы. В этом нам поможет Dreamweaver — он создаст за нас особое серверное поведение, которое и выполнит проверку на "легальность" посетителя.

Последний — четвертый — шаг будет заключаться в создании страницы для корректного выхода с сайта. После того как зарегистрированный пользователь выполнит процедуру выхода, сайт "забудет" о том, что он зарегистрирован в списке пользователей, и будет считать его обычным гостем. Если же этот посетитель снова захочет зайти на административные страницы, он должен будет выполнить процедуру входа заново.

Ну вот и все. Теперь мы знаем, что нам нужно сделать. Можно приступать к работе.

Создание таблицы списка пользователей

Запустим сервер MySQL (если он еще не запущен) и воспользуемся программой клиента данных (например, описанным в *приложении 4* phpMyAdmin) для того, чтобы создать в базе данных `site` новую таблицу. Эта таблица будет хранить список зарегистрированных пользователей нашего сайта. Назовем ее `users`.

Структура нашей новой таблицы показана в табл. 11.1. Обратим внимание, что мы создали в ней отдельное поле для хранения сведений о правах пользователя. Это строковое поле размером в один символ будет содержать букву `a` для администратора и букву `m` для ведущего (от английского moderator — ведущий).

Таблица 11.1. *Структура таблицы* `users` *(списка пользователей)*

Имя поля	Описание поля	Тип данных поля	Описание типа данных
`id`	Идентификатор записи	`SMALLINT UNSIGNED`	Целые числа от 0 до 65 535
`name`	Имя пользователя	`VARCHAR(20)`	Строки длиной до 20 символов
`password`	Пароль	`VARCHAR(20)`	
`rights`	Права доступа	`VARCHAR(1)`	Строки длиной в 1 символ

Полю `rights` дадим значение по умолчанию — символ `m`. Пусть все вновь созданные пользователи будут ведущими, а уж тех, кого надо, мы сделаем администраторами сами.

На основе поля `id` создадим ключевой индекс. Второй индекс — уникальный — мы создадим на основе полей `name` и `password` и назовем его `np`.

После этого добавим в свежесозданную таблицу одну-единственную запись, задающую администратора, под именем которого мы сможем потом войти на сайт. Пусть этот администратор имеет имя `admin`, пароль `admin`, а права — разумеется, `a`.

Внимание

Поскольку наш сайт учебный и не содержит секретной информации, мы задали очень простые имя и пароль администратора. В "настоящих" сайтах нужно делать их как можно более сложными (по крайней мере, пароль). Будет лучше всего, если в качестве имени и пароля задать бессмысленные наборы букв и цифр, и чем длиннее, тем лучше.

Создание страницы входа на сайт

Таблица, где будут храниться все зарегистрированные пользователи, готова. Теперь можно приступать к созданию Web-страницы входа на сайт.

Процесс создания страницы входа на сайт в Dreamweaver

Запустим Dreamweaver и создадим в нем новую пустую страницу PHP. Зададим для нее название `Вход`, введем такой же заголовок и какой-либо текст, приглашающий ввести имя и пароль в форму. И сохраним эту страницу в папке Admin под именем Login.php.

Теперь нам нужно создать форму для ввода имени и пароля. Сначала создадим саму форму и дадим ей имя `login`. После этого поместим в нее два поля ввода и кнопку, не забыв также о поясняющих надписях. И зададим для полей ввода и кнопки необходимые параметры.

Параметры первого поля ввода:

- имя — `name`;
- ширина в символах — 20 (поле `name` таблицы `users` имеет размер 20 символов);
- максимальное количество символов — 20;
- тип — обычное поле ввода в одну строку;
- значение по умолчанию — ничего.

Параметры второго поля ввода:

- имя — `password`;
- ширина в символах — 20 (поле `password` таблицы `users` имеет размер 20 символов);
- максимальное количество символов — 20;
- тип — поле ввода пароля (переключатель **Password** группы **Type** редактора свойств (см. рис. 9.6));
- значение по умолчанию — ничего.

Ненадолго остановимся и поговорим о *поле ввода пароля*. Это обычное поле ввода в одну строку, единственное отличие которого заключается в том, что любой набранный в нем текст отображается в виде точек, таким образом сюда можно заносить секретные данные.

Осталось задать параметры для кнопки:

- имя — `submit`;
- надпись — `Войти`;
- действие, выполняемое кнопкой, — отправка данных.

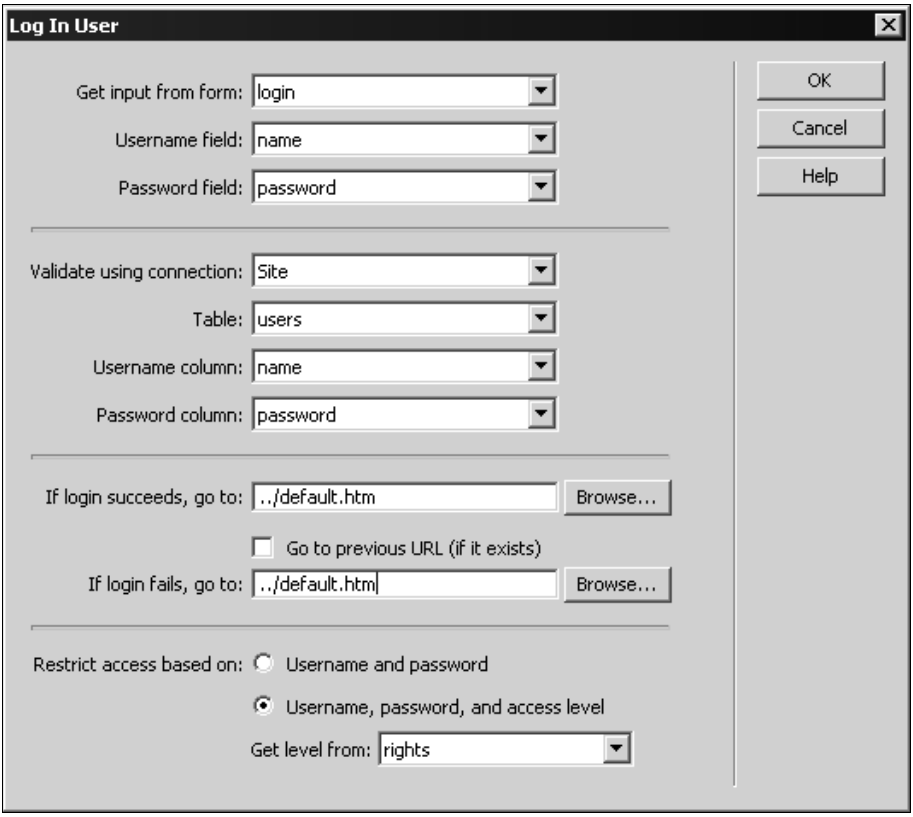


Рис. 11.1. Диалоговое окно **Log In User**

Итак, форма готова. Теперь выводим на экране панель **Server Behaviors**, нажимаем кнопку со знаком "плюс" и выбираем в появившемся на экране меню пункт **User Authentication**. После этого на экране появится подменю, в котором нам будет нужно выбрать пункт **Log In User**. И теперь зададим необходимые параметры в появившемся на экране диалоговом окне **Log In User**, показанном на рис. 11.1.

В раскрывающемся списке **Get input from form** задается форма, из которой будут взяты имя и пароль пользователя для проверки. Выберем в этом списке пункт **login** — имя нашей формы.

В раскрывающихся списках **Username field** и **Password field** выбираются поля ввода, из которых будут взяты, соответственно, имя и пароль. Выберем в первом списке пункт **name**, а во втором — **password**. Именно так называются соответствующие поля ввода нашей формы `login`.

Раскрывающийся список **Validate using connection** служит для задания зарегистрированной в Dreamweaver базы данных, содержащей таблицу списка пользователей. Нам будет нужно выбрать в этом списке пункт **Site**.

Раскрывающийся список **Table** служит для задания таблицы списка пользователей. В нашем случае — это таблица `users`.

Два раскрывающихся списка — **Username column** и **Password column** — служат для выбора полей таблицы списка пользователей, из которых будут выбираться, соответственно, имена и пароли. В нашем случае — это поля `name` и `password`.

Поле ввода **If login succeeds, go to** служит для задания Web-страницы, на которую будет выполнен переход в случае удачного входа (если введенные имя и пароль есть в списке пользователей). Нажмем кнопку **Browse**, выберем в появившемся на экране диалоговом окне открытия файла Dreamweaver главную страницу нашего сайта default.htm и нажмем кнопку **OK** этого окна.

А поле ввода **If login fails, go to** служит, наоборот, для задания Web-страницы, на которую будет выполнен переход в случае неудачного входа (если введенные пользователем имя и пароль в списке пользователей отсутствуют — на наш сайт пытается попасть "самозванец"). Опять же, нажмем кнопку **Browse**, выберем в появившемся на экране диалоговом окне открытия файла Dreamweaver главную страницу default.htm и нажмем кнопку **OK**.

Но зачем нужен флажок **Go to previous URL (if it exists)**? И почему мы его пропустили? Давайте разберемся.

Может случиться так, что какой-либо пользователь перейдет к одной из административных страниц напрямую, набрав в строке адреса Web-обозревателя прямой интернет-адрес вида:

http://localhost:8080/Admin/Categories_admin.php

Серверное поведение, которое мы создадим позже на этой странице, перенаправит его на страницу входа Login.php. После того как пользователь введет в форму свои имя и пароль, страница входа отправит его на главную страницу нашего сайта default.htm. Но если мы включим флажок **Go to previous URL (if it exists)**, она отправит его обратно на ту страницу, куда он пытался попасть, — на Categories_admin.php.

Так почему бы не включить этот флажок? Дело в том, что PHP-код, который в этом случае Dreamweaver добавит в создаваемое нами серверное поведение, не будет работать правильно. Вероятно, это ошибка в самом Dreamweaver. Поэтому оставим флажок **Go to previous URL (if it exists)** не включенным.

Давайте лучше обратим внимание на группу переключателей **Restrict access based on**. С ее помощью мы можем задать, будет ли допуск на закрытые страницы сайта предоставляться только на основе имени и пароля (переключатель **Username and password**) или на основе имени, пароля и прав доступа (переключатель **Username, password, and access level**). Поскольку мы хотим предоставлять доступ к закрытым страницам нашего сайта на основе имени, пароля и прав доступа (мы даже создали специальное поле `rights`

в таблице `users`), то должны будем включить переключатель **Username, password, and access level**.

Если мы хотим, чтобы серверное поведение учитывало права доступа пользователя, то оно должно будет откуда-то взять эти права. Поле таблицы, из которого эти права будут взяты, задается в раскрывающемся списке **Get level from**. Выберем в нем пункт **rights**.

Теперь можно нажать кнопку **OK** диалогового окна **Log In User**. Через некоторое время серверное поведение **Log In User** будет создано. Теперь нам осталось только создать гиперссылку, указывающую на главную страницу сайта (на случай, если самозванец, забравшийся на страницу входа, смог-таки достойно капитулировать), и сохранить полностью готовую страницу Login.php.

Давайте пока подождем копировать ее в корневую папку Web-сервера. Вообще, оставим компьютер в покое и немного поговорим о хранении данных в PHP.

Сессии. Переменные уровня сессии

Опишем работу серверного поведения **Log In User**, которое мы только что создали. Не будем рассматривать случай неуспешного входа — тут все понятно. Давайте предположим, что пользователь выполнил успешный вход на сайт, то есть его имя и пароль были найдены в списке пользователей (таблице `users`).

1. Имя и пароль пользователя (а также, если это было особо указано, и его права) сохраняются для дальнейшего использования.
2. При перемещении на закрытую страницу другое серверное поведение (мы создадим его потом) проверяет, было ли сохранено имя пользователя (и его права) ранее. При этом подразумевается, что это имя присутствует в списке пользователей — серверное поведение **Log In User** его уже проверило.
3. Если имя пользователя было сохранено, проверяются права этого пользователя. (Здесь мы предполагаем, что доступность различных закрытых страниц зависит от прав пользователя; собственно, в нашем случае так и есть.) Далее решение, пускать или не пускать пользователя на данную страницу, будет приниматься на основе его прав.
4. Когда пользователь выполняет выход с сайта, специальное (уже третье по счету) серверное поведение удаляет сохраненные имя, пароль и права. После чего для доступа к закрытым страницам этому пользователю снова придется выполнить процедуру входа.

В принципе, ничего особо сложного в этом нет. За одним исключением — где хранить имя, пароль и права пользователя? Ведь эти данные должны быть доступны всем серверным Web-страницам сайта, чтобы они смогли

проверить, разрешен ли этому пользователю доступ к ним. Такие данные в обычной переменной PHP не сохранишь — они будут доступны только текущей странице.

Можно, конечно, передавать данные о пользователе от одной страницы другой методом `GET`. Но это, во-первых, потребует лишнего кода PHP, а во-вторых, крайне небезопасно. Вспомним главную особенность метода `GET`, описанную в *главе 9*, — передаваемые им данные добавляются к интернет-адресу. А интернет-адрес отображается в строке адреса Web-обозревателя, где его сможет прочитать кто угодно.

Но выход из этого положения есть, и весьма изящный. Информация, которая должна быть доступна разным Web-страницам, может быть сохранена в так называемых переменных уровня сессии. Сейчас мы о них и поговорим.

В терминологии администраторов Web-серверов и Web-программистов *сессия* — это процесс нахождения посетителя на одном сайте. То есть когда посетитель путешествует по страницам одного сайта — это сессия. Если же посетитель уйдет на другой сайт, сессия заканчивается (и начинается сессия уже на другом сайте, но это уже проблемы другого сайта).

Практически все платформы для создания серверных программ поддерживают возможность хранения на Web-сервере данных, доступных для всех страниц в течение сессии. Эти данные сохраняются в особых переменных, которые так и называются — *переменные уровня сессии*. Когда посетитель уходит с сайта, серверная программа выжидает определенный промежуток времени — *таймаут* — и удаляет эти переменные, чтобы не занимать понапрасну память серверного компьютера и его диски.

Разумеется, PHP не исключение из этого правила и тоже позволяет создавать переменные уровня сессии. Серверные поведения, "ответственные" за вход пользователя и проверку его прав, хранят данные о пользователе как раз в таких переменных.

Перед созданием и использованием в сценариях PHP любых переменных уровня сессии нужно вызвать не принимающую аргументов встроенную функцию `session_start`. Эта функция дает обработчику PHP понять, что мы собираемся работать с переменными уровня сессии.

Все переменные уровня сессии PHP представляют собой элементы встроенного массива `$_SESSION`. Индекс каждого его элемента соответствует имени созданной переменной уровня сессии. Это значит, что для сохранения имени пользователя можно написать вот такой небольшой сценарий PHP:

```
$_SESSION["username"] = "Admin";
```

Здесь мы поместили в массив `$_SESSION` элемент с индексом `username`, создав тем самым переменную уровня сессии, имеющую имя `username`. После

этого мы можем использовать сохраненное в этой переменной значение где угодно:

```
if (isset($_SESSION["username"])) {
  echo $_SESSION["username"];
}
```

Просто и удобно!

К сожалению, приведенный ранее синтаксис для доступа к переменным уровня сессии работает только в версии 4.*x* PHP. В более старых версиях для создания переменной уровня сессия нужно использовать встроенную функцию `session_register`:

```
$username = "Admin";
session_register("username");
```

Заметим, что мы должны передать функции `session_register` имя переменной, которую мы хотим сделать переменной уровня сессии, в строковом виде. Значок доллара, которым должно предваряться имя любой переменной PHP, при этом указывать не нужно. Доступ же к созданной таким образом переменной уровня сессии выполняется также через массив `$_SESSION`.

Функция `session_register` поддерживается и в новых версиях PHP, но в них ее использование не рекомендовано. В самом деле, ведь намного удобнее и нагляднее просто создать новый элемент массива `$_SESSION`. А эта странноватая функция только все запутывает.

Однако Dreamweaver в своих серверных поведениях использует "старый" способ создания переменных уровня сессии. Это сделано для совместимости со всеми версиями PHP — и старыми, и новыми.

Разбор кода PHP, выполняющего вход

Теперь настала пора посмотреть, что же создал Dreamweaver в ответ на наши священнодействия с диалоговым окном **Log In User**. Рассмотрим созданный им код PHP по частям.

```
session_start();
$loginFormAction = $_SERVER['PHP_SELF'];
```

Здесь мы видим вызов уже знакомой нам функции `session_start`. Следующее выражение присваивает переменной `$loginFormAction` имя текущей Web-страницы, которое будет подставлено в код HTML, создающий форму `login`. Это тоже нам знакомо.

```
if (isset($accesscheck)) {
  $GLOBALS['PrevUrl'] = $accesscheck;
  session_register('PrevUrl');
}
```

А это выражение — явно лишнее. Мы видим, что оно с помощью встроенной функции `isset` проверяет, существует ли переменная `$accesscheck`. Нигде в сценариях текущей страницы данная переменная не объявлена, а значит, тело этого условного выражения никогда выполнено не будет. Еще одна ошибка в Dreamweaver?..

Пропустим большой фрагмент кода PHP и HTML и посмотрим на HTML-код формы.

```
<FORM ACTION="<?php echo $loginFormAction; ?>" METHOD="POST" NAME="login"
↳ID="login">
  <P>Имя
    <INPUT NAME="name" TYPE="text" ID="name" SIZE="20" MAXLENGTH="20">
  </P>
  <P>Пароль
    <INPUT NAME="password" TYPE="PASSWORD" ID="password" SIZE="20">
  </P>
  <P>
    <INPUT NAME="submit" TYPE="submit" ID="submit" VALUE="Войти">
  </P>
</FORM>
```

Опять ничего нового! В качестве значения атрибута `ACTION` тега `<FORM>` подставляется имя текущей страницы (значение переменной `$loginFormAction`); это значит, что для проверки, есть ли пользователь в списке, будет использован сценарий, находящийся в этой же странице.

Пожалуй, посмотреть стоит только на тег `<INPUT>`, создающий поле ввода пароля. Атрибут `TYPE` этого тега имеет значение `password` — запомним это.

Хорошо! Пользователь ввел в форму имя и пароль и нажал кнопку **Войти**. Перенесемся в начало кода страницы и посмотрим на сценарий, выполняющий проверку существования такого пользователя в списке. Этот сценарий столь велик, что мы рассмотрим его по частям.

```
if (isset($_POST['name'])) {
```

Сначала проверяется, передан ли этой странице методом `POST` аргумент `name`. Если такой аргумент был передан, то данные в форму уже введены, и нужно выполнить проверку пользователя. Если же такого аргумента нет, то данные еще не были введены, а значит, нужно вывести на экран форму для их ввода.

```
  $loginUsername=$_POST['name'];
  $password=$_POST['password'];
```

Сохраняем введенные имя и пароль в переменных `$loginUsername` и `$password` соответственно.

```
$MM_fldUserAuthorization = "rights";
$MM_redirectLoginSuccess = "../default.htm";
$MM_redirectLoginFailed = "../default.htm";
$MM_redirecttoReferrer = false;
```

А здесь объявляются переменные, задающие поле таблицы, где хранятся права пользователя (`$MM_fldUserAuthorization`), Web-страницы, на которые будет выполнен переход в случае успешного и неуспешного входа (`$MM_redirectLoginSuccess` и `$MM_redirectLoginFailed` соответственно). Последняя переменная (`$MM_redirecttoReferrer`) задает, будет ли после успешного входа выполнен возврат на предыдущую страницу (ее значение равно `false` — вспомним отключенный флажок **Go to previous URL (if exists)** в диалоговом окне **Log In User**).

```
mysql_select_db($database_Site, $Site);
$LoginRS__query = sprintf("SELECT name, password, rights FROM users
↳WHERE name='%s' AND password='%s'", get_magic_quotes_gpc() ?
↳$loginUsername : addslashes($loginUsername), get_magic_quotes_gpc() ?
↳$password : addslashes($password));
$LoginRS = mysql_query($LoginRS__query, $Site) or die(mysql_error());
```

Здесь выполняется подключение к базе данных и поиск в таблице `users` пользователя с заданными именем и паролем. Для поиска используется запрос SQL, который создает самое длинное выражение в этом фрагменте сценария.

```
$loginFoundUser = mysql_num_rows($LoginRS);
if ($loginFoundUser) {
```

Извлекаем содержимое первой записи возвращенного в результате выполнения запроса SQL набора в массив `$loginFoundUser` и проверяем, не пуст ли этот массив (то есть были ли в наборе записи или, другими словами, найден ли пользователь с такими именем и паролем). Если массив не пуст (пользователь найден), то выполняется следующий код PHP.

```
    $loginStrGroup = mysql_result($LoginRS, 0, 'rights');
```

Функция `mysql_result` возвращает содержимое заданного поля записи с заданным номером. Первым аргументом ей передается идентификатор набора записей, вторым — номер записи, третьим — строка с именем поля. В приведенном ранее выражении эта функция извлекает значение поля `rights` (права пользователя) и помещает его в переменную `$loginStrGroup`.

(Непонятно, зачем вообще нужно это выражение — можно было воспользоваться уже объявленным массивом `$loginFoundUser`. К тому же, как говорят

сами разработчики PHP, функция `mysql_result` работает медленнее, чем `mysql_num_rows`.)

```
$GLOBALS['MM_Username'] = $loginUsername;
$GLOBALS['MM_UserGroup'] = $loginStrGroup;
session_register("MM_Username");
session_register("MM_UserGroup");
```

Сохраняем имя и права пользователя в переменных уровня сессии `MM_Username` и `MM_UserGroup` соответственно.

Что касается встроенного массива `$GLOBALS`, то он используется для объявления глобальных переменных вместо ключевого слова `global`. (О глобальных переменных см. *главу 7*.)

```
if (isset($_SESSION['PrevUrl']) && false) {
  $MM_redirectLoginSuccess = $_SESSION['PrevUrl'];
}
```

А это выражение опять лишнее. Да что такое случилось с разработчиками фирмы Macromedia! Давайте посмотрим — в условии используется оператор логического И `&&`, один из аргументов которого равен `false`. Это значит, что при любом раскладе результат выполнения этого оператора будет равен `false` и это условное выражение выполнено не будет.

```
  header("Location: " . $MM_redirectLoginSuccess );
}
```

Последнее, что сценарий PHP сделает в случае успешного входа, — выполнит перенаправление на страницу, чей интернет-адрес хранится в переменной `$MM_redirectLoginSuccess`. Эта переменная была объявлена ранее.

```
else {
  header("Location: ". $MM_redirectLoginFailed );
}
}
```

А в случае неуспешного входа (то есть пользователя с такими именем и паролем в таблице `users` найдено не было) — добро пожаловать, многоуважаемый "самозванец", на страницу, чей интернет-адрес записан в переменную `$MM_redirectLoginFailed`. Эта переменная также была объявлена ранее.

Вот и весь сценарий. И если бы не целые "куски" ненужного кода PHP, вставленного зачем-то Dreamweaver, все было бы замечательно. Но — увы! — в мире нет ничего совершенного...

Разграничение доступа к закрытым Web-страницам

Так, страница входа на сайт готова. Наш следующий шаг — закрытие доступа к административным страницам.

Процесс разграничения доступа к страницам в Dreamweaver

Всего административных страниц у нас восемь. Это страница списка категорий, страница списка статей и файлов, а также по три страницы, предназначенные для добавления, правки и удаления соответственно категорий и статей с файлами. Придется потрудиться.

Начнем мы со страницы списка категорий Categories_admin.php. Откроем ее в Dreamweaver. И — уже привычная процедура: выводим на экран панель **Server Behaviors**, нажимаем кнопку со знаком "плюс" и выбираем в подменю **User Authentication** появившегося на экране меню пункт **Restrict Access To Page**. На экране появится диалоговое окно **Restrict Access To Page** (рис. 11.2).

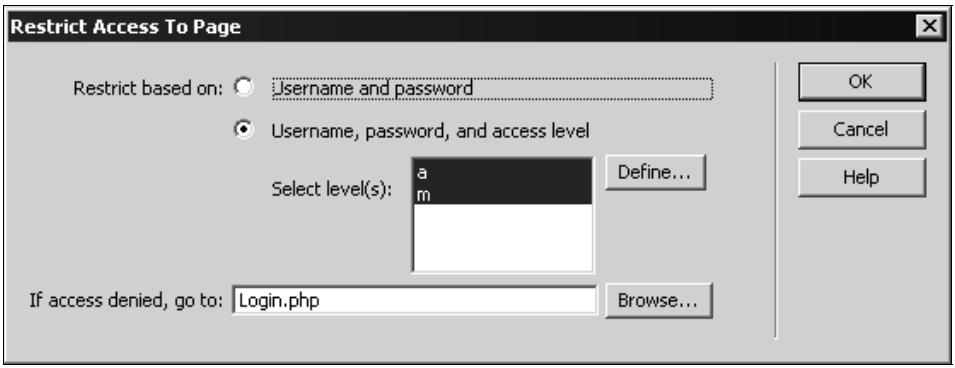


Рис. 11.2. Диалоговое окно **Restrict Access To Page**

В этом окне мы видим уже знакомую нам группу переключателей **Restrict based on**, задающую, на основании чего будет производиться допуск или недопуск на данную страницу. Переключатель **Username and password** заставляет серверное поведение проверять только имя и пароль, а переключатель **Username, password, and access level** — еще и права доступа. Его-то мы и включим.

Сразу же после этого станет доступным список **Select level(s)**, где задаются права, которые должен иметь пользователь, чтобы попасть на эту страницу. Изначально список пуст, поэтому нам сначала придется его заполнить.

Для этого нажмем кнопку **Define**, после чего на экране появится диалоговое окно **Define Access Levels** (рис. 11.3).

Рис. 11.3. Диалоговое окно **Define Access Levels**

В этом окне находится большой список уже введенных прав доступа, изначально, правда, пустой. Ниже его имеется поле ввода **Name**, где и вводятся обозначения прав. Давайте введем туда букву `a` (права администратора нашего сайта) и щелкнем по любому месту в списке прав (лучше всего — по пустому пространству). Ага — в списке появился пункт **a**! Дело почти сделано.

Теперь нам нужно добавить еще и права ведущего (букву `m`). Для этого щелкнем кнопку со значком "плюс", введем в поле ввода **Name** букву `m` и щелкнем по списку прав. Все, теперь список содержит два пункта — **a** и **m**, — обозначающих права администратора и ведущего.

Если мы ошиблись при вводе буквы, это легко исправить. Выделим в списке прав неправильный пункт, введем в поле ввода **Name** нужную букву и снова щелкнем по списку. А удалить лишний пункт совсем просто — выделяем его и щелкаем кнопку со знаком "минус".

Задав необходимые права, мы можем нажать кнопку **OK**. После этого мы снова окажемся в диалоговом окне **Restrict Access To Page**, где продолжим задание параметров одноименного серверного поведения.

Да, в списке **Select level(s)** присутствуют оба нужные нам пункта — **a** и **m**. Но Dreamweaver это ничего не говорит — может, мы сейчас ввели их просто так, а использовать не собираемся (заготовили их на будущее). Нам придется выбрать оба эти пункта. Щелкнем по одному из них, нажмем клавишу <Ctrl> и щелкнем по другому — они будут выбраны, как на рис. 11.2.

Последнее, что нам осталось сделать, — это задать Web-страницу, на которую будет отправлен пользователь, не обладающий нужными правами или просто не выполнивший процедуру входа. Щелкнем по кнопке **Browse** правее поля ввода **If access denied, go to**, выберем в появившемся на экране

диалоговом окне открытия файла Dreamweaver страницу входа Login.php и нажмем кнопку **OK** этого окна. Пусть, прежде чем войти на данную страницу, пользователь выполнит вход на сайт.

Все, кажется, мы ничего не забыли... Нажимаем кнопку **OK**. Пара секунд "задумчивости" — и Dreamweaver создаст нужное серверное поведение.

Осталось закрыть доступ к остальным пяти административным страницам: Category_add.php, Category_edit.php, Category_delete.php, Items_admin.php, Item_add.php, Item_edit.php и Item_delete.php. Сделаем это точно так же, как и для страницы Categories_admin.php. После этого сохраним исправленные страницы и закроем их.

Кстати, в дальнейшем нам не придется вводить права доступа в диалоговое окно **Define Access Levels**. Dreamweaver запомнит их и подставит в список **Select level(s)** диалогового окна **Restrict Access To Page** сам. Нам останется только их выбрать в этом списке.

Разбор кода PHP, выполняющего разграничение доступа

Что ж, откроем страницу Categories_admin.php и посмотрим на созданный Dreamweaver код PHP. Это будет весьма поучительно.

Начнем с того, что для проверки прав пользователя Dreamweaver создаст функцию по имени `isAuthorized`. Выражение, объявляющее эту функцию, мы рассмотрим в первую очередь и, опять же, по частям.

```
function isAuthorized($strUsers, $strGroups, $UserName, $UserGroup) {
```

Итак, функция `isAuthorized` принимает четыре аргумента:

- `$strUsers` — список "разрешенных" имен пользователей, разделенных запятыми;
- `$strGroups` — список "разрешенных" прав доступа, разделенных запятыми;
- `$UserName` — имя пользователя, которое нужно проверить;
- `$UserGroup` — права пользователя, которые нужно проверить.

```
  $isValid = False;
```

Переменная `$isValid` содержит `true`, если пользователь (его имя или права) входит в список разрешенных, и `false` — в противном случае. Изначально мы полагаем, что пользователь не входит в число тех, кому доступна эта страница, — принцип презумпции невиновности здесь не уместен.

```
  if (!empty($UserName)) {
```

Здесь проверяется, было ли задано имя пользователя (аргумент `$UserName`), и, если оно было задано, выполняется следующий код. Встроенная функция

empty возвращает true, если переданная ей строка пуста (""), и false в противном случае.

```
$arrUsers = Explode(",", $strUsers);
$arrGroups = Explode(",", $strGroups);
```

Встроенная функция explode разбивает строку, переданную вторым аргументом, на подстроки по символу, который был передан первым аргументом. После этого она создает массив, "складывает" в него полученные подстроки и возвращает этот массив в качестве результата. В нашем случае массивы $arrUsers и $arrGroups будут содержать переданные аргументами $strUsers и $strGroups имена и права пользователей соответственно.

```
if (in_array($UserName, $arrUsers)) {
  $isValid = true;
}
```

Здесь проверяется, содержится ли переданное функции isAuthorized в аргументе $UserName имя пользователя в массиве $arrUsers. Если оно там содержится (пользователь входит в число "избранных"), переменной $isValid присваивается true.

Встроенная функция in_array возвращает true, если значение, переданное первым аргументом, имеется среди элементов массива, переданного вторым аргументом. В противном случае, как это логично предположить, возвращается false.

```
if (in_array($UserGroup, $arrGroups)) {
  $isValid = true;
}
```

Та же самая проверка, но уже прав пользователя.

```
if (($strUsers == "") && false) {
  $isValid = true;
}
```

И еще один ненужный фрагмент кода. Достаточно посмотреть в условие этого выражения — и мы поймем почему.

```
  }
  return $isValid;
}
```

Возвращаем результат — значение переменной $isValid. Все, объявление функции кончилось.

А теперь давайте посмотрим на остальной код PHP. Он не слишком велик.

```
session_start();
$MM_authorizedUsers = "a,m";
$MM_donotCheckaccess = "false";
```

Даем PHP понять, что мы собираемся использовать переменные уровня сессии, и объявляем две переменные. Переменная `$MM_authorizedUsers` содержит список прав доступа, разделенных запятыми; они потом будут переданы функции `isAuthorized`. Вторая переменная (`$MM_donotCheckaccess`) фактически не нужна — больше нигде в коде сценария она не встречается.

Пропускаем код объявления функции `isAuthorized` — мы его уже рассмотрели. Смотрим дальше

```
$MM_restrictGoTo = "Login.php";
```

Сохраняем в переменной `$MM_restrictGoTo` интернет-адрес страницы, на которую будет выполнен переход, если у пользователя нет нужных прав, или если он не выполнил процедуру входа.

```
if (!((isset($_SESSION['MM_Username'])) &&
↳(isAuthorized("",$MM_authorizedUsers,
↳$_SESSION['MM_Username'], $_SESSION['MM_UserGroup']))))) {
```

А здесь выполняется проверка пользователя перед допуском на страницу. Сначала проверяется, вошел ли он на сайт (существует ли переменная уровня сессии `MM_Username`, хранящая его имя). Далее с помощью функции `isAuthorized` выясняется, имеет ли он необходимые для доступа к странице права. Если это не так, то выполняется последующий код.

```
  $MM_qsChar = "?";
  $MM_referrer = $_SERVER['PHP_SELF'];
  if (strpos($MM_restrictGoTo, "?")) $MM_qsChar = "&";
  if (isset($QUERY_STRING) && strlen($QUERY_STRING) > 0)
  $MM_referrer .= "?" . $QUERY_STRING;
  $MM_restrictGoTo = $MM_restrictGoTo. $MM_qsChar . "accesscheck=" .
  ↳urlencode($MM_referrer);
```

Этот код сначала извлекает интернет-адрес текущей страницы, а потом проверяет, были ли ей переданы методом `GET` какие-либо аргументы. Эти аргументы добавляются к интернет-адресу текущей страницы. Потом к интернет-адресу, хранящемуся в переменной `$MM_restrictGoTo`, добавляется аргумент `accesscheck` со значением, равным закодированному адресу текущей страницы с добавленными аргументами, которые были ей переданы методом `GET`. Это нужно для того, чтобы в случае успешного входа на сайт произошел автоматический возврат на эту страницу. Но создаваемый Dreamweaver код PHP, обеспечивающий автоматический возврат, не работает, поэтому весь приведенный ранее фрагмент кода бесполезен.

Встроенная функция `strlen` принимает в качестве аргумента строку и возвращает ее длину в символах. А встроенная функция `urlencode` принимает

аргумент — строку, подготавливает ее для отправки методом GET и возвращает в качестве результата.

```
  header("Location: ". $MM_restrictGoTo);
  exit;
}
```

Этот код PHP обеспечивает переход на страницу, адрес которой хранится в переменной $MM_restrictGoTo, для всех тех пользователей, прав которых не хватает для доступа к текущей странице, или не выполнивших процедуру входа. На этом сценарий заканчивается.

Встроенная функция exit прерывает выполнение сценария. Собственно, она здесь не очень нужна — сценарий и так закончился. Вероятно, Dreamweaver хотел перестраховаться

Создание страницы выхода с сайта

После завершения работы с административными страницами зарегистрированный пользователь должен выполнить процедуру выхода с сайта. При этом все переменные уровня сессии, хранящие его имя и права, будут удалены, и сайт "забудет" об этом пользователе, пока он снова не выполнит процедуру входа.

Для выхода с сайта мы создадим особую страницу по имени Logout.php. Чтобы пользователь смог на нее попасть, мы также поместим указывающую на нее гиперссылку на главную страницу нашего сайта default.htm. Пожалуй, страница Logout.php будет самой простой из всех, что мы создали.

Процесс создания страницы выхода с сайта в Dreamweaver

Запустим Dreamweaver и создадим в нем новую пустую страницу PHP. Зададим для нее название Выход, введем такой же заголовок и какой-либо поясняющий текст. Сохраним эту страницу в папке Admin под именем Logout.php.

Теперь создадим гиперссылку, при щелчке на которой будет выполнена процедура выхода. Создадим новый абзац и введем в нем какое-нибудь слово, например, Выход. Выделим это слово и пока оставим страницу в покое.

Выведем на экран панель **Server Behaviors**, нажмем кнопку со знаком "плюс" и выберем в подменю **User Authentication** появившегося на экране меню пункт **Log Out User**. На экране появится диалоговое окно **Log Out User** (рис. 11.4).

С помощью набора переключателей **Log out when** задается момент, когда будет произведен выход с сайта. Если включен переключатель **Link clicked**, то выход будет выполнен при щелчке на гиперссылке. Если же мы включим

переключатель **Page loads**, то процедура выхода выполнится сразу же при загрузке данной страницы. Поскольку мы хотим дать пользователю возможность передумать и вернуться к работе (мало ли что — может, он случайно зашел на эту страницу), включим переключатель **Link clicked**.

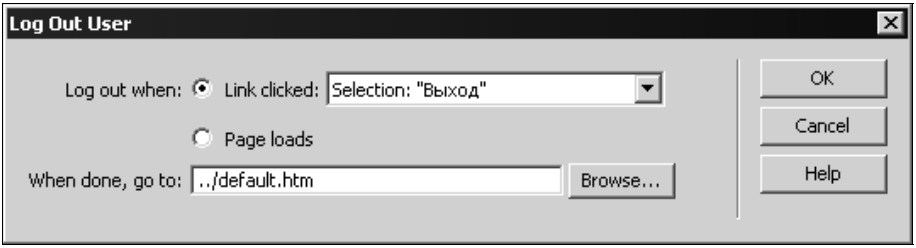


Рис. 11.4. Диалоговое окно **Log Out User**

Сразу же после этого станет доступным раскрывающийся список правее переключателя **Link clicked**. В нем выбирается либо уже существующая на странице гиперссылка, либо пункт **Selection: <*выделенное слово*>**, который предписывает Dreamweaver создать гиперссылку из выделенного фрагмента текста страницы. Выберем этот пункт, поскольку на нашей странице нет подходящих гиперссылок (да и вообще никаких нет).

И напоследок зададим Web-страницу, на которую будет выполнен переход сразу же после успешного выхода. Щелкнем по кнопке **Browse** правее поля ввода **When done, go to**, выберем в появившемся на экране диалоговом окне открытия файла Dreamweaver главную страницу default.htm и нажмем кнопку **OK** этого окна.

Теперь можно нажать кнопку **OK** — и Dreamweaver создаст серверное поведение **Log Out User**. Сохраним готовую страницу Logout.php и закроем ее.

Осталось открыть главную страницу нашего сайта default.htm и создать под заголовком `Администрирование` новый абзац. Введем в него текст `Когда закончите работу, щелкните по этой гиперссылке.`, а слово `этой` превратим в гиперссылку, указывающую на только что созданную страницу Logout.php. Сохраним исправленную главную страницу и закроем ее.

Теперь можно, в принципе, проверить, что мы сделали. Опубликуем все исправленные страницы на локальном Web-сервере, запустим Apache и MySQL, если еще не запустили их, и попробуем в действии нашу систему опознавания "свой-чужой". Как только мы щелкнем по одной из гиперссылок, ведущей на административную страницу, должна появиться страница входа на сайт. Если она появляется, значит, мы все сделали правильно.

Разбор кода PHP, выполняющего выход

Код PHP, выполняющий процедуру выхода, очень прост. Но рассмотреть его все-таки следует. Мы же хотим научиться программировать на PHP?

```
$logoutAction = $_SERVER['PHP_SELF']."?doLogout=true";
if ((isset($_SERVER['QUERY_STRING'])) &&
↳($_SERVER['QUERY_STRING'] != "")){
  $logoutAction .="&". htmlentities($_SERVER['QUERY_STRING']);
}
```

Этот код выясняет интернет-адрес текущей страницы, добавляет к нему аргумент `doLogout` со значением `true`, а также все аргументы, переданные этой странице методом `GET`. Готовый интернет-адрес со всеми этими аргументами помещается в переменную `$logoutAction`; потом он будет подставлен в гиперссылку `Выход`.

А вот и код HTML, создающий гиперссылку `Выход`:

```
<A HREF="<?php echo $logoutAction ?>">Выход</A>
```

Видно, что при щелчке по этой гиперссылке будет снова вызвана текущая страница. Но теперь ей методом `GET` будет передан аргумент `doLogout` — сигнал для серверного поведения **Log Out User**, что нужно выполнить процедуру выхода.

А сейчас вернемся к началу страницы и посмотрим на код PHP, выполняющий собственно выход. Мы рассмотрим его по частям.

```
session_start();
```

И в этом случае не забываем дать PHP знать, что мы будем использовать переменные уровня сессии.

```
if ((isset($_GET['doLogout'])) &&($_GET['doLogout']=="true")){
```

Здесь выполняется проверка, был ли задан переданный методом `GET` аргумент `doLogout` и действительно ли он имеет значение `true`. Если это так, то выполняется следующий код:

```
  session_unregister('MM_Username');
  session_unregister('MM_UserGroup');
```

Встроенная функция `session_unregister` удаляет переменную уровня сессии, имя которой передано ей в качестве аргумента в строковом виде. Здесь мы удаляем переменные уровня сессии `MM_Username` и `MM_UserGroup`, хранящие, соответственно, имя пользователя и его права.

```
  $logoutGoTo = "../default.htm";
  if ($logoutGoTo) {
    header("Location: $logoutGoTo");
```

```
    exit;
  }
}
```

Очередное чудо от Dreamweaver. Нет, чтобы написать проще:

```
$logoutGoTo = "../default.htm";
header("Location: $logoutGoTo");
```

Ну да ладно! Потом, когда мы приобретем опыт, исправим все ошибки Dreamweaver (точнее, ошибки его разработчиков).

Создание административных страниц для управления пользователями

Теперь нам нужно создать административные Web-страницы для управления зарегистрированными пользователями: страницу списка пользователей, которую мы назовем Users_admin.php, и страницы для добавления (User_add.php), правки (User_edit.php) и удаления (User_delete.php) пользователя. Все эти страницы будут храниться в папке Admin нашего сайта.

Как создавать страницы для вывода набора записей и правки данных, мы уже знаем. Подобная информация была во всех подробностях представлена в *главах 8* и *9*. Так что мы сможем сделать это без посторонней подсказки.

Чтобы попасть на страницу списка пользователей, добавим на главную страницу default.htm новый абзац. Вставим его между двумя уже существующими абзацами под заголовком `Администрирование`, напишем в нем текст наподобие `Чтобы попасть на список пользователей, щелкните эту гиперссылку.` и превратим слово `эту` в гиперссылку, указывающую на страницу списка пользователей Users.php.

И еще одно замечание. Поскольку доступ к административным страницам у нас ограничен, нам нужно будет обязательно создать на всех страницах управления пользователями серверное поведение **Restrict Access To Page**. Только в этом случае мы должны будем выбрать в списке **Select level(s)** диалогового окна **Restrict Access To Page** (см. рис. 11.2) только пункт **a**. Мы же условились еще в начале этой главы, что доступ к страницам управления пользователями будут иметь только администраторы, но никак не ведущие.

Напоследок опубликуем созданные страницы управления пользователями на Web-сервере и откроем наш сайт в Web-обозревателе. Попробуем открыть страницу управления пользователями, выполним вход на сайт под именем `Admin` и добавим еще одного пользователя. Дадим ему имя `Manager`, такой же пароль и права ведущего, введя в поле ввода **Права** страницы User_add.php букву `m`. После этого выполним процедуру выхода и остановимся на главной странице.

Разграничение доступа к фрагментам Web-страниц

Всем хороша наша система разграничения доступа к административным страницам. Вот только на главной странице назойливо "маячат" гиперссылки, указывающие на административные страницы, причем видны они всегда, даже если мы еще не выполнили вход на сайт. Давайте сделаем так, чтобы они показывались только после успешного входа.

Dreamweaver неплохо справляется с разграничением доступа к целым Web-страницам. Но если мы хотим скрыть от непрошеных гостей не страницу целиком, а какой-то ее фрагмент, нам придется самим писать нужные сценарии PHP. Но это не так уж и сложно, ведь мы уже знаем, каким образом реализуется разграничение доступа в PHP, по крайней мере, как это делает Dreamweaver.

Откроем страницу Categories_admin.php и переключимся в режим отображения кода HTML. Найдем сценарий, "отвечающий" за проверку имени и прав пользователя, благо уже знаем, как он выглядит. Вот он:

```
$MM_restrictGoTo = "Login.php";
if (!((isset($_SESSION['MM_Username'])) &&
↳(isAuthorized("",$MM_authorizedUsers,
↳$_SESSION['MM_Username'], $_SESSION['MM_UserGroup']))))) {
  . . .
  header("Location: ". $MM_restrictGoTo);
  exit;
}
```

Здесь мы опустили лишний код, созданный Dreamweaver, и оставили только реально работающий. Он содержит условное выражение, проверяющее, было ли задано имя пользователя (то есть выполнил ли он вход) и достаточны ли его права. Если нет, то выполняется переход на другую страницу, в нашем случае — на страницу входа на сайт Login.php.

Теперь посмотрим на HTML-код страницы default.htm, выводящий на экран все три абзаца под заголовком `Администрирование`.

```
<H2>Администрирование</H2>
<P>Чтобы попасть на административную страницу списка категорий файлов,
↳щелкните по <A HREF="Admin/Categories_admin.php?file=1">этой
↳гиперссылке</A>. А <A HREF="Admin/Categories_admin.php?file=0">эта
↳гиперссылка</A> ведет на список категорий статей.</P>
<P>Чтобы попасть на список пользователей, щелкните
↳<A HREF="Admin/Users.php">эту</A> гиперссылку.</P>
```

```
<P>Когда закончите работу, щелкните по
↳<A HREF="Admin/Logout.php">этой</A> гиперссылке.</P>
```

Нам необходимо поместить этот код внутрь условного выражения, приведенного ранее. И не забыть еще убрать оператор логического НЕ `!` из условия этого выражения, иначе получим результат, прямо противоположный тому, какой нам нужен.

```
<?php if (((isset($_SESSION['MM_Username'])) &&
↳(isAuthorized("","a,m", $_SESSION['MM_Username'],
↳$_SESSION['MM_UserGroup']))))) { ?>
<H2>Администрирование</H2>
<P>Чтобы попасть на административную страницу списка категорий файлов,
↳щелкните по <A HREF="Admin/Categories_admin.php?file=1">этой
↳гиперссылке</A>. А <A HREF="Admin/Categories_admin.php?file=0">эта
↳гиперссылка</A> ведет на список категорий статей.</P>
<P>Чтобы попасть на список пользователей, щелкните
↳<A HREF="Admin/Users.php">эту</A> гиперссылку.</P>
<P>Когда закончите работу, щелкните по
↳<A HREF="Admin/Logout.php">этой</A> гиперссылке.</P>
<?php } ?>
```

Вот и все. Можно вставить этот код прямо в страницу default.htm. Но работать он не будет. И вот почему.

Во-первых, в условном выражении используется функция `isAuthorized`. Эта функция не встроенная, а значит, должна быть объявлена где-нибудь в начале файла default.htm. Давайте позаимствуем код ее объявления из все той же страницы Categories_admin.php и вставим в самое начало страницы default.htm, перед всем HTML-кодом.

Во-вторых, чтобы созданные нами в странице default.htm сценарии PHP были нормально выполнены, нам придется переименовать ее, а именно изменить ее расширение с htm на php. Для этого выведем на экран панель **Files**, выделим в ее списке файл default.htm, нажмем клавишу <F2>, изменим расширение файла и напоследок нажмем клавишу <Enter>. После этого на экране появится диалоговое окно **Update Files** (см. рис. 4.6); нажмем кнопку **Update**, чтобы Dreamweaver изменил гиперссылки на других страницах нашего сайта, указывающих на default.htm.

В-третьих, перед объявлением функции нам будет нужно подставить вызов функции `session_start`. Мы ведь будем обращаться к переменным уровня сессии, значит, обработчик PHP должен быть готов к этому.

Вот теперь можно проверять исправленную главную страницу в действии. Но давайте немного подождем и подумаем, ничего ли мы не упустили из виду.

Административные страницы нашего сайта должны быть доступны только зарегистрированным пользователям. Это мы сделали. Но страницы управления пользователями, как мы условились, должны быть доступны только администраторам. А у нас гиперссылка, указывающая на страницу Users_admin.php, видна всем — и администраторам, и ведущим. Будет логично скрыть ее от ведущих.

Как это сделать? Очень просто. Мысленно разобьем фрагмент текста страницы default.htm под заголовком `Администрирование` на три части.

- Первая часть будет включать сам заголовок и первый абзац с гиперссылками, указывающими на списки категорий. Она должна быть доступна всем зарегистрированным пользователям.
- Вторая часть содержит абзац с гиперссылкой, указывающей на страницу списка пользователей, и должна быть доступна только администраторам.
- На долю третьей части останется последний абзац и гиперссылка на страницу выхода. Эта часть, опять же, должна быть доступна всем зарегистрированным пользователям.

Далее поместим каждую из этих частей в отдельное условное выражение PHP, подобное тому, что было приведено ранее. Все эти выражения (помимо содержимого, разумеется) будут отличаться друг от друга только одним — значением второго аргумента функции `isAuthorized`. В первом и третьем выражении он будет равен `a,m`, а во втором — `a`.

Исходя из этого, давайте исправим написанный ранее код.

```
<?php if (((isset($_SESSION['MM_Username'])) &&
↳(isAuthorized("","a,m", $_SESSION['MM_Username'],
↳$_SESSION['MM_UserGroup']))))) { ?>
<H2>Администрирование</H2>
<P>Чтобы попасть на административную страницу списка категорий файлов,
↳щелкните по <A HREF="Admin/Categories_admin.php?file=1">этой
↳гиперссылке</A>. А <A HREF="Admin/Categories_admin.php?file=0">эта
↳гиперссылка</A> ведет на список категорий статей.</P>
<?php } ?>
<?php if (((isset($_SESSION['MM_Username'])) &&
↳(isAuthorized("","a", $_SESSION['MM_Username'],
↳$_SESSION['MM_UserGroup']))))) { ?>
<P>Чтобы попасть на список пользователей, щелкните
↳<A HREF="Admin/Users.php">эту</A> гиперссылку.</P>
<?php } ?>
<?php if (((isset($_SESSION['MM_Username'])) &&
↳(isAuthorized("","a,m", $_SESSION['MM_Username'],
```

```
↳$_SESSION['MM_UserGroup']))))) { ?>
<P>Когда закончите работу, щелкните по
↳<A HREF="Admin/Logout.php">этой</A> гиперссылке.</P>
<?php } ?>
```

Все хорошо, только вот код получился слишком громоздкий и, как говорят профессиональные программисты, неоптимальный. Посмотрим сами — в двух условных выражениях используется одно и то же условие, которое вычисляется дважды. Давайте выделим это условие в отдельное выражение и присвоим его значение переменной, скажем, `$isAorM`. А уже в условных выражениях используем значение этой переменной.

Вот так будет выглядеть исправленный код после проведенной нами *оптимизации*:

```
<?php $isAorM = (((isset($_SESSION['MM_Username'])) &&
↳(isAuthorized("","a,m", $_SESSION['MM_Username'],
↳$_SESSION['MM_UserGroup']))))); ?>
<?php if ($isAorM) { ?>
<H2>Администрирование</H2>
<P>Чтобы попасть на административную страницу списка категорий файлов,
↳щелкните по <A HREF="Admin/Categories_admin.php?file=1">этой
↳гиперссылке</A>. А <A HREF="Admin/Categories_admin.php?file=0">эта
↳гиперссылка</A> ведет на список категорий статей.</P>
<?php } ?>
<?php if (((isset($_SESSION['MM_Username'])) &&
↳(isAuthorized("","a", $_SESSION['MM_Username'],
↳$_SESSION['MM_UserGroup']))))) { ?>
<P>Чтобы попасть на список пользователей, щелкните
↳<A HREF="Admin/Users.php">эту</A> гиперссылку.</P>
<?php } ?>
<?php if ($isAorM) { ?>
<P>Когда закончите работу, щелкните по
↳<A HREF="Admin/Logout.php">этой</A> гиперссылке.</P>
<?php } ?>
```

Этот код можно вставить в главную страницу, и он будет работать.

Вот и все. Можно проверять созданные нами страницы в действии. Только не забудем удалить из корневой папки Web-сервера старую главную страницу default.htm, иначе Web-сервер может подсунуть нам именно ее, и мы будем долго искать, в каком месте кода мы допустили ошибку.

Но как же теперь попасть на Web-страницу входа на сайт Login.php? Очень просто — по прямой ссылке вида

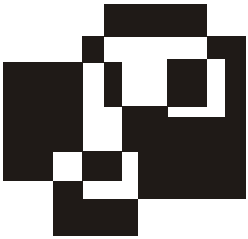
http://localhost:8080/Admin/Login.php

Всем администраторам и ведущим сайта мы должны будем дать эту ссылку, иначе они не смогут войти на сайт. Таким образом мы дополнительно защитим административные страницы от посягательств посторонних — никто, кроме особо приближенных к сайту лиц, не сможет добраться до страницы входа.

Что дальше?

Ну что ж, разграничение доступа к страницам нашего сайта мы реализовали. Теперь никто, кроме администраторов и ведущих, не сможет добраться до административных страниц и испортить наши данные.

Следующий шаг усиления безопасности — поддержание ссылочной целостности данных. Этим мы займемся в следующей главе.



Глава 12

Поддержание ссылочной целостности данных

Не только разграничение доступа к страницам сайта относится к безопасности. Поддержание данных в порядке — это тоже безопасность. В самом деле, нарушится строгая структура связей между таблицами — и часть записей станет недоступной. Придется нам запускать клиент данных (phpMyAdmin, MySQL Control Center и т. п.) и вручную править "потерянные" записи, чтобы вернуть их "в строй".

Вспомним, что есть ссылочная целостность и зачем ее нужно поддерживать. Предположим, что у нас есть две связанных таблицы — первичная и вторичная. (А у нас и есть две связанных таблицы — `categories` и `items`.) Согласно правилам построения реляционных баз данных, каждая запись вторичной таблицы обязательно должна быть связана только с одной записью первичной таблицы. Профессиональные разработчики баз данных говорят при этом, что соблюдается ссылочная целостность; все связи однозначны и не нарушены. Это значит, что мы можем взять любую запись таблицы `categories` (категорию) и извлечь из таблицы `items` все связанные с ней записи (все соответствующие этой категории статьи и файлы).

Теперь предположим, что эта связь разорвана (например, одну из записей первичной таблицы удалили). Тогда все записи вторичной таблицы, связанные с удаленной записью первичной таблицы, становятся недоступными. Все они потеряны для пользователя! Попробуем, например, удалить запись таблицы `categories`, соответствующую категории `Интернет`, — и все статьи (файлы), относящиеся к этой категории, будут потеряны. Ссылочная целостность нарушена.

Внимание

Все-таки такой эксперимент лучше не проводить...

Подавляющее большинство серверов данных, присутствующих на рынке, сами следят за ссылочной целостностью и не допускают ее нарушений. Если

незадачливый пользователь попытается удалить запись первичной таблицы, на которую ссылаются записи вторичной таблицы, сервер либо выдаст сообщение об ошибке и не станет удалять эту запись, либо удалит также и все связанные записи вторичной таблицы. (Это задается самим разработчиком базы данных при создании связи между таблицами. Удобнейшая, кстати, штука!)

Но MySQL перекладывает заботу о ссылочной целостности на плечи программиста, который будет разрабатывать клиенты данных для него. Нам самим придется думать о поддержании ссылочной целостности и не допускать ее нарушений. Чем мы сейчас и займемся.

В этой главе мы будем, в основном, работать со страницей удаления категории Category_delete.php. Именно она выполняет удаление записи первичной таблицы, чреватое потерей межтабличной связи. Так что мы можем сразу же открыть ее в Dreamweaver.

Простейший способ поддержания ссылочной целостности

Вообще, существует два способа сохранения ссылочной целостности — простой и сложный. Мы рассмотрим их по очереди. И начнем, конечно, с простого.

А простой способ заключается в том, чтобы прямо перед удалением записи первичной таблицы выполнить проверку, существуют ли во вторичной таблице записи, ссылающиеся на удаляемую. Если они существуют, удаление записи не выполняется, и пользователю выводится соответствующее предупреждение. Если же на удаляемую запись первичной таблицы не ссылается ни одна запись вторичной таблицы, — путь свободен.

Простым этот способ называется, в основном, потому, что его можно реализовать исключительно встроенными средствами Dreamweaver. Реализация его выполняется в три шага.

1. Создаем набор записей, извлекающий из вторичной таблицы записи, ссылающиеся на удаляемую.
2. Создаем необязательную область, содержащую форму с кнопкой **Удалить** и выводящуюся на экран, если созданный на первом шаге набор записей пуст.
3. Создаем вторую необязательную область, но содержащую уже текст предупреждения о невозможности удаления записи. Понятно, что выводиться на экран она должна, если созданный на первом шаге набор содержит записи (не пуст).

Что ж, если все это так просто, давайте воплотим этот способ в программном коде. Dreamweaver у нас уже запущен, страница Category_delete.php открыта, значит, можно приступать к работе прямо сейчас.

Но не будем торопиться. Давайте сначала скопируем страницу Category_delete.php в какую-нибудь другую папку. Мы ведь собираемся реализовать поддержание ссылочной целостности двумя разными способами, так что нам понадобится та же страница в "нетронутом" виде.

Сначала нам нужно создать набор записей вторичной таблицы, ссылающихся на удаляемую запись первичной таблицы. Мы уже знаем, как это делается, из *главы 8* этой книги. Отберем в набор те записи из таблицы `items`, значение поля `catid` которых равно значению аргумента `id`, переданного этой странице методом `GET`. (Данный аргумент также используется для поиска удаляемой записи первичной таблицы `categories`.) Пусть в набор записей будет включено только поле `id` — нам все равно нужно только узнать, есть ли записи в наборе, а никаких значений из него мы использовать не будем. Назовем этот набор записей, скажем, `Items`.

Создав набор записей, выделим всю форму и превратим ее в необязательную область, выводящуюся на экран, если набор записей `Items` пуст. Как это сделать, было также описано в *главе 8*.

Теперь нам нужно ухитриться поместить сразу после формы абзац с текстом вида `В этой категории есть файлы или статьи. Удалите сначала их.`. Проще всего это сделать, переключившись в режим отображения HTML-кода и введя нужный код вручную.

Осталось выделить только что созданный абзац и превратить его в необязательную область, выводящуюся на экран, если набор записей `Items` не пуст. Сделаем это — и можно испытать готовую страницу Category_delete.php в действии.

Вот таков самый простой и очевидный способ поддержать ссылочную целостность данных нашего сайта. Как говорится, дешево и сердито. Но — только до поры до времени...

Второй — более сложный — способ поддержания ссылочной целостности

Второй способ поддерживать ссылочную целостность не очень уж сложен. Просто для его реализации нам придется узнать кое-что новое о MySQL. И, вдобавок, Dreamweaver в этом случае нам не особый помощник — он не имеет встроенных средств, позволяющих сделать то, что нам нужно, несколькими щелчками мыши. Но нам уже не привыкать писать PHP-код вручную, поэтому вперед!

Недостаток первого способа поддержания ссылочной целостности и попытка его устранить

Да, но почему потребовалось выдумывать еще один способ поддерживать ссылочную целостность? Неужели PHP-программистам больше нечего делать, кроме как изобретать все более и более хитроумные приемы программирования? Ведь нам вполне хватает и первого способа. Он прекрасно работает, сохраняя структуру связей между таблицами в целости. Если мы попытаемся удалить категорию, в которой содержатся статьи (файлы), сценарий PHP, созданный Dreamweaver, не даст нам этого сделать. Но все это работает, если мы будем заниматься администрированием сайта в полном одиночестве. Как только на сайт войдет еще один администратор или ведущий, первый способ начнет давать сбои.

Давайте рассмотрим ситуацию, когда двое ведущих, `А` и `Б`, вошли на сайт и начинают править списки категорий и статей (файлов) соответственно. Причем работают они одновременно, в один и тот же момент.

1. Ведущий `А` собирается удалить категорию `Интернет`, не содержащую статей (файлов). Для этого он щелкнул гиперссылку `Удалить` на странице списка категорий Categories_admin.php и перешел на страницу удаления категории Category_delete.php. Поскольку категория `Интернет` не содержит статей (файлов), ее удаление разрешено.
2. Ведущий `Б` вошел в список статей (файлов), относящихся к категории `Интернет`, и добавил туда статью или файл.
3. Ведущий `А` наконец-то щелкнул кнопку **Удалить** на странице Category_delete.php. Категория `Интернет` удалена.

Что мы получим в результате? А получим мы все то же нарушение ссылочной целостности. Пока ведущий `А` молчаливо созерцал страницу Category_delete.php, размышляя о превратностях жизни, ведущий `Б` поместил в категорию `Интернет` новую статью (файл). Когда ведущий `А` все-таки решился сделать непоправимое и удалил запись таблицы `categories`, соответствующую категории `Интернет`, в таблице `items` осталась связанная с ней запись — добавленная статья (файл). И сценарий PHP, служащий для поддержания ссылочной целостности, что мы создали ранее, не помог.

Да, разделение труда приносит не только облегчение, но и новые проблемы. Так что нам делать в этом случае? Как спасти данные от краха?

Сразу напрашивается вот такое решение. А что если проверять существование в таблице `items` записей, связанных с удаляемой записью таблицы `categories`, перед самым ее удалением? То есть немного усовершенствовать первый способ, сделать его немного "оперативнее".

Давайте еще раз посмотрим на фрагмент сценария PHP, соответствующего серверному поведению **Delete Record** Dreamweaver. (Полностью этот сценарий был рассмотрен в *главе 9*.)

```
$deleteSQL = sprintf("DELETE FROM categories WHERE id=%s",
↳GetSQLValueString($_POST['id'], "int"));
mysql_select_db($database_Site, $Site);
$Result1 = mysql_query($deleteSQL, $Site) or die(mysql_error());
```

Здесь аргумент `id`, переданный методом `POST`, задает значение поля `id` записи таблицы `categories`, которую нужно удалить. Переменная `$Site` хранит идентификатор соединения с сервером, а `$database_Site` — имя базы данных.

Теперь изменим его так, чтобы он проверял перед удалением записи таблицы `categories`, есть ли ссылающиеся на нее записи в таблице `items`. И разберем новый код по частям.

```
$deleteSQL = sprintf("DELETE FROM categories WHERE id=%s",
↳GetSQLValueString($_POST['id'], "int"));
```

Это выражение целиком взято из первоначального фрагмента кода.

```
$CheckSQL = sprintf("SELECT id FROM items WHERE catid=%s",
↳GetSQLValueString($_POST['id'], "int"));
mysql_select_db($database_Site, $Site);
$Check = mysql_query($CheckSQL, $Site) or die(mysql_error());
```

Подготавливаем код SQL-запроса, выбирающего из таблицы `items` связанные записи, и выполняем его.

```
if (!($row_Check = mysql_fetch_assoc($Check))) {
  $Result1 = mysql_query($deleteSQL, $Site) or die(mysql_error());
}
mysql_free_result($Check);
```

Проверяем, есть ли в получившемся после выполнения этого запроса наборе записи, и, если их нет, выполняем запрос удаления. После чего обязательно удаляем из памяти серверного компьютера проверочный набор записей `$Check` — нам он больше не нужен, а компьютерная память может пригодиться для чего-нибудь более важного.

Вот такой не очень сложный код PHP. Мы можем взять сохраненную ранее копию первоначальной страницы Category_delete.php и вставить этот код в нужное место. А он будет работать.

Но возникает другой вопрос: насколько оперативно обновленный сценарий будет выполнять проверку на существование связанных записей? К сожалению, существует очень большая вероятность того, что ведущий `B` успеет

добавить связанную запись в момент между выполнением проверяющего запроса и запроса на удаление. А именно —

```
$Check = mysql_query($CheckSQL, $Site) or die(mysql_error());
// Вот здесь!!!
if (!($row_Check = mysql_fetch_assoc($Check))) {
  $Result1 = mysql_query($deleteSQL, $Site) or die(mysql_error());
}
```

Набор записей, созданный в результате выполнения запроса SQL `$CheckSQL`, пуст, и запрос на удаление записи будет выполнен. Но таблица `items` на самом деле содержит связанную запись, добавленную ведущим `B`! Так что мы опять получаем нарушение ссылочной целостности.

Вот если бы как-нибудь запретить другим администраторам и ведущим добавление, правку и удаление записей в таблице `items`. Не навсегда, разумеется, а только на небольшое время, пока выполняются запросы `$CheckSQL` и удаления записи. Сколько бы проблем мы могли решить!..

Блокировка таблиц MySQL и ее использование

Так давайте возьмем и запретим! А потом, когда все сделаем, разрешим.

MySQL, конечно, оригинальный сервер данных — он не предоставляет возможности задать связи между таблицами и не следит за ссылочной целостностью. Но зато он предоставляет нам замечательную возможность временно закрыть к определенной таблице доступ по записи или чтению для всех остальных пользователей. Как говорят администраторы MySQL и разработчики решений на его основе, наложить на таблицу *блокировку*.

Мы можем наложить блокировку на чтение или на запись. *Блокировка на запись* запрещает другим пользователям записывать данные в заблокированную таблицу. (Под записью подразумевается добавление, изменение и удаление записей. Да-да, и удаление, поскольку при этом в таблицу тоже записываются какие-то данные.) Записывать туда что-либо будем иметь право только мы.

Блокировка на чтение — самая жесткая. Если мы ее наложим, то никто, кроме нас, не сможет даже прочитать данные из таблицы, не то что записать (это значит, что блокировка на чтение блокирует таблицу также и на запись). Остальным пользователям придется ждать, пока мы не снимем блокировку.

Для наложения и снятия блокировки используются особые запросы SQL. Сейчас мы их рассмотрим.

Запрос блокировки таблицы создается с помощью ключевого слова `LOCK TABLES` языка SQL и имеет такой формат:

```
LOCK TABLES <имя таблицы 1> READ|WRITE, <имя таблицы 2> READ|WRITE,…
```

Здесь ключевое слово `READ`, стоящее после имени блокируемой таблицы, заставляет MySQL заблокировать ее на чтение, а ключевое слово `WRITE` — на запись. Видно, что с помощью одного запроса можно блокировать сразу несколько таблиц, причем по-разному: одни таблицы — на чтение, другие — на запись.

Чтобы снять блокировку, мы будем должны отправить MySQL вот такой запрос:

```
UNLOCK TABLES
```

Он снимает блокировки сразу со всех блокированных нами таблиц. Снять блокировку только с одной таблицы, оставив другие блокированными, нельзя.

Блокировки с таблиц также снимаются при подаче нового запроса `LOCK TABLES` или при разрыве соединения с MySQL.

Внимание

Продолжительность блокирования таблиц должна быть минимальной. Необходимо помнить, что база данных нужна не только нам, но и другим пользователям.

Замечание

При выполнении запросов на добавление, изменение и удаление записи MySQL сам блокирует таблицу на чтение для того, чтобы никто не смог прочитать данные из таблицы прежде, чем выполнится этот запрос.

Реализация второго способа поддержания ссылочной целостности

Итак, у нас есть возможность временно запретить запись в таблицу. А значит, мы можем использовать для проверки, если ли в таблице `items` связанные записи, приведенный ранее код PHP. Разумеется, дополненный выражениями, выполняющими блокировку и разблокировку таблиц.

Давайте разберем исправленный код по частям.

```
$deleteSQL = sprintf("DELETE FROM categories WHERE id=%s",
↳GetSQLValueString($_POST['id'], "int"));
$CheckSQL = sprintf("SELECT id FROM items WHERE catid=%s",
↳GetSQLValueString($_POST['id'], "int"));
mysql_select_db($database_Site, $Site);
```

Эти три выражения целиком взяты из первоначального кода. Они подготавливают запросы на проверку существования в таблице `items` связанных

записей и удаление записи таблицы `categories` и выполняют подключение к базе данных `site`.

```
$R = mysql_query("LOCK TABLES items WRITE, categories WRITE", $Site) or
↳die(mysql_error());
```

Выполняем запрос блокировки таблиц `categories` и `items` на запись. Лучше заблокировать обе таблицы — так будет надежнее.

```
$Check = mysql_query($CheckSQL, $Site) or die(mysql_error());
if (!($row_Check = mysql_fetch_assoc($Check))) {
  $Result1 = mysql_query($deleteSQL, $Site) or die(mysql_error());
}
mysql_free_result($Check);
```

Опять фрагмент изначального кода. Он проверяет, есть ли записи в наборе `$Check`, и, если этот набор пуст, выполняет удаление записи таблицы `categories`. И, разумеется, удаляет из памяти серверного компьютера уже ненужный набор записей `$Check`.

```
$R = mysql_query("UNLOCK TABLES", $Site) or die(mysql_error());
```

Не забываем снять блокировку с блокированных ранее таблиц, иначе никто, кроме нас, не сможет в них ничего записать.

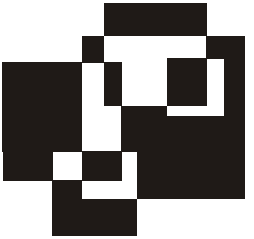
Мы можем вставить этот код в сохраненную ранее копию первоначальной страницы Category_delete.php. Но будет лучше, если мы поместим его в измененную страницу Category_delete.php, уже содержащую код, реализующий первый способ сохранения ссылочной целостности. Так сказать, сделаем двойную проверку.

Последнее, что нам останется сделать, — это испытать уже полностью готовую страницу Category_delete.php в действии. Сначала попробуем удалить категорию, содержащую статьи (файлы); если у нас не получилось это сделать, значит, написанный нами код PHP работает правильно. После этого создадим новую, "пустую" (не содержащую ни статей, ни файлов) категорию и попробуем удалить ее. Вот "пустая" категория будет удалена успешно.

Что дальше?

Собственно, наш сайт полностью готов. Мы создали все необходимые страницы — пользовательские и административные — и решили все проблемы с безопасностью. Это значит, что мы смело можем публиковать свой сайт в Сети.

А когда мы проверим его в реальной работе, обзаведемся преданными поклонниками и деятельными помощниками — администраторами и ведущими, — вернемся к нашему сайту. Мы добавим на страницы кое-какие "вкусности", которыми хвастаются другие сайты: список рассылки, управление файлами через Web-интерфейс и пр. Как раз этим "вкусностям" посвящена четвертая, заключительная, часть этой книги.



Часть IV

Наносим последние штрихи

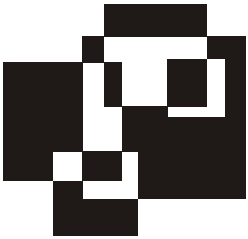
Глава 13. Вывод сообщений об ошибках

Глава 14. Хранение данных на стороне клиента

Глава 15. Управление файлами через Web-интерфейс

Глава 16. Организация почтовой рассылки

Что ж, наш сайт практически готов. Осталось нанести последние штрихи, добавить несколько полезных мелочей, которые дополнят наш сайт, возможно, даже выделят его из сонма других Web-творений. Да, они не обязательны, но давайте все-таки их сделаем.



Глава 13

Вывод сообщений об ошибках

Как и было обещано ранее, эта часть будет посвящена различным "вкусностям", которыми мы "приправим" наш уже полностью готовый сайт. Конечно, они не являются жизненно необходимыми для его нормальной работы, но пренебрегать ими тоже не годится. Как толика перца делает блюдо вкуснее, так и несколько полезных мелочей сделают наш сайт лучше.

Итак, что же это за "вкусности"? Давайте их перечислим.

- Нормальные, понятные для обычного посетителя, не искушенного в интернет-технологиях, сообщения об ошибках: отсутствие запрашиваемой страницы, ошибка подключения к серверу данных, ошибка обработки запроса SQL и т. д.
- Возможности посетителя настроить Web-страницы по своему вкусу. Например, мы можем предоставить посетителю возможность сортировать списки статей и файлов не только по дате добавления, но и по автору, названию и пр. При этом можно сделать так, чтобы сведения о порядке сортировки, заданном посетителем, сохранялись на его компьютере.
- Управление файлами (Web-страницами, изображениями, программами, архивами и пр.), опубликованными на нашем сайте, через специально созданные для этого Web-страницы. Одним словом, Web-интерфейс для управления файлами.
- Почтовая рассылка, сообщающая всем желающим об обновлениях на нашем сайте (новые статьи и файлы, исправленные гиперссылки, обновление дизайна всего сайта и пр.).

Вот, по крайней мере, такими "украшениями" щеголяют многие Web-сайты, созданные профессиональными программистами. Мы тоже научимся их делать, благо PHP предоставляет нам для этого все возможности.

А теперь — маленькая ложка дегтя в обещанную автором большую бочку меда (этакая приправа особого толка). Дело в том, что Dreamweaver в этом случае нам не помощник. Он не имеет встроенных инструментов для создания

серверных поведений, выполняющих, например, рассылку новостей по почте. Так что нам придется писать код PHP самим. Но ведь мы уже довольно опытны в PHP-программировании, так что нас это не страшит.

Начнем мы с того, что заставим наши Web-страницы показывать посетителю сайта нормальные, удобочитаемые сообщения об ошибках. В конце концов, Web-сервер, на котором мы опубликуем наш сайт, тоже может перестать работать, так пусть посетители хотя бы знают, что с ним случилось.

Страница, сообщающая об ошибке 404

Об ошибке 404 мы знаем из *главы 4*. Сообщение об этой ошибке Web-сервер отправляет Web-обозревателю, если посетитель пытается открыть несуществующую страницу или несуществующий сайт. При этом Web-обозреватель выводит в свое окно текст, предупреждающий посетителя, что такой страницы или такого сайта нет, и советующий проверить правильность набранного им интернет-адреса или обратиться к администратору сайта (если отсутствует страница, но сам сайт существует). В разных программах Web-обозревателей этот текст различается, но смысл все равно несет одинаковый.

Вообще, в стандарте протокола HTTP описано довольно много ошибок, которые могут произойти при работе в Интернете. Каждая такая ошибка имеет свой код — особый уникальный номер, который Web-сервер и отправляет Web-обозревателю. Например, ошибка 404 имеет код 404, от которого, собственно, и получила свое название.

Вообще, способов оповестить Web-обозреватель (а через него — и посетителя) о возникновении какой-либо ошибки существует три. Давайте их рассмотрим.

- Web-сервер отправляет Web-обозревателю только код возникшей ошибки, а Web-обозреватель сам интерпретирует его. Вся эта операция выполняется автоматически самим Web-сервером, когда он не может найти запрошенную страницу. (Самый простой способ.)
- Вместо кода ошибки Web-сервер отправляет Web-обозревателю особую Web-страницу с сообщением об ошибке и предложением перейти на главную страницу сайта, отправить письмо администратору или выполнить поиск по сайту. Эта страница формируется самим Web-сервером (как говорят профессиональные программисты, "зашита" в нем).
- Web-дизайнер сам создает страницу с сообщением об ошибке 404 и записывает ее имя в настройках Web-сервера. После этого при возникновении ошибки Web-сервер отправляет Web-обозревателю именно эту страницу. (Этот способ предоставляет Web-дизайнеру больше свободы.)

Последний способ часто применяется на сайтах, публикуемых на бесплатных Web-серверах. Давайте и мы его используем.

Запустим Dreamweaver и создадим в нем простую статичную Web-страницу. Дадим ее название `Ошибка`, такой же заголовок и введем текст вида `Запрошенная вами страница отсутствует.`. Поместим на этой странице гиперссылку для перехода на главную страницу сайта и адрес электронной почты администратора. Также можно поместить на эту страницу форму поиска, "позаимствовав" ее с главной страницы.

Сохраним готовую страницу под именем Error.htm в корневой папке сайта. И сразу же опубликуем ее на локальном Web-сервере, чтобы потом не забыть это сделать.

Теперь нам как-то нужно дать Web-серверу понять, что созданная нами страница должна отправляться посетителю при возникновении ошибки с кодом 404. Для этого мы используем особый файл локальной конфигурации Apache, имеющий имя .htaccess (точка в начале имени обязательна!).

Файл локальной конфигурации .htaccess — это обычный текстовый файл, задающий некоторые параметры Web-сервера, применимые только для папки, в которой он находится. С его помощью можно, например, ограничить доступ к папке средствами самого Web-сервера, задав в этом файле имена пользователей, или задать для сайта Web-страницу с сообщением об ошибке 404 (что нам и нужно). Файл локальной конфигурации просто помещается в папку, параметры которой он задает, а Web-сервер автоматически прочитывает его и применяет заданные параметры для этой папки.

Давайте запустим Блокнот или аналогичный текстовый редактор и напишем в нем такую строку:

```
ErrorDocument 404 /Error.htm
```

Далее сохраним готовый файл под именем .htaccess в корневой папке удаленной копии сайта — той, что опубликована на Web-сервере. (По умолчанию это папка htdocs, находящаяся в папке, где установлен Apache.) Только когда будем набирать имя файла в поле ввода стандартного диалогового окна сохранения файла Windows, заключим его в кавычки, иначе Блокнот его не сохранит.

Введенная нами в файле .htaccess строка предписывает Web-серверу искать страницу с сообщением об ошибке 404 в файле Error.htm, хранящемся в корневой папке сайта. Синтаксис ее "прозрачен": сначала — ключевое слово `ErrorDocument`, далее, через пробел, — код ошибки (404), потом, снова через пробел, — имя файла страницы с сообщением об этой ошибке.

Теперь настало время проверить, как будет вести себя Web-сервер после всех этих манипуляций. Запустим его, откроем Web-обозреватель и попробуем набрать в строке адреса интернет-адрес заведомо не существующей страницы. Например, вот такой:

http://localhost:8080/abcdef.htm

Если мы все сделали правильно, то Web-сервер выдаст нам страницу Error.htm. Если же нет, будет нужно проверить, правильно ли мы его настроили. (Процесс установки и настройки Web-сервера Apache описан в *приложении 1*.)

Внимание

При публикации готового сайта на удаленном Web-сервере нужно быть готовым к тому, что администратор этого сервера отключил обработку файлов .htaccess. Выяснить, так это или нет, можно у самого администратора или на специальной Web-странице сведений для клиентов.

Сообщения об ошибках в сценариях PHP

Сделав свою страницу, сообщающую об ошибке 404, давайте займемся сообщениями об ошибках в сценариях PHP. А именно сделаем так, чтобы наши серверные страницы сообщали посетителю сайта об ошибках работы с базой данных: ошибке подключения к серверу, ошибке выполнения запроса и пр.

Вообще-то, сам обработчик PHP умеет выявлять эти ошибки и сообщать о них посетителю. Правда, эти сообщения предназначены, скорее, не для обычного посетителя, зашедшего на сайт за нужной ему информацией, а для разработчика. Посмотрим сами на сообщение об ошибке доступа к серверу данных, который в данный момент не запущен, — рис. 13.1. Выглядит жутковато...

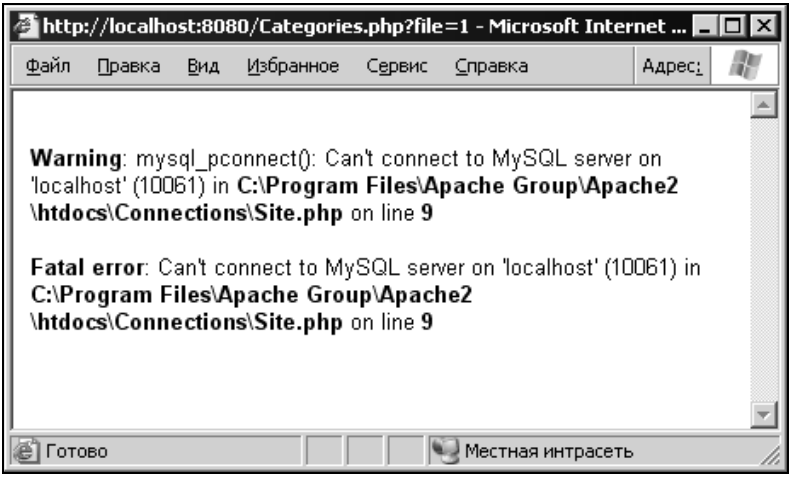


Рис. 13.1. Сообщение об ошибке доступа к серверу данных, выведенное обработчиком PHP

Пожалуй, увидев раз такое сообщение, посетитель испугается и больше не зайдет на наш сайт. А нам нельзя терять посетителей — ведь этот сайт мы сделали для них и только для них! Так что давайте что-то делать со стандартными сообщениями об ошибках PHP.

А что мы можем сделать? Сейчас разберемся.

Как мы уже заметили, в PHP каждая встроенная функция, предназначенная для работы с данными, возвращает некий результат. Это может быть идентификатор соединения с сервером, набор записей или просто сообщение о том, нормально ли была выполнена какая-то операция или нет. Вот этот результат мы и можем использовать для того, чтобы выяснить, произошла ли ошибка, и, если нужно, вывести посетителю предупреждение.

Первое, что нам необходимо проверить, было ли установлено соединение с самим сервером записей. Для этого откроем файл Site.php, хранящийся в папке Connections корневой папки нашего сайта, и посмотрим на его содержимое. Как мы помним из *главы 8*, этот файл содержит сценарий PHP, выполняющий подключение к серверу данных MySQL.

```
$hostname_Site = "localhost";
$database_Site = "site";
$username_Site = "site";
$password_Site = "site";
$Site = mysql_pconnect($hostname_Site, $username_Site, $password_Site) or
↳trigger_error(mysql_error(),E_USER_ERROR);
```

Здесь Dreamweaver уже позаботился о том, что подключение к серверу данных может оказаться неудачным, и вставил код для вывода сообщения об ошибке (встроенная функция `trigger_error`). Но, поскольку мы хотим по возможности избавиться от стандартных сообщений об ошибках, давайте перепишем окончание этого сценария вот так:

```
$Site = mysql_pconnect($hostname_Site, $username_Site, $password_Site);
if (!($Site)) {
  echo "<PRE>Произошла ошибка подключения к серверу данных.\r\n";
  echo "Напишите администратору сайта - admin@mail.somesite.ru.</PRE>";
  die();
}
```

Здесь мы проверяем, что вернула функция `mysql_pconnect`. Если она вернула идентификатор соединения с сервером, то не делаем ничего. Если же она вернула ноль (соединение с сервером не было установлено), то выводим сообщение для посетителя и прерываем выполнение сценария функцией `die`.

Давайте впишем приведенный ранее исправленный код в файл Site.php, сохраним его, опубликуем на Web-сервере и проверим в работе. Для этого запустим Web-сервер, но сервер данных запускать не будем. Далее откроем

в Web-обозревателе главную страницу нашего сайта и попытаемся попасть на одну из страниц списка категорий.

Но что это? Перед текстом нашего предупреждения все равно выводится стандартное сообщение об ошибке PHP. Как его подавить?

Специально для подавления некоторых или всех сообщений об ошибках в PHP предусмотрена встроенная функция `error_reporting`. Эта функция принимает один-единственный аргумент — целочисленный код типа ошибок, стандартные сообщения о которых будут выводиться. Автор этой книги экспериментальным путем подобрал значение этого аргумента, равное 2045; полное же описание данной функции и всех типов ошибок приведено в руководстве по PHP.

Итак, давайте вставим вызов этой функции в сценарий Site.php:

```
error_reporting(2037);
$Site = mysql_pconnect($hostname_Site, $username_Site, $password_Site);
if (!($Site)) {
  echo "<PRE>Произошла ошибка подключения к серверу данных.\r\n";
  echo "Напишите администратору сайта - admin@mail.somesite.ru.</PRE>";
  die();
}
```

После этого сохраним этот файл, опубликуем его на Web-сервере и снова проверим. Ага, теперь работает — Web-обозреватель выводит только текст нашего предупреждения.

Вообще, использовать функцию `error_reporting` нужно только в уже готовых серверных страницах, предназначенных для публикации в Сети. Во время отладки лучше всего будет закомментировать ее, поставив в начале выражения, содержащего ее вызов, знак комментария `//`. Если же необходимо разрешить вывод стандартных предупреждений обо всех ошибках, то достаточно вставить в нужное место сценария такое выражение:

```
error_reporting(2047);
```

А выражение:

```
error_reporting(0);
```

подавляет вывод стандартных предупреждений PHP обо всех ошибках. Так что осторожнее с ним!

С процедурой подключения к серверу данных мы покончили. Возьмемся теперь за процедуры выбора базы данных и выполнения запросов SQL. Код, выполняющий их, находится во всех серверных страницах, так что работа нам предстоит немалая.

Откроем пользовательскую страницу списка категорий Categories.php и переключимся в режим отображения кода HTML. Найдем в сценарии выражение, выполняющее выбор базы данных. Вот оно:

```
mysql_select_db($database_Site, $Site);
```

Функция `mysql_select_db` возвращает `true` в случае успешного подключения к базе данных и `false` в противном случае. В сценарии, созданном Dreamweaver, возвращаемый ею результат не использовался. А мы его используем!

Удалим из сценария приведенное ранее выражение и заменим его вот таким фрагментом кода:

```
if (!(mysql_select_db($database_Site, $Site))) {
  echo "<PRE>Произошла ошибка подключения к базе данных site.\r\n";
  echo "Напишите администратору сайта - admin@mail.somesite.ru.</PRE>";
  die();
}
```

Этот фрагмент кода очень похож на тот, что мы написали ранее, так что комментировать мы его не будем. Единственное, что можно сказать, это то, что вызов функции `error_reporting` здесь ставить не нужно, поскольку он уже есть в файле Site.php. А этот файл обрабатывается при вызове каждой страницы, обращающейся к данным, поскольку подключен к ней с помощью оператора `require_once`.

База данных сдалась под нашим напором. На очереди — хитрые запросы SQL и коварные наборы записей. Давайте примемся и за них.

На странице Categories.php выполняется всего один запрос SQL, так что расправиться с ним будет просто. Вот выражение, которое его выполняет:

```
$Categories = mysql_query($query_Categories, $Site) or
↳die(mysql_error());
```

Функция `mysql_query` возвращает набор записей, если запрос SQL был успешно выполнен, и `false` в противном случае. Если же запрос не может вернуть набор записей (например, это запрос на добавление, изменение и удаление записи), то при удачном его выполнении возвращается `true`, а при неудачном — опять `false`.

Убираем это выражение из сценария и вписываем на его место вот что:

```
$Categories = mysql_query($query_Categories, $Site);
if (!($Categories)) {
  echo "<PRE>Произошла ошибка выполнения запроса SQL.\r\n";
  echo "Напишите администратору сайта - admin@mail.somesite.ru.</PRE>";
  die();
}
```

Комментарии не нужны — тут все то же самое, что мы делали ранее.

А вот со страницей Items.php возникает небольшая сложность. Дело в том, что в ее сценариях выполняются два запроса SQL — один за другим, — и в результате получаются два набора записей. И когда мы будем выводить сообщение об ошибке выполнения второго запроса SQL, нам надо будет выгрузить из памяти серверного компьютера результат выполнения первого.

Вот выражение, выполняющее второй запрос:

```
$Category = mysql_query($query_Category, $Site) or die(mysql_error());
```

На его место нужно вписать вот такой код:

```
$Category = mysql_query($query_Category, $Site);
if (!($Category)) {
  echo "<PRE>Произошла ошибка выполнения запроса SQL.\r\n";
  echo "Напишите администратору сайта - admin@mail.somesite.ru.</PRE>";
  mysql_free_result($Items);
  die();
}
```

Остальные изменения, которые нам нужно будет сделать в сценариях страницы Items.php, будут аналогичны тем, что приведены ранее.

Другие страницы сайта также не вызовут у нас затруднений. Нужно только подставить в соответствующее место их сценариев приведенные ранее фрагменты кода и поменять имена переменных.

Сделав все необходимые изменения, проверим обновленные странички в работе. Для этого можно специально внести в код запросов SQL ошибки, например, добавить в конец имени каждой таблицы несколько цифр. В ответ Web-обозреватель должен вывести текст предупреждения об ошибке, который мы задали.

Вообще, написанные нами выражения "отлова" ошибок крайне неоптимальны. Давайте посмотрим на них — текст сообщений об ошибке, да и сами вызовы функций повторяются во всех сценариях практически без изменений. А это очень плохой стиль программирования.

Давайте вынесем повторяющиеся участки кода PHP в функции. Создадим новую пустую страницу PHP, сохраним ее в корневой папке сайта под именем Library.php, откроем в Dreamweaver и переключимся в режим отображения кода HTML. Эта страница будет содержать только код PHP, объявляющий функции:

- `selectDatabase` — подключения к базе данных;
- `runQuery` — выполнения запроса.

Функция `selectDatabase` будет самой простой. Вот код ее объявления:

```
require_once('Connections/Site.php');
// Нам понадобятся переменные, объявленные в файле Site.php
function selectDatabase() {
  global $database_Site, $Site;
  if (!(mysql_select_db($database_Site, $Site))) {
    echo "<PRE>Произошла ошибка подключения к базе данных site.\r\n";
    echo "Напишите администратору сайта - admin@mail.somesite.ru.</PRE>";
    die();
  }
}
```

Видно, что она не принимает аргументов и не возвращает результата. Обратите также внимание на то, что она использует переменные `$database_Site` и `$Site`, объявленные в файле Site.php. Чтобы получить к этим переменным доступ из тела функции, мы превратили их в глобальные с помощью ключевого слова `global`.

Функция `runQuery` будет чуть сложнее:

```
function runQuery($query, $unloadRecordset = NULL) {
  global $Site;
  $q = mysql_query($query, $Site);
  if ($q) {
    return $q;
  } else {
    echo "<PRE>Произошла ошибка выполнения запроса SQL.\r\n";
    echo "Напишите администратору сайта - admin@mail.somesite.ru.</PRE>";
    if (isset($unloadRecordset)) mysql_free_result($unloadRecordset);
    die();
  }
}
```

Первый аргумент этой функции — код запроса SQL в строковом виде. Второй аргумент не является обязательным и содержит набор записей, который нужно выгрузить из памяти в случае возникновения ошибки. Для проверки, был ли задан второй аргумент, мы использовали выражение:

```
if (isset($unloadRecordset)) mysql_free_result($unloadRecordset);
```

Значение по умолчанию для второго аргумента — `NULL` (см. список аргументов в объявлении функции). Значит, встроенная функция `isset` вернет `true`, если при вызове функции `runQuery` для второго аргумента было задано значение, и выполнится функция `mysql_free_result`. Так что функция `runQuery` у нас получилась универсальной.

Осталось только сказать, что функция `runQuery` возвращает в качестве результата полученный после выполнения запроса SQL набор записей.

Что ж, кажется, выяснили все... Теперь вписываем код объявления обеих этих функций в файл Library.php, не забыв заключить его в тег `<?php...?>`. После этого в начало каждой из страниц, в которой будут использованы эти функции, подставляем выражение

```
<?php require('Library.php'); ?>
```

или

```
<?php require('../Library.php'); ?>
```

в зависимости от того, в какой папке сохранена страница — корневой или папке Admin. Данное выражение заставляет обработчик PHP выполнить серверную страницу, указанную в качестве аргумента, всякий раз, когда оно встретится в сценарии. Это гарантирует, что обработчик PHP прочитает код объявления обеих созданных нами функций перед тем, как их выполнит.

Теперь нам осталось вписать в сценарии всех страниц вместо выражений, выполняющих подключение к базе данных и выполнение запросов SQL, вызов соответствующей функции. И можно считать, что мы реализовали вывод сообщений об ошибках.

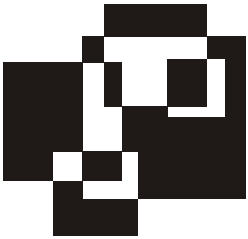
Вообще, будет неплохой идеей перенести в файл Library.php также функции, созданные самим Dreamweaver, — `isAuthorized` и `GetSQLValueString`. (Описание первой из этих функций приведено в *главе 11*, второй — в *главе 9*.) Наши серверные страницы после этого станут только компактнее.

Вообще, сообщения об ошибках, которые мы создали, слишком уж непрезентабельны. Они выводятся в виде обычного текста, что может не понравиться некоторым пользователям. Но ведь мы пока что учимся, не так ли? Когда мы наберемся опыта, то сможем сделать красивые сообщения в формате HTML, целые Web-страницы, разъясняющие посетителю, что нужно сделать в случае возникновения ошибки. И посетители будут довольны.

Что дальше?

Так, с выводом сообщений об ошибках на русском языке мы покончили. Теперь стандартные сообщения, выводимые обработчиком PHP, не распугают посетителей нашего сайта.

Далее на очереди — реализация настройки сайта под каждого посетителя. Мы дадим возможность пользователям задавать порядок сортировки статей и файлов, причем так, чтобы выбранный им порядок сохранялся, когда он вернется на наш сайт. А значит, для этого нам придется решить проблему хранения данных на компьютере клиента.



Глава 14

Хранение данных на стороне клиента

Следующее, что мы реализуем на нашем сайте, — это некоторые возможности по его настройке. Сделаем так, чтобы любой посетитель, неважно, администратор, ведущий или просто гость, мог задать порядок сортировки статей и файлов, и этот порядок автоматически применялся, когда он снова зайдет на сайт. Так мы сделаем свой сайт чуть более дружественным к посетителям.

Задание порядка сортировки — задача несложная. Необходимо просто подставить нужное имя поля после ключевого слова `ORDER BY` в запросе SQL выборки записей. Это мы уже делали, когда создавали свои первые серверные страницы в *главе 8*.

А вот хранение выбранного посетителем порядка сортировки — задача уже весьма нетривиальная. Где его хранить? В каком виде? Все эти вопросы нам придется решать во время работы над страницей Items.php (списком статей и файлов).

Отложим их решение на потом. Пока что давайте реализуем само задание порядка сортировки. Для этого нам придется немного переделать код сценариев PHP, созданных Dreamweaver на странице Items.php. Мы уже имеем какой-никакой опыт "ручного" программирования на PHP, так что — вперед!

Задание порядка сортировки записей

Запустим Dreamweaver, если он еще не запущен, и откроем в нем страницу Items.php. Переключимся в режим отображения кода HTML и найдем строку, задающую код запроса SQL, выбирающего записи из таблицы `items`. Найдем мы ее без труда:

```
$query_Items = sprintf("SELECT author, name, added, href FROM items WHERE
↳catid = %s ORDER BY added DESC", $colname_Items);
```

Здесь `$colname_Items` — это переменная, хранящая код категории, выбранной посетителем на странице Categories.php.

Порядок сортировки здесь задан жестко — прямо в коде запроса. Видно, что сортировка записей выполняется по полю `added`, причем по убыванию. Чтобы задать сортировку по другим полям таблицы `items` (а их всего два — `author` и `name`; по полю `href` сортировать бесполезно), нам нужно подставить имя необходимого поля в код запроса сразу после ключевого слова `ORDER BY`.

Объявим переменную `$sortOrder`, хранящую имя поля, по которому должна производиться сортировка. И перепишем приведенное ранее выражение таким образом:

```
$sortOrder = "added DESC";
$query_Items = sprintf("SELECT author, name, added, href FROM items WHERE
↳catid = %s ORDER BY %s", $colname_Items, $sortOrder);
```

Теперь нам нужно как-то принять от посетителя порядок сортировки, который он хочет задать для статей или файлов, перечисленных на странице. Как это сделать? Существуют два разных способа, один — попроще, другой — посложнее.

Первый способ (простой) заключается в том, что для задания сортировки используются особые гиперссылки, указывающие на эту же страницу. Интернет-адреса этих гиперссылок имеют такой вид:

Items.php?catid=<*код текущей категории*>&sort=<*порядок сортировки*>

То есть порядок сортировки пересылается этой же странице в аргументе `sort` методом `GET`. Заметим, что нам также нужно переслать и код текущей категории (аргумент `catid`), иначе при своем последующем открытии страница Items.php выведет неизвестно что.

Практически всегда гиперссылки, управляющие порядком сортировки, помещаются в той же таблице, что и сами данные из базы, а именно в строке заголовка. Уже стало традицией превращать в такие гиперссылки заголовки столбцов таблицы. И если посетитель хочет отсортировать записи по другому полю, он уже автоматически щелкает по заголовку соответствующего столбца. (И, бывает, удивляется, если ничего не происходит.)

Второй способ (сложный) отличается от первого тем, что для сбора данных о порядке сортировки использует специальную форму. Его преимущество заключается в том, что мы можем предоставить посетителю возможность задать множество разных параметров: имя поля, по которому выполняется сортировка, направление (то есть по возрастанию или по убыванию будут сортироваться записи) и т. п. Также форма, как правило, заметнее, чем гиперссылки, и посетитель сразу видит, что на этой странице он может управлять сортировкой записей.

А в остальном второй способ не отличается от первого. Данные, собранные формой, пересылаются этой же странице методом `GET`. Также будет нужно

переслать и код текущей категории, для хранения которого можно использовать специально созданное в форме скрытое поле.

Как правило, на сайтах, хранящих данные в базах, для задания сортировки используется первый способ. Его возможностей в большинстве случаев хватает, да и реализовать этот способ проще. Давайте и мы возьмем его на вооружение.

Прокрутим HTML-код страницы Items.php вниз и найдем тот его участок, который создает заголовки столбцов таблицы. Вот он:

```
<TH SCOPE="col">Автор</TH>
<TH SCOPE="col">Название</TH>
<TH SCOPE="col">Добавлено</TH>
<TH SCOPE="col"> </TH>
```

Превратим текст этих заголовков в гиперссылки, задающие порядок сортировки (см. ранее):

```
<TH SCOPE="col"><A HREF="Items.php?catid=<?php echo $colname_Items;
↳?>&sort=1">Автор</A></TH>
<TH SCOPE="col"><A HREF="Items.php?catid=<?php echo $colname_Items;
↳?>&sort=2">Название</A></TH>
<TH SCOPE="col"><A HREF="Items.php?catid=<?php echo $colname_Items;
↳?>&sort=3">Добавлено</A></TH>
<TH SCOPE="col"> </TH>
```

Заметим, что вместо имени поля мы пересылаем просто целое число. Почему? Дело в том, что если бы мы пересылали имена полей, их любой смог бы увидеть в строке адреса Web-обозревателя, ведь аргументы, пересылаемые методом `GET`, передаются как часть интернет-адреса. А разглашать сведения о структуре нашей базы данных не стоит — злоумышленник может использовать их для взлома нашего сайта, что нам совсем не нужно.

Теперь осталось вернуться в начало страницы и добавить код PHP, реализующий проверку значения аргумента `sort` и присвоение нужного имени поля переменной `$sortOrder`. Вот этот код:

```
$sort = "3";
if (isset($_GET['sort'])) {
  $sort = (get_magic_quotes_gpc()) ? $_GET['sort'] :
  ↳addslashes($_GET['sort']);
}
switch ($sort) {
  case "1":
    $sortOrder = "author";
```

```
    break;
  case "2":
    $sortOrder = "name";
    break;
  case "3":
    $sortOrder = "added DESC";
}
$query_Items = sprintf("SELECT author, name, added, href FROM items WHERE
↳catid = %s ORDER BY %s", $colname_Items, $sortOrder);
```

К данному коду вряд ли нужны особые пояснения — все нам уже знакомо. Так что сразу же вставим его в нужное место страницы, сохраним ее, опубликуем на Web-сервере и попробуем в действии. Работает? Замечательно!

Хранение настроек посетителя

Теперь можно задуматься над тем, где нам сохранить значение переменной `$sort` на время, пока посетитель блуждает где-то вне нашего сайта. Как говорилось ранее, задача это нетривиальная, и поиск ее решения может затянуться надолго...

Способы хранения настроек

Вообще, способов сохранить настройки, сделанные посетителем сайта, существует целых три. Мы рассмотрим их по очереди.

Первый способ вполне очевиден. Для хранения настроек, заданных посетителем нашего сайта, мы можем использовать специальную таблицу базы данных или специальные поля уже существующей таблицы. Благо наш сайт и так использует базу данных для хранения списков категорий и статей с файлами.

Для хранения настроек посетителя мы можем создать отдельную таблицу, назвав ее, скажем, `settings`. Эта таблица должна содержать следующие поля:

- имя посетителя или иной уникальный идентификатор, по которому можно будет найти нужную запись. На основе этого поля необходимо создать ключевой индекс;
- по одному полю для каждой настройки, что мы предоставили посетителю, либо одно поле типа blob, в котором будут храниться сразу все настройки.

Чтобы прочитать нужные настройки, мы создадим запрос SQL, выполняющий поиск по полю `name`, и выполним его. Если такой посетитель найден, мы прочитаем значения полей полученной записи и применим их.

Поскольку у нас уже есть таблица `users`, хранящая всех зарегистрированных пользователей (администраторов и ведущих), мы можем добавить поля для хранения настроек прямо в эту таблицу. Так даже будет проще: для хранения настроек не нужно создавать отдельную таблицу, а значит, наши запросы SQL будут проще (и, следовательно, будут быстрее выполняться).

Недостаток этого способа хранения настроек посетителя мы уже заметили. Нам либо придется давать возможность настройки страниц сайта только зарегистрированным пользователям, либо заносить в таблицу `users` всех посетителей сайта. Но в первом случае мы оттолкнем от сайта гостей, не записанных в список "особо приближенных" (в таблицу `users`), а во втором "раздуем" базу данных до слишком больших размеров. А это нам не подходит.

Хотя если мы собираемся давать зарегистрированным пользователям какие-то особые возможности по настройке сайта, то эти настройки вполне могут храниться в базе данных. Такое может быть, например, в случае, если мы создаем электронный магазин. (Кстати, в электронных магазинах так и делается.)

Второй способ предусматривает хранение настроек в переменных уровня сессии. Например, в нашем случае мы можем записать содержимое переменной `$sort` в переменную уровня сессии:

```
session_start();
$_SESSION["sort"] = $sort;
```

а потом извлечь его и использовать:

```
session_start();
$sort = $_SESSION["sort"];
switch ($sort) {
. . .
```

Второй способ не имеет одного из недостатков первого. Здесь нам не нужно заносить всех посетителей в таблицу, чтобы сохранить сделанные ими настройки. С другой стороны, возможность сохранения настроек доступна всем, даже гостям.

Но здесь возникает другая проблема. Дело в том, что переменные уровня сессии хранятся на Web-сервере весьма ограниченное время. Как только посетитель уйдет с сайта, обработчик PHP отсчитывает определенный интервал времени (таймаут) и удаляет все переменные его сессии. Этот таймаут задается в настройках обработчика PHP и по умолчанию составляет 180 минут, или 3 часа. И если посетитель вернется на наш сайт позже, чем через три часа, все заданные им настройки пропадут.

Но настройки пропадут, даже если посетитель вернется на сайт раньше, чем через три часа. Ведь сессия при этом будет уже другая, не та, в которой были сохранены эти настройки. Еще одна проблема...

В идеале, настройки посетителя сайта должны храниться где-нибудь на его компьютере (на стороне клиента, как говорят профессиональные программисты). Тогда в какое бы время ни зашел посетитель на сайт, его настройки всегда будут доступны для наших сценариев PHP. Но как это реализовать?

Cookie и их использование

Очень просто! Для этого достаточно использовать cookie, создание которых поддерживают практически все современные Web-обозреватели.

Cookie — это небольшие текстовые файлы, создаваемые Web-обозревателем и сохраняемые на клиентском компьютере в особой папке. Эти файлы содержат имена и значения аргументов, переданных в особым образом сформированном серверном ответе. Именно они в большинстве случаев применяются для хранения настроек посетителя и другой несекретной информации.

При создании cookie привязываются к интернет-адресу сайта, пославшего серверный ответ. Когда Web-обозреватель посылает клиентский запрос Web-серверу, он извлекает с диска клиентского компьютера cookie, отправленный этим сервером ранее (если, конечно, такой cookie есть), и посылает его в составе запроса. Сценарий PHP, расположенный на этом Web-сервере, может прочитать отправленные Web-обозревателем cookie и извлечь сохраненные в нем аргументы.

Также при создании cookie сценарий PHP может задать интервал времени, по истечении которого cookie будет удален с диска клиентского компьютера. Если интервал времени не задан, cookie будет удален сразу же после закрытия программы Web-обозревателя.

Внимание

Все Web-обозреватели, поддерживающие cookie, позволяют пользователю запретить их прием с определенных или сразу всех сайтов. Поэтому если мы собираемся использовать cookie, то должны будем сообщить на страницах нашего Web-сайта, чтобы посетитель включил в своем Web-обозревателе поддержку cookie.

Также некоторые или все cookie могут блокироваться прокси-серверами или брандмауэрами, через которые посетитель выходит в Интернет. (О прокси-серверах и брандмауэрах см. *главу 4*.) По вопросам снятия блокировки следует обращаться к администраторам, обслуживающим эти программы.

PHP имеет встроенные средства обработки cookie. Использовать их очень просто, и мы в этом сейчас убедимся.

Для создания cookie на компьютере клиента и помещения в него какого-либо аргумента нужно использовать встроенную функцию `setcookie`. Формат ее вызова таков:

```
setcookie(<имя аргумента>, <значение аргумента>[, <время уничтожения
↳cookie>]);
```

Эта функция принимает три аргумента. Первым ей передается имя создаваемого в cookie аргумента в строковом виде. Вторым аргументом мы должны передать значение аргумента cookie, также в строковом виде. А третий — необязательный аргумент — задает время, через которое cookie должен быть удален с компьютера клиента. С первыми двумя аргументами все понятно, а вот о третьем нужно поговорить подробнее.

Прежде всего, время уничтожения cookie задается как количество секунд, прошедших с особой даты, называемой "началом эпохи Unix" (полночь 1 января 1970 года). Этот формат задания времени так и называется — *формат Unix*.

Чтобы получить нужное нам значение времени уничтожения cookie в формате Unix, нам потребуется сначала получить значение текущего времени в этом же формате, а затем прибавить к нему нужное количество секунд. Для получения текущего времени в формате Unix мы можем вызвать встроенную функцию `time`, не принимающую аргументов. Ну, а пересчитать дни в секунды — задача элементарной математики.

Так, предположим, нам нужно получить время, отстоящее от текущего на 300 дней в будущем. Для этого мы напишем такое выражение:

```
$fitureTime = time() + 60 * 60 * 24 * 300;
```

Здесь мы умножаем 60 секунд в минуте на 60 минут в часе, потом — на 24 часа в сутках, а напоследок — на 300 суток. И все это прибавляем к значению, возвращенному функцией `time`.

Замечание

Вообще-то, функция `setcookie` принимает больше аргументов, но нам сейчас пригодятся только три из них. Полное описание этой функции есть в руководстве по языку PHP.

Осталось только сказать, что функция `setcookie` возвращает в качестве результата `true`, если cookie был успешно создан, и `false` в противном случае. Как правило, этот результат игнорируется.

Выяснив все о функции `setcookie`, давайте создадим cookie с аргументом `sort`:

```
setcookie("sort", $sort, time() + 60 * 60 * 24 * 300);
```

Внимание

Вызов функции `setcookie` должен предшествовать любому коду HTML.

После создания cookie мы можем получить доступ к хранящимся в нем аргументам и их значениям, воспользовавшись встроенным массивом `$_COOKIE`:

```
echo $_COOKIE["sort"];
```

Чтобы удалить cookie раньше заданного нами времени, достаточно еще раз вызвать функцию `setcookie`, передав ей третьим аргументом любое уже прошедшее значение времени. Например:

```
setcookie("arg", "", time() - 3600);
```

Здесь мы присваиваем аргументу `arg` пустую строку и задаем время уничтожения cookie — час назад. Web-обозреватель при получении cookie проверит время и, выяснив, что оно уже прошло, тут же удалит cookie.

Реализация хранения настроек в cookie

Ну что ж, о cookie мы выяснили все, что нам нужно. Давайте теперь воплотим наши знания в код PHP. Откроем страницу Items.php, если уже закрыли ее, и переключимся в режим отображения кода HTML Dreamweaver.

Сначала найдем фрагмент сценария PHP, считывающий значение аргумента `sort`, переданного странице методом `GET`. Он нам уже знаком, ведь мы сами его писали в *главе 13*:

```
$sort = "3";
if (isset($_GET['sort'])) {
  $sort = (get_magic_quotes_gpc()) ? $_GET['sort'] :
  ↳addslashes($_GET['sort']);
}
```

Сюда нам будет нужно вставить выражение, прочитывающее значение аргумента `sort`, сохраненного в cookie. Тогда приведенный ранее фрагмент кода будет выглядеть так (вставленное выражение выделено полужирным шрифтом):

```
$sort = "3";
if (isset($_GET['sort'])) {
  $sort = (get_magic_quotes_gpc()) ? $_GET['sort'] :
  ↳addslashes($_GET['sort']);
} else {
  if ((isset($_COOKIE['sort'])) && ($_COOKIE['sort'] != "")) {
    $sort = $_COOKIE['sort'];
  }
}
```

Здесь мы сначала проверяем, передан ли странице методом `GET` аргумент `sort`, и, если он передан, считываем его значение. Если же такого аргумента нет, то проверяем, существует ли в cookie аргумент `sort`, и, если он существует, берем данные о порядке сортировки из него. Если же нет и cookie с таким аргументом, мы применяем сортировку по умолчанию.

Теперь нам нужно сохранить полученное значение в cookie. Давайте сделаем это немедленно, для чего вставим сразу же после приведенного ранее фрагмента кода вот такое выражение:

```
setcookie("sort", $sort, time() + 60 * 60 * 24 * 300);
```

Оно нам уже знакомо, так что никаких разъяснений здесь не требуется.

Вставим два этих фрагмента кода в нужное место страницы Items.php, сохраним ее, опубликуем на Web-сервере и испытаем. Только перед испытанием проверим, включена ли в нашем Web-обозревателе поддержка cookie. О том, как ее включить, написано в интерактивной документации, поставляемой вместе с Web-обозревателем.

Какие данные стоит хранить в cookie

Напоследок поговорим о том, какие данные стоит и какие — не стоит хранить в cookie.

Заметим, что cookie при создании автоматически привязываются к определенному интернет-адресу и в дальнейшем отправляются только тому Web-серверу, который расположен по этому адресу. Так что cookie вроде бы защищены от перехвата с других интернет-адресов злоумышленниками. Но дело в том, что подделать интернет-адрес в серверном ответе — задача не слишком сложная; любой хакер владеет приемами и программами, позволяющими это сделать. Так что cookie — вещь потенциально небезопасная.

Замечание

Несколько лет назад Европарламент даже пытался законодательно запретить использование cookie. В качестве причины называлась именно их небезопасность.

Исходя из этого, можно сказать, что в cookie следует хранить только несекретные данные, например, настройки сайта. Даже если хакер и прочитает их, то все равно никакой выгоды от них не получит. Будет совсем хорошо, если эти настройки не станут отражать реальную структуру базы данных, деталей кода PHP и пр., будут как можно более абстрактными. Так, мы вместо имен полей таблицы `items` использовали ничего не значащие, на взгляд постороннего, числа. Наш сценарий знает, что с ними делать, а злоумышленник — нет.

А вот имена пользователей, пароли, идентификаторы кредитных карточек и прочие секретные сведения хранить в cookie категорически не рекомендуется. Или, по крайней мере, шифровать их как можно более устойчивыми к взлому алгоритмами. Пусть злоумышленник помучается!..

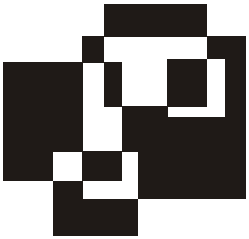
И еще. Не следует забывать, что посетитель нашего сайта ничем нам не обязан. Он может с легкостью отключить в своем Web-обозревателе поддержку

cookie, и винить его в этом не нужно. Можно только предупредить его на страницах сайта, что для доступа к некоторым возможностям ему придется включить поддержку cookie. Но даже в этом случае не стоит надеяться, что он это сделает.

Что дальше?

Вот мы и создали простейшие средства для настройки нашего сайта под конкретного посетителя. Более того, эти настройки сохраняются на компьютере посетителя и автоматически применяются, когда он входит на сайт. Конечно, настроек совсем немного, но это только пока.

Третий шаг — реализация системы управления файлами нашего сайта. Мы сделаем так, чтобы ведущие могли публиковать на сайте свои статьи, пользуясь для этого специально созданными нами Web-страницами. Многие ли сайты могут похвастаться подобным?..



Глава 15

Управление файлами через Web-интерфейс

Итак, что мы уже успели сделать на своем сайте? Выдачу понятных для обычных посетителей сообщений об ошибках — раз. Некоторые возможности настройки страниц сайта посетителями с хранением этих настроек на компьютере клиента — два. Неплохо. И самое главное — очень просто, благо PHP предоставляет все нужные средства.

Но все это просто "красивости". В данной главе мы займемся намного более нужными и полезными вещами.

Идет время. Наш сайт растет и развивается. Количество доступных статей и файлов постоянно растет — администраторы и ведущие работают не покладая рук. Посетители приходят и остаются, более того — рассказывают о нашем сайте другим пользователям, которые тоже приходят и остаются. Популярность сайта постепенно растет.

А на соседнем бесплатном Web-сервере какие-то выскочки опубликовали другой сайт-архив статей и файлов. Сделали они его самостоятельно, без нашей помощи, или "творчески" позаимствовали идею у нас — неизвестно, да и неважно. Важно другое — этот конкурирующий сайт тоже имеет своих постоянных посетителей. И популярность его равна популярности нашего сайта, если не больше.

Как нам выжить в жестокой борьбе за поклонников? Разве для того мы делали свой сайт, чтобы проиграть в ней? Нет, нет и еще раз нет!

Давайте подумаем, что мы можем сделать, чтобы оттянуть посетителей сайта-конкурента и привлечь их на свой сайт. Сделать красивый дизайн? Вряд ли — этим сейчас никого не удивишь. Заставить свой сайт работать быстрее? Тоже вряд ли — конкурент также создан на связке PHP + MySQL, поэтому никакая оптимизация особо не поможет. Изобрести что-то совсем уж хитроумное и эксклюзивное? Пока мы будем изобретать, сами не заметим, как вылетим в трубу...

Есть! Замечательный способ привлечь посетителей на наш сайт — разместить на нем авторские статьи и программы, написанные специально для

нашего сайта. Расположим прямо на главной странице сайта текст, вдохновляющий посетителей на написание статей, и почтовый адрес, куда им следует эти самые статьи присылать. Через некоторое время в нашем электронном почтовом ящике окажутся несколько статей, которые мы решим опубликовать на сайте. И опубликуем — это нетрудно, имея Dreamweaver.

Популярность нашего сайта резко подскочит вверх. Посетители в восторге, администраторы сайта-конкурента в растерянности. Авторов становится все больше и больше, и мы уже просто завалены статьями. Просматривать их нам некогда. Лучше всего будет переложить отбор и публикацию подходящих материалов на администраторов и ведущих. Возможно, для этого придется увеличить их штат, но это совсем не сложная задача, и с ней мы справимся — дадим еще одно объявление на нашем сайте. Сложнее другие — как дать администраторам и ведущим возможность публиковать статьи и файлы на нашем сайте, но при этом не позволить им хозяйничать в написанных нами Web-страницах. Как сохранить страницы Categories.php, Items.php, все, что хранится в папке Admin, от нежелательных глаз — вот задачка!..

Разумеется, это можно сделать. Даже не одним способом, а двумя. Сейчас мы их рассмотрим.

Два способа управления файлами на сайте

Как водится, один из этих способов простой, а другой — сложный. И рассмотрим мы их в порядке от простого к сложному.

Простой способ — дать всем, кому нужно публиковать файлы на нашем сайте, то имя пользователя, под которым мы записаны в списке пользователей FTP-сервера, и пароль. Пусть они подключаются к FTP-серверу с помощью программы FTP-клиента или того же Dreamweaver и публикуют свои файлы на сайте. То есть поступают точно так же, как и мы.

Сделать это, конечно, просто. Но это та самая простота, которая, как гласит пословица, хуже воровства. И вот почему.

- Для публикации файлов на сайте нужно специальное программное обеспечение. Клиенты FTP, а уж тем более Web-редакторы с возможностями публикации сайтов не входят в число широко распространенных программ. И может случиться так, что у какого-либо администратора или ведущего их не окажется.
- Чтобы публиковать файлы на сайте с помощью клиентов FTP, нужны особые навыки. Этими навыками начинающие интернетчики могут и не обладать.
- Большие проблемы с безопасностью. Подключившись по протоколу FTP к нашему сайту, любой желающий сможет "гулять", где хочет. Он запро-

сто откроет любую Web-страницу, прочитает код PHP, который, возможно, ноу-хау, исправит текст или вставит в страницу что-нибудь непотребное. И мы ничего не сможем сделать.

Вообще-то, если мы сами администрируем свой FTP-сервер, то сможем прописать в списках пользователей всех, кому нужен доступ к сайту, и дать им права только на определенную папку. То же самое доступно и для клиентов платных Web-серверов: кроме всего прочего, они имеют дополнительные возможности администрирования программ-серверов. Но ведь мы нацеливаемся на бесплатный Web-сервер, так что обо всем этом нам остается лишь мечтать...

Единственное, что мы можем сделать, — это перейти ко второму, *сложному, способу*. Заключается он в том, что для управления файлами на сайте используются специально написанные Web-страницы, или, как говорят профессионалы, Web-интерфейс.

PHP имеет встроенные средства для управления файлами и папками на сайте: создания, копирования, перемещения, переименования и удаления. Этих средств нам хватит с лихвой. А язык HTML предусматривает для форм специальные элементы управления, с помощью которых можно элементарно отправить серверной программе любой файл. Нам остается только написать эту программу.

Недостаток у данного способа управления файлами всего один — сложность реализации. Нам придется самим писать нужные сценарии PHP, которые будут принимать файлы от Web-обозревателя, копировать их, перемещать, переименовывать и удалять. И, разумеется, нам будет нужно сделать соответствующие формы, с помощью которых пользователь будет выполнять все это.

Зато сколько у второго способа достоинств!

- Для управления файлами не нужно никаких специальных программ — хватит одного Web-обозревателя. А Web-обозреватель сейчас, можно сказать, стандартная программа, зачастую поставляющаяся вместе с операционной системой.
- От пользователя не требуется никаких специальных навыков. Все, что он должен знать, — грубо говоря, на какую кнопку нажать.
- Поскольку мы сами пишем сценарии для управления файлами, мы сами можем задавать, куда пользователи имеют право заходить, а куда их пускать не стоит. Мы можем элементарно скрыть от посторонних реальную структуру сайта, предоставив им доступ к одной-единственной папке, где они и должны будут публиковать свои творения. (Кстати, создание такой папки-"песочницы" (калька с английского sandbox) — обычная практика на реальных сайтах.)

Решено! Выбираем второй способ. И сразу же начинаем реализовывать его на практике.

Отправка файлов на Web-сервер

И начнем мы реализовывать его с того, что создадим средства для отправки файлов на Web-сервер. Ведь прежде чем пользователь сможет управлять файлами в своей папке-"песочнице", он должен поместить туда хотя бы один файл.

Создавать средства для отправки файлов мы будем в два этапа:

- создание нужной формы на Web-странице;
- написание сценария PHP.

Как отправить файл из Web-обозревателя

Для отправки файла из Web-обозревателя используется обычная форма HTML. Формы нам уже давно знакомы — мы рассмотрели их еще в *главе 9* и с тех пор неоднократно применяли на практике.

Вот только элемент управления для отправки файла применяется не совсем обычный. Это так называемое *поле ввода файла*, в которое пользователь должен будет ввести имя отправляемого файла. От обычного поля ввода оно отличается двумя особенностями. Во-первых, оно имеет справа кнопку **Обзор** (Browse), при нажатии которой открывается стандартное диалоговое окно открытия файла Windows, где пользователь может выбрать нужный ему файл. Во-вторых, оно отправляет серверной программе не введенное в него имя файла, а сам этот файл.

Поле ввода файла создается с помощью уже знакомого нам тега `<INPUT>`, атрибут `TYPE` которого имеет значение `file`:

```
<INPUT TYPE="file" NAME="filename">
```

Но это еще не все. Полю ввода файла должно предшествовать скрытое поле, задающее максимальный размер отправляемого файла. Это поле должно иметь имя `MAX_FILE_SIZE` и значение, задающее максимальный размер в байтах.

```
<INPUT TYPE="hidden" NAME="MAX_FILE_SIZE" VALUE="65536">
<INPUT TYPE="file" NAME="filename">
```

Здесь мы задали максимальный размер отправляемого файла, равный 65 536 байт или 64 Кбайт. Этого должно хватить для не слишком большой статьи и средних размеров графического изображения.

И еще. Форма, из которой будет отправляться файл, обязательно должна кодировать данные по методу `multipart/form-data` и передавать данные только методом `POST`. Это важно!

```
<FORM NAME="upload" ACTION="accept.php" ENCTYPE="multipart/form-data"
↳METHOD="POST">
```

```
  <INPUT TYPE="hidden" NAME="MAX_FILE_SIZE" VALUE="65536">
  <INPUT TYPE="file" NAME="sendfile">
  <INPUT TYPE="submit" NAME="submit" VALUE"Отправить">
</FORM>
```

Все, наша форма готова! Остальное возьмет на себя Web-обозреватель.

Как принять отправленный файл

Принять отправленный Web-обозревателем файл на стороне Web-сервера тоже не очень сложно. Благо PHP предоставляет для этого все нужные средства.

Начать необходимо с того, что все принятые файлы помещаются обработчиком PHP в особую служебную папку, которая не является частью никакого Web-сайта; назовем эту папку "отстойником". Нам будет нужно написать сценарий, перемещающий принятые файлы из "отстойника" в нужную папку нашего сайта. Этот сценарий будет совсем простым.

Начать необходимо с того, что сведения обо всех принятых и помещенных в "отстойник" файлах хранятся во встроенном массиве `$_FILES`. Каждый элемент этого массива соответствует принятому файлу и представляет собой вложенный массив, содержащий различные сведения о файле.

Чтобы получить, скажем, имя отправленного файла, мы напишем вот такое выражение:

```
$fileName = $_FILES["sendfile"]["name"];
```

Здесь мы сначала обращаемся к элементу массива `$_FILES`, указав в качестве индекса имя поля ввода файла, из которого он был отправлен (в нашем случае — `sendfile`). Затем мы обращаемся к элементу вложенного массива с индексом `name` — это он содержит имя файла.

Вообще, мы можем получить из массива `$_FILES` такие сведения:

- `$_FILES["sendfile"]["name"]` — имя отправленного файла;
- `$_FILES["sendfile"]["size"]` — размер отправленного файла в байтах;
- `$_FILES["sendfile"]["tmp_name"]` — имя отправленного файла, под которым он хранится в "отстойнике";
- `$_FILES["sendfile"]["error"]` — код ошибки, возникшей при приеме файла;
- `$_FILES["sendfile"]["type"]` — тип отправленного файла.

С первыми тремя параметрами все ясно. А вот на двух последних стоит остановиться.

Код ошибки, возникшей при приеме файла, может принимать пять таких значений:

- `0` — ошибки не было (файл успешно принят);

- `1` — размер принятого файла больше максимального размера, указанного в настройках обработчика PHP;
- `2` — размер принятого файла больше максимального размера, указанного в скрытом поле `MAX_FILE_SIZE` формы (см. ранее);
- `3` — файл принят не полностью;
- `4` — файл вообще не принят.

Разумеется, перед любыми действиями с принятым файлом нужно будет проверить, возникла ли при его приеме ошибка. Если возникла, нужно как-то дать знать об этом пользователю, например, вывести на Web-страницу предупреждающий текст.

Что касается типа файла, то он задается в особом универсальном формате. Это *тип MIME* (Multipurpose Internet Mail Extensions, многоцелевые расширения почты Интернета), используемый для задания типа файлов, включенных в электронные письма. В табл. 15.1 приведены некоторые типы файлов MIME.

Таблица 15.1. *Некоторые типы файлов MIME*

Тип файлов	Тип MIME
Web-страница	`text/html`
Архив RAR	`application/x-tar`
Архив ZIP	`application/x-zip-compressed`
Аудио- или видеозапись ASF	`video/x-ms-asf`
Аудио- или видеозапись WMV	`video/x-ms-wmv`
Аудиозапись AIFF	`audio/aiff`
Аудиозапись AU	`audio/basic`
Аудиозапись MIDI	`audio/mid`
Аудиозапись MP3	`audio/mpeg`
Аудиозапись WAV	`audio/wav`
Аудиозапись WMA	`audio/x-ms-wma`
Видеозапись AVI	`video/avi`
Видеозапись Indeo (IVF)	`video/x-ivf`
Видеозапись MPEG	`video/mpeg`
Визитная карточка, используемая почтовыми программами для хранения данных об адресате	`text/x-vcard`

Таблица 15.1 *(окончание)*

Тип файлов	Тип MIME
Графический файл ART	`image/x-jg`
Графический файл BMP	`image/bmp`
Графический файл GIF	`image/gif`
Графический файл JPEG	`image/jpeg`
Графический файл Macromedia Flash	`application/futuresplash`
Графический файл TIFF	`image/tiff`
Документ Adobe Acrobat	`application/pdf`
Документ Microsoft Excel	`application/x-msexcel`
Документ Microsoft Word	`application/msword`
Документ RTF	`application/msword`
Документ XML	`text/xml`
Обычный текстовый документ	`text/plain`
Приложение	`application/x-msdownload`
Приложение HTML (HTA)	`application/hta`
Таблица стилей CSS	`text/css`

Замечание

Приложение HTML — это особым образом написанная Web-страница, которая хранится в файле с расширением hta и ведет себя как обычное приложение Windows. Процесс создания приложений HTML описан на сайте MSDN (**http://msdn.microsoft.com**).

Тип файла обычно используется для того, чтобы проверить, тот ли файл был отправлен. Например, можно разрешить пользователям отправлять на сайт только Web-страницы, графические изображения и архивы, запретив отправку файлов всех остальных типов. В любом случае, мы его использовать не будем.

Хорошо, отправленные пользователем файлы были благополучно приняты обработчиком PHP и помещены в "отстойник". Теперь нам нужно как-то извлечь их оттуда и поместить в одну из папок нашего сайта. Для этого используем встроенную функцию `move_uploaded_file`, формат вызова которой приведен далее.

```
move_uploaded_file(<временное имя файла>, <новое имя файла>);
```

Первым аргументом этой функции передается имя файла, под которым он был сохранен в "отстойнике". Это имя можно извлечь из массива `$_FILES`, как было показано ранее. Второй аргумент задает путь, по которому должен находиться принятый файл, и его новое имя. В качестве нового имени обычно используется изначальное имя отправленного файла, которое также можно извлечь из массива `$_FILES`.

Функция `move_uploaded_file` возвращает `true`, если файл был успешно перемещен на новое местоположение. Если операция переноса файла не была выполнена (например, был задан несуществующий файл или несуществующий путь), возвращается `false`.

Исходя из всего этого, давайте напишем небольшой сценарий, переносящий принятый файл в папку uploads, находящуюся в корневой папке сайта. И рассмотрим этот код построчно.

```
$tmpFile = $_FILES["sendfile"]["tmp_name"];
```

Получаем имя, под которым файл был сохранен в "отстойнике", и сохраняем в переменной `$tmpFile`.

```
$destFile = $_SERVER["DOCUMENT_ROOT"] . "/uploads/" .
↳$_FILES["sendfile"]["name"];
```

Получаем путь, по которому должен храниться принятый файл, и сохраняем в переменной `$destFile`. В качестве нового имени файла мы использовали его изначальное имя, извлеченное из массива `$_FILES`.

Здесь нужно сделать важное замечание. Обработчик PHP работает на серверном компьютере и имеет доступ к любому его файлу и любой его папке. Поэтому мы должны задать полный путь к корневой папке сайта, иначе он ее не найдет. Для этого мы использовали элемент с индексом `DOCUMENT_ROOT` встроенного массива `$_SERVER` — он содержит полный путь корневой папки нашего сайта относительно файловой системы серверного компьютера. В нашем случае он будет таким:

```
C:\Program Files\Apache Group\Apache2\htdocs
```

Осталось только выполнить собственно перенос файла:

```
move_uploaded_file($tmpFile, $destFile);
```

Теперь отправленный пользователем файл находится в папке uploads. А уж потом мы посмотрим, что это за файл, и определим его на место "постоянного хранения".

Кстати, так поступают практически все сайты, позволяющие посетителям отправлять им файлы. Все эти файлы помещаются в папку uploads (она так и называется), а уже потом администратор или ведущий переносит их в те папки, где они должны находиться. Это позволяет быстро и просто отслеживать, какие файлы были приняты.

Реализация отправки файла

Ну что ж, теоретическую часть мы успешно преодолели. Пора заняться практикой. Создадим Web-страницу для выполнения всех операций по управлению файлами: загрузки, копирования, перемещения, переименования и удаления.

Запустим Dreamweaver, создадим новую пустую страницу PHP, дадим ей название `Управление файлами` и такой же заголовок, ниже добавим подзаголовок `Загрузка файла` и напишем какой-нибудь поясняющий текст. После этого сохраним страницу в папке Admin под именем Files.php.

Теперь создадим форму с такими параметрами:

- имя (поле ввода **Form name** редактора свойств) — `file_upload`;
- серверная программа, которой будет отправлен файл (поле ввода **Action**), — File_upload.php (мы создадим ее потом);
- метод отправки данных (раскрывающийся список **Method**) — `POST` (одноименный пункт списка);
- метод кодирования данных (раскрывающийся список **Enctype**) — `multipart/form-data` (одноименный пункт списка).

Далее поставим текстовый курсор внутри формы и создадим там скрытое поле. Зададим для этого поля имя `MAX_FILE_SIZE` и значение 65 536. После этого введем текст `Файл`, поставим пробел и выберем пункт **File Field** подменю **Form** меню **Form**. Когда поле ввода файла будет создано, щелкнем по нему мышью, чтобы выделить, и зададим для него имя — `sendfile` (поле ввода **FileField name** редактора свойств). Потом создадим новый абзац и в нем — кнопку отправки данных с именем `submit` и надписью `Отправить`. Все, страница с формой для отправки файла готова.

Поскольку к странице Files.php должны иметь доступ только зарегистрированные пользователи — администраторы и ведущие, — нам будет нужно также создать серверное поведение **Restrict Access To Page**. Как это сделать, было описано в *главе 11*.

Теперь нам нужно создать страницу File_upload.php, выполняющую перенос файла в папку uploads. Сначала создадим саму эту папку и сразу же опубликуем ее на Web-сервере. Потом создадим еще одну пустую страницу PHP и сразу же переключимся в режим отображения кода HTML.

Страница File_upload.php будет содержать только сценарий PHP:

```
<?php
if (isset($_POST["MAX_FILE_SIZE"])) {
  $tmpFile = $_FILES["sendfile"]["tmp_name"];
  $destFile = $_SERVER["DOCUMENT_ROOT"] . "/uploads/" .
  ↳$_FILES["sendfile"]["name"];
```

```
  move_uploaded_file($tmpFile, $destFile);
}
header("Location: Files.php");
?>
```

Здесь мы сначала дополнительно проверяем, из той ли формы посланы данные (есть ли в этой форме скрытое поле `MAX_FILE_SIZE`). Если же форма та самая, то выполняем перенос принятого файла в папку uploads и снова переходим на страницу Files.php.

Напоследок откроем страницу default.php и найдем в ней код, выводящий абзац с гиперссылками, указывающими на административные страницы списков категорий. Вот он:

```
<?php if ($isAorM) { ?>
<H2>Администрирование</H2>
<P>Чтобы попасть на административную страницу списка категорий файлов,
щелкните по <A HREF="Admin/Categories_admin.php?file=1">этой
гиперссылке</A>. А <A HREF="Admin/Categories_admin.php?file=0">эта
гиперссылка</A> ведет на список категорий статей.</P>
<?php } ?>
```

Добавим сюда еще один абзац (он выделен полужирным шрифтом):

```
<?php if ($isAorM) { ?>
<H2>Администрирование</H2>
<P>Чтобы попасть на административную страницу списка категорий файлов,
щелкните по <A HREF="Admin/Categories_admin.php?file=1">этой
гиперссылке</A>. А <A HREF="Admin/Categories_admin.php?file=0">эта
гиперссылка</A> ведет на список категорий статей.</P>
<P><A HREF="Admin/Files.php">Эта</A> гиперссылка ведет на страницу
управления файлами.</P>
<?php } ?>
```

Переменная `$isAorM` содержит значение `true`, если посетитель является администратором или ведущим.

Сохраним эти три страницы, опубликуем их на Web-сервере и проверим в действии. Выберем какой-нибудь не очень большой файл и попробуем отправить его на Web-сервер. Должно получиться; а если почему-то не получилось, проверим, все ли мы сделали правильно.

Внимание

Нужно обязательно иметь в виду, что администратор удаленного Web-сервера может настроить обработчик PHP так, чтобы он не допускал отправку файлов на сервер. Как правило, на бесплатных Web-серверах он именно так и настроен.

Управление файлами, находящимися на Web-сервере

А теперь можно приступать к созданию системы управления файлами, уже находящимися на Web-сервере. Эта система должна позволять пользователям делать следующее:

- копировать, перемещать, переименовывать и удалять файлы;
- создавать и удалять папки.

Этого должно хватить им для публикации своих статей и исправления ошибок, если такие случатся.

Для публикации статей мы создадим на своем сайте особую папку и назовем ее, скажем, articles. Именно в ней будут находиться файлы Web-страниц, графических изображений, таблиц стилей, архивов и программ. Точнее, не в ней, а во вложенных в нее папках, структуру которых нам нужно будет продумать.

Можно, например, все файлы, относящиеся к определенной статье, помещать в отдельную папку. Такой подход оправдает себя, если у нас будет много статей — будет проще найти все соответствующие определенной статье файлы. Хотя, с другой стороны, появится слишком много вложенных папок, но это меньшее зло по сравнению с огромной "кучей" файлов, в которых нам придется искать те, что соответствуют нужной нам статье.

Если же мы не предполагаем публикацию большого количества статей, то можно создать папки с именами вида htmls, images и downloads. Первая папка будет содержать сами Web-страницы, содержащие тексты статей, вторая — относящиеся к ним графические изображения, третья — программы и архивы.

Можно придумать еще какую-нибудь систему организации файлов и папок, но сейчас мы не будем ломать над ней голову. Подумаем лучше над тем, как реализовать систему управления файлами на Web-сервере и что для этого нам может предложить PHP.

Средства PHP для управления файлами

Здесь мы подробно рассмотрим все средства управления файлами, что поддерживает PHP. А нам понадобятся средства для:

- выбора папки;
- просмотра содержимого выбранной папки, то есть всех файлов и папок, которые в ней содержатся;
- работы с файлами (копирования, перемещения, переименования и удаления);
- работы с папками (создания и удаления).

Ну, и чем нам может помочь PHP?

Выбор папки

Прежде чем начать просмотр содержимого папки, нам нужно сделать ее текущей. Для этого служит встроенная функция `chdir`:

```
chdir(<путь папки, которую нужно сделать текущей>);
```

Эта функция возвращает `true` в случае успешного выполнения операции. Если указанная папка не была сделана текущей (например, она отсутствует), возвращается `false`.

```
chdir($_SERVER["DOCUMENT_ROOT"] . "/uploads/");
```

Это выражение делает текущей папку uploads нашего сайта.

```
chdir($_SERVER["DOCUMENT_ROOT"]);
```

А это выражение делает текущей его корневую папку.

Чтобы узнать, какая папка является в данный момент текущей, нужно вызвать не принимающую аргументов встроенную функцию `getcwd`. Она вернет путь текущей папки в строковом виде.

```
$currentDir = getcwd();
```

Самое интересное, что эта функция всегда возвращает путь в формате файловой системы Unix, то есть папки в возвращенном ей пути всегда разделяются символом прямого слеша. Так происходит, даже если обработчик PHP работает под операционной системой Windows. В некоторых случаях эта особенность может быть очень полезной.

Просмотр содержимого папки

Для просмотра содержимого текущей папки в PHP служат три встроенные функции. Рассмотрим их по порядку.

Функция `opendir` указывает PHP начать просмотр содержимого папки. Путь к папке, содержимое которой нужно просмотреть, передается ей в качестве единственного аргумента в строковом виде. А возвращает эта функция особый идентификатор папки, который обязательно нужно сохранить в какой-либо переменной, так как он нам впоследствии понадобится.

Здесь возникает небольшая сложность. Дело в том, что функция `opendir` принимает путь только в формате Unix (папки должны разделяться символом прямого слеша). Поэтому, если мы напишем такое выражение:

```
$folder = opendir($_SERVER["DOCUMENT_ROOT"]);
```

оно работать не будет, так как элемент с индексом `DOCUMENT_ROOT` встроенного массива `$_SERVER` возвращает путь в формате, который поддерживает файловая система серверного компьютера. И если на серверном компьютере установлена Windows, то путь будет записан в формате Windows (для разделения папок используется символ обратного слеша), и функция `opendir` не сможет его обработать.

Выход из этого положения очень прост — достаточно написать вот такой код:

```
chdir($_SERVER["DOCUMENT_ROOT"]);
$folder = opendir(getcwd());
```

Мы уже знаем, что функция `getcwd` всегда возвращает путь в формате Unix. Даже если мы работаем под Windows.

Приведенный ранее код может быть использован и для того, чтобы преобразовать путь вида /uploads/uploads1/../ к нормальному виду /uploads/. Впоследствии это нам очень пригодится.

Функция `readdir` возвращает имя первого файла или папки, встретившегося в заданной при вызове функции `opendir` папке, в строковом виде. Единственный аргумент, который принимает эта функция, — это сохраненный ранее идентификатор папки, возвращенный функцией `opendir`.

```
$file = readdir($folder);
```

Если вызвать функцию `readdir` второй раз, она вернет имя второго файла или папки и т. д. Это значит, что, вызывая ее раз за разом, мы можем получить имена всех файлов и папок, хранящихся в заданной папке.

```
$i = 0;
while (($file = readdir($folder) !== false) {
  $files[$i] = $file;
  $i++;
}
```

Замечание

Посмотрим на условие, которое мы задали для цикла. Дело в том, что теоретически в какой-либо папке у нас может храниться файл или папка с именем false. А строка `false` при вычислении условия будет преобразована в логическое значение `false`, и цикл прервется раньше времени. Также заметим, что для сравнения возвращенного функцией `readdir` значения с `false` мы использовали оператор строгого неравенства `!==`.

После просмотра содержимого папки обязательно нужно вызвать функцию `closedir`. Эта функция удаляет из памяти компьютера идентификатор папки, переданный ей в качестве аргумента. Она не возвращает результата.

```
closedir($folder);
```

Очень часто бывает нужно, выполнив просмотр папки, вернуться к началу и просмотреть ее заново. Для этого достаточно вызвать функцию `rewinddir`, передав ей в качестве аргумента идентификатор необходимой папки.

```
rewinddir($folder);
```

Получение сведений о файлах и папках

Здесь мы рассмотрим встроенные функции PHP, позволяющие узнать разные данные о файлах и папках.

Функция `is_file` возвращает `true`, если в качестве аргумента ей было передано имя файла.

```
$i = 0;
chdir($_SERVER["DOCUMENT_ROOT"] . "/uploads/);
$folderID = opendir(getcwd());
while (($file = readdir($folderID) !== false) {
  if (is_file($file)) {
    $files[$i] = $file;
    $i++;
  }
}
closedir($folderID);
```

Приведенный ранее сценарий поместит в массив `$files` имена всех файлов, находящихся в папке uploads.

А функция `is_dir`, наоборот, возвращает `true`, если в качестве аргумента ей было передано имя папки.

```
$i = 0;
chdir($_SERVER["DOCUMENT_ROOT"] . "/uploads/);
$folderID = opendir(getcwd());
while (($folder = readdir($folderID) !== false) {
  if (is_dir($folder)) {
    $folders[$i] = $folder;
    $i++;
  }
}
closedir($folderID);
```

После выполнения этого сценария мы получим в массиве `$folders` все папки, хранящиеся в папке uploads.

Функция `basename` принимает в качестве аргумента полный путь к файлу или папке и возвращает только имя.

```
$filename = basename("/uploads/letter.html");
```

После выполнения этого выражения в переменной `$filename` окажется строка `letter.html`.

Функция `dirname` также принимает в качестве аргумента полный путь к файлу или папке, но возвращает только сам путь, без имени.

```
$dir1 = dirname("/uploads/letter.html");
$dir2 = dirname("/image.jpg");
```

После выполнения этих двух выражений в переменной `$dir1` окажется строка `/uploads` (без конечного слеша), а в переменной `$dir2` — строка `.` (точка, обозначающая текущую папку).

Копирование, перемещение, переименование и удаление файлов

Для копирования файлов используется встроенная функция, которая так и называется — `copy`. Вот формат ее вызова:

```
copy(<исходный файл>, <результирующий файл>);
```

Первым аргументом ей передается путь изначального файла, который нужно скопировать, а вторым — путь результирующего файла, который должен быть получен после копирования. Если копирование было успешно выполнено, функция вернет `true`.

```
if (!(copy("/uploads/letter.html", "/articles/article3/letter.html"))) {
  echo "При копировании произошла ошибка.";
}
```

Для перемещения и переименования файлов используется одна и та же функция — `move`.

```
move(<исходный файл>, <результирующий файл>);
```

Здесь первым аргументом также передается путь изначального файла, который нужно переместить или переименовать, а вторым — путь файла, который должен быть получен после перемещения или переименования. Если перемещение или переименование было успешно выполнено, функция вернет `true`.

```
if (!(move("/uploads/image.gif", "/articles/article21/image.gif"))) {
  echo "При перемещении/переименовании произошла ошибка.";
}
```

Замечание

Функцию `move` можно также применять для перемещения и переименования папок.

Для удаления файла нужно вызвать функцию `unlink`. Формат ее вызова очень прост: она принимает путь удаляемого файла и возвращает `true`, если файл был успешно удален.

```
if (!(unlink("/articles/article21/image.gif"))) {
  echo "Проверьте, существует ли этот файл.";
}
```

Создание и удаление папок

Для создания папок используется встроенная функция `mkdir`. В качестве аргумента она принимает строку с именем создаваемой папки и возвращает `true`, если папка была успешно создана.

```
chdir("/articles/");
if (!(mkdir("article4"))) {
  echo "Не удалось создать папку.";
}
```

Чтобы удалить папку, нужно вызвать функцию `rmdir`. Она принимает путь удаляемой папки и возвращает `true`, если папка была успешно удалена.

```
chdir("/articles/");
if (!(rmdir("article34567"))) {
  echo " Проверьте, существует ли эта папка.";
}
```

Внимание

Функция `rmdir` может удалять только пустые, то есть не содержащие файлов папки.

Создание Web-интерфейса для управления файлами

Выяснив все о средствах для управления файлами, предлагаемых PHP, можно приступать к реализации Web-интерфейса. Снова откроем в Dreamweaver страницу Files.php. Именно на ней мы и поместим все необходимые для этого формы.

Просмотр содержимого папки

Первое, что нам нужно сделать, — это средства для просмотра содержимого текущей папки. Для этого мы создадим еще две формы и поместим в них по списку. В одном списке будут отображаться все папки, а в другом — все файлы.

Первую форму, где будут отображаться все папки, мы поместим сразу под созданной ранее формой, предназначенной для отправки файлов на Web-сервер. Из всех параметров этой формы пока зададим только имя — `folders_mgmt`.

Создав эту форму, поставим в нее текстовый курсор, напишем текст `Папка` и поставим пробел. Сразу же после пробела поместим раскрывающийся список и дадим ему имя `folder_name`.

Теперь переключимся в режим отображения кода HTML и поместим в самое начало страницы, после всех сценариев, созданных Dreamweaver, и перед кодом HTML, такой сценарий PHP:

```
<?php
$current_dir = $_SERVER["DOCUMENT_ROOT"] . "/uploads/";
chdir($current_dir);
$current_dir = getcwd();
?>
```

Этот сценарий преобразует путь папки uploads в формат Unix и сохраняет его в переменной `$current_dir`.

Далее найдем код HTML, относящийся к только что созданной нами форме `folder_mgmt`. Вот он:

```
<FORM ACTION="" METHOD="post" NAME="folders_mgmt" ID="folders_mgmt">
  Папка
  <SELECT NAME="folder_name" ID="folder_name">
  </SELECT>
</FORM>
```

Здесь мы видим тег `<SELECT>`, создающий раскрывающийся список `folder_name`. Пока что он не имеет ни одного пункта. (Пункты списка создаются с помощью парного тега `<OPTION>`, внутри которого помещается название этого пункта.)

Давайте впишем сюда сценарии PHP, заполняющие список именами всех папок, хранящихся в текущей папке.

```
<FORM ACTION="" METHOD="post" NAME="folders_mgmt" ID="folders_mgmt">
  Папка
  <SELECT NAME="folder_name" ID="folder_name">
  <?php
  $folderID = opendir($current_dir);
  while (($folder = readdir($folderID)) !== false) {
  if ((is_dir($folder)) && ($folder != ".")) {
  ?>
```

```
    <OPTION><?php echo $folder; ?></OPTION>
  <?php
  } }
  ?>
  </SELECT>
</FORM>
```

Здесь мы просматриваем содержимое папки uploads и заполняем именами всех хранящихся в ней папок раскрывающийся список `folder_name`. Заметим, что мы не включаем в этот список саму просматриваемую папку (обозначается символом точки), чтобы не загромождать его. Также обратим внимание, что мы не вызываем функцию `closedir` — идентификатор папки понадобится нам потом, для вывода списка файлов.

Создав форму со списком папок, переключимся в режим отображения Web-страницы и поместим ниже еще одну форму, в которой будет выводиться раскрывающийся список файлов. Дадим ей имя `files_mgmt`. Введем в ней текст `Файл`, поставим пробел и создадим раскрывающийся список, которому дадим имя `file_name`.

Теперь опять переключимся в режим отображения кода HTML и найдем код, создающий эту форму. Вот он:

```
<FORM ACTION="" METHOD="post" NAME="files_mgmt" ID="files_mgmt">
  Файл
  <SELECT NAME="file_name" ID="file_name">
  </SELECT>
</FORM>
```

Все практически то же самое. И сценарии, которые будут заполнять список `file_name` именами файлов, будут практически такими же.

```
<FORM ACTION="" METHOD="post" NAME="files_mgmt" ID="files_mgmt">
  Файл
  <SELECT NAME="file_name" ID="file_name">
  <?php
  rewinddir($folderID);
  while (($file = readdir($folderID)) !== false) {
  if (is_file($file)) {
  ?>
    <OPTION><?php echo $file; ?></OPTION>
  <?php
  } }
  closedir($folderID);
```

```
  ?>
  </SELECT>
</FORM>
```

Здесь мы вызываем функцию `rewinddir`, чтобы просмотреть содержимое папки сначала. И не забываем в конце вызвать функцию `closedir`, чтобы очистить память серверного компьютера от не нужного нам более идентификатора папки.

Напоследок можем вписать перед формой со списком папок заголовок второго уровня `Папки`, а перед формой со списком файлов — заголовок `Файлы`.

Теперь мы можем сохранить исправленную страницу Files.php, опубликовать ее на Web-сервере и проверить в действии то, что мы сделали. Отправить парочку файлов на Web-сервер и посмотреть — они должны отображаться в раскрывающемся списке файлов формы `files_mgmt`.

Переходы между папками

Просмотреть содержимое одной-единственной папки было не очень сложно. Вот переход от одной папки к другой будет сделать труднее. Но мы справимся!

Первое, что мы должны сделать, — это "переключение" между папкой uploads, куда "складываются" все отправленные на Web-сервер файлы, и папкой, где будут публиковаться авторские статьи (как мы условились ранее, она будет называться articles). Создадим эту папку с помощью панели **Files** и сразу же опубликуем на Web-сервере.

Для перехода между папками uploads и articles мы создадим две формы — по форме на каждую папку — для чего переключимся в режим отображения Web-страницы. Так будет проще.

Сначала создадим новую форму прямо под заголовком `Папки` и перед формой `folders_mgmt` со списком папок. У нее будут такие параметры:

- имя — `uploads_folder`;
- серверная программа, которой будут отправлены данные, — Files.php (то есть эта же самая страница);
- метод отправки данных — `POST`;
- метод кодирования данных — `application/x-www-form-urlencoded`.

Создадим в этой форме скрытое поле с именем `folder_name` и значением `/uploads/` и кнопку отправки данных с именем `submit` и надписью `Перейти в папку uploads`. Все, первая форма готова.

Вторая форма будет точно такой же за следующими исключениями:

- имя формы должно быть `articles_folder`;

- значение скрытого поля должно быть `/articles/`;
- надпись на кнопке — `Перейти в папку articles`.

Теперь переключимся в режим отображения кода HTML и найдем в начале страницы сценарий, устанавливающий текущей папку uploads. Мы сами его создали, поэтому найдем его без труда:

```
<?php
$current_dir = $_SERVER["DOCUMENT_ROOT"] . "/uploads/";
chdir($current_dir);
$current_dir = getcwd();
?>
```

Изменим этот сценарий так, чтобы он выполнял переход на нужную нам папку:

```
<?php
if ((isset($_POST["folder_name"])) && ($_POST["folder_name"] != "")) {
  $current_dir = $_POST["folder_name"];
} else {
  $current_dir = "/uploads/";
}
chdir($_SERVER["DOCUMENT_ROOT"] . $current_dir);
$current_dir = getcwd();
?>
```

Теперь сохраним исправленную страницу, опубликуем ее на Web-сервере и проверим в работе. Нам нужно убедиться, что наши сценарии написаны правильно, чтобы потом ими больше не заниматься.

Следующий шаг — реализовать переход в папку, выбранную в раскрывающемся списке `folder_name`. Этот список имеет то же имя, что и скрытые поля в формах `uploads_folder` и `articles_folder`, так что изменения в коде сценариев будут минимальны (по крайней мере, на первый взгляд).

Для начала зададим для формы `folders_mgmt` недостающие параметры:

- серверная программа, которой будут отправлены данные, — Files.php (эта же самая страница);
- метод отправки данных — `POST`;
- метод кодирования данных — `application/x-www-form-urlencoded`.

Потом поставим в форму `folders_mgmt`, сразу после списка `folder_name` текстовый курсор и нажмем клавишу <Enter>, чтобы создать новый абзац. Создадим в этом абзаце кнопку отправки данных с именем `submit` и надписью `Перейти`. И подумаем, ничего ли мы не забыли...

Давайте предположим, что мы создали в папке articles другую папку, с именем, скажем, private. Теперь нам нужно перейти в эту папку. Мы щелкаем кнопку **Перейти в папку articles** страницы Files.php, потом выбираем в раскрывающемся списке `folder_name` имя нужной нам папки (private) и нажимаем кнопку **Перейти**. Что произойдет? Посмотрим на приведенный ранее сценарий. Обработчик PHP сначала поместит имя выбранной нами папки в переменную `$current_dir`, а потом, встретив выражение

```
chdir($_SERVER["DOCUMENT_ROOT"] . $current_dir);
```

добавит ее к пути корневой папки сайта. А папка private находится вовсе не там, так что приведенное ранее выражение не выполнится!

Если делать все правильно, то нам будет нужно добавить папку private к текущей папке, то есть к папке articles, чтобы получить путь <*путь корневой папки*>/articles/private. Значит, нам как-то нужно отслеживать, щелкнули ли мы по одной из кнопок **Перейти в папку uploads** или **Перейти в папку articles** либо выбрали нужную нам папку в раскрывающемся списке `folder_name`. И соответственно этому формировать путь папки, в которую мы хотим перейти.

Проще всего для этого использовать особое скрытое поле. Давайте создадим его во всех трех формах: `uploads_folder`, `articles_folder` и `folders_mgmt`. У этого поля будет одинаковое имя — `add`, но разные значения. В формах `uploads_folder` и `articles_folder` оно будет иметь значение `0`, а в форме `folders_mgmt` — `1`.

И еще нам будет нужно как-то сохранить путь папки, просматриваемой в данный момент. Для этого используем переменную уровня сессии `curdir`.

И нам останется в очередной раз исправить приведенный ранее сценарий:

```
<?php
if (isset($_SESSION["curdir"])) {
  $current_dir = $_SESSION["curdir"];
} else {
  $current_dir = $_SERVER["DOCUMENT_ROOT"] . "/uploads/";
}
if ((isset($_POST["folder_name"])) && ($_POST["folder_name"] != "")) {
  if ($_POST["add"] == 1) {
    $current_dir = $current_dir . "/" . $_POST["folder_name"];
  } else {
    $current_dir = $_SERVER["DOCUMENT_ROOT"] . $_POST["folder_name"];
  }
}
chdir($current_dir);
$current_dir = getcwd();
```

```
if ((!(strpos($current_dir, "uploads"))) && (!(strpos($current_dir,
↳"articles")))) {
  $current_dir = $_SERVER["DOCUMENT_ROOT"] . "/uploads/";
  chdir($current_dir);
  $current_dir = getcwd();
}
$_SESSION["curdir"] = $current_dir;
?>
```

Все? Еще нет. Существует опасность, что злоумышленник может "пропутешествовать" в корневую папку сайта, а то и дальше, и узнать, какие файлы там хранятся. Давайте добавим в наш сценарий выражение, которое не будет пускать его дальше папок uploads и articles. Далее приведен фрагмент этого сценария с уже внесенными исправлениями (добавленный код выделен полужирным шрифтом):

```
chdir($current_dir);
$current_dir = getcwd();
if ((!(strpos($current_dir, "uploads")) && (!(strpos($current_dir,
↳"articles"))) {
  $current_dir = $_SERVER["DOCUMENT_ROOT"] . "/uploads/";
  chdir($current_dir);
  $current_dir = getcwd();
}
$_SESSION["curdir"] = $current_dir;
```

Встроенная функция `strpos` возвращает позицию строки, переданной вторым аргументом, в строке, переданной первым аргументом. Мы используем эту функцию, чтобы выяснить, присутствует ли в пути просматриваемой папки строки `uploads` и `articles`. Если же этих строк там нет, значит, пользователь вышел "за пределы" одноименных папок и его нужно принудительно вернуть в папку uploads. Пусть знает свое место!

Управление файлами

Для управления файлами мы используем форму `files_mgmt`, в которой уже имеется готовый и нормально работающий список файлов `file_name`. Осталось только добавить другие элементы управления и написать сценарии, которые будут выполнять разные действия над файлами.

Первым же делом зададим для формы `files_mgmt` не установленные при ее создании параметры:

- серверная программа, которой будут отправлены данные, — Files.php;
- метод отправки данных — `POST`;
- метод кодирования данных — `application/x-www-form-urlencoded`.

Для выбора действия, которое необходимо выполнить над нужным файлом, мы создадим в этой форме набор переключателей. Всего таких переключателей будет три: для копирования файла, для его перемещения и переименования, а также для удаления.

Поставим в форму `files_mgmt` текстовый курсор, нажмем клавишу <Enter>, чтобы создать новый абзац, и поместим в него первый переключатель. Чтобы сделать это, выберем пункт **Radio Button** подменю **Form** меню **Form**. Когда переключатель будет создан, щелкнем по нему, чтобы выделить, и посмотрим на редактор свойств (рис. 15.1).

Рис. 15.1. Вид редактора свойств при выделенном переключателе

С помощью редактора свойств мы зададим такие параметры созданного нами переключателя:

- имя (поле ввода **Radio Button**) — `file_op`;
- значение, которое будет отправлено серверной программе, если переключатель включен (поле ввода **Checked value**), — `copy`;
- изначальное состояние — включен (переключатель **Checked** набора **Initial state**).

Сразу же за переключателем напишем текст `Скопировать`. Далее опять нажмем клавишу <Enter> и в новом абзаце создадим еще один переключатель с параметрами:

- имя — `file_op`;
- значение, которое будет отправлено серверной программе, если переключатель включен, — `move`;
- изначальное состояние — выключен (переключатель **Unchecked** набора **Initial state**).

Введем текст `Переместить/Переименовать`, создадим еще один новый абзац и в нем — третий переключатель. Его параметры будут такие же, как у предыдущего, за исключением значения — `delete`. После него введем текст `Удалить` и создадим еще один новый абзац.

Прежде чем скопировать, переместить или переименовать файл, обработчик PHP должен принять от пользователя новый путь файла. Для него мы создадим в новом абзаце поле ввода и дадим ему имя `new_file`.

Последнее, что нам осталось создать в форме `files_mgmt`, — это кнопку отправки данных. Дадим ей имя `submit` и надпись `Выполнить`.

Все, форма готова. Настало время написать сценарий PHP, который и будет управлять файлами. Вот он:

```
<?php
if (isset($_POST["file_op"])) {
  // Формируем имя изначального файла
  $old_file_name = $current_dir . "/" . $_POST["file_name"];
  // Формируем имя результирующего файла
  if ((isset($_POST["new_file"])) && ($_POST["new_file"] != "")) {
    $new_file_name = $_SERVER["DOCUMENT_ROOT"] . "/articles" .
    ↳$_POST["new_file"];
  }
  switch ($_POST["file_op"]) {
    case "copy":
      copy($old_file_name, $new_file_name);
      break;
    case "move":
      rename($old_file_name, $new_file_name);
      break;
    case "delete":
      unlink($old_file_name);
  }
}
?>
```

Помещаем этот сценарий в код страницы Files.php сразу после сценария, выполняющего переходы между папками, — это важно! Ведь именно в сценарии, выполняющем переходы между папками, объявлена переменная `$current_dir`, которую мы здесь используем. После этого можно будет сохранить почти готовую Web-страницу, опубликовать ее на Web-сервере и проверить в работе.

Создание и удаление папок

Для управления папками мы используем форму `folders_mgmt`, в которой уже имеется готовый и нормально работающий список папок `folder_name`. Нам придется только добавить новые элементы управления, написать сценарии, которые будут управлять папками, и, конечно, изменить надпись на кнопке.

Для выбора действия, которое нужно выполнить над выбранной в списке папкой, мы создадим набор переключателей. Переключателей в этом наборе

должно быть три: для перехода на выбранную папку, для создания в ней новой папки и для удаления выбранной папки.

Итак, поставим сразу после списка текстовый курсор, создадим новый абзац, а в нем — переключатель с такими параметрами:

- имя — `folder_op`;
- значение, которое будет отправлено серверной программе, если переключатель включен, — `open`;
- изначальное состояние — включен.

После этого переключателя напишем текст `Переход` и создадим еще один новый абзац. В него мы поместим второй переключатель, параметры которого будут такими:

- имя — `folder_op`;
- значение, которое будет отправлено серверной программе, если переключатель включен, — `create`;
- изначальное состояние — выключен.

Далее введем текст `Создать`, создадим третий абзац с еще одним переключателем. Параметры этого, третьего по счету, переключателя будут такими же, как у второго, за исключением значения. Оно должно быть равно `delete`. После этого переключателя введем текст `Удалить` и создадим очередной новый абзац.

Для создания новой папки обработчик PHP должен принять от пользователя ее имя. Для ввода этого имени мы поместим в новом абзаце поле ввода и дадим ему имя `new_folder`.

Последнее, что нам осталось создать в форме `folders_mgmt`, — это изменить надпись на кнопке отправки данных на `Выполнить`.

Теперь можно заняться сценарием PHP, выполняющим разные действия с выбранной в списке папкой. Мы в очередной раз исправим сценарий, осуществляющий переход между папками, вот этот (часть кода пропущена):

```
<?php
if (isset($_SESSION["curdir"])) {
  $current_dir = $_SESSION["curdir"];
} else {
  $current_dir = $_SERVER["DOCUMENT_ROOT"] . "/uploads/";
}
. . .
$_SESSION["curdir"] = $current_dir;
?>
```

Измененный сценарий будет выглядеть так:

```
<?php
if (isset($_SESSION["curdir"])) {
  $current_dir = $_SESSION["curdir"];
} else {
  $current_dir = $_SERVER["DOCUMENT_ROOT"] . "/uploads/";
}
if (isset($_POST["folder_op"])) {
  switch ($_POST["folder_op"]) {
    case "open":
      if ($_POST["add"] == 1) {
        $current_dir = $current_dir . "/" . $_POST["folder_name"];
      } else {
        $current_dir = $_SERVER["DOCUMENT_ROOT"] .
        ↳$_POST["folder_name"];
      }
      chdir($current_dir);
      $current_dir = getcwd();
      if ((!(strpos($current_dir, "uploads"))) && (!(strpos($current_dir,
      ↳"articles")))) {
        $current_dir = $_SERVER["DOCUMENT_ROOT"] . "/uploads/";
        chdir($current_dir);
        $current_dir = getcwd();
      }
      break;
    case "create" :
      $new_folder_path = $current_dir . "/" . $_POST["new_folder"];
      mkdir($new_folder_path);
      break;
    case "delete" :
      $folder_to_delete = $current_dir . "/" . $_POST["folder_name"];
      rmdir($folder_to_delete);
  }
}
chdir($current_dir);
$current_dir = getcwd();
$_SESSION["curdir"] = $current_dir;
?>
```

Теперь, чтобы этот сценарий нормально работал, нам будет нужно создать скрытые поля в формах `uploads_folder` и `articles_folder`. Они будут имитировать включение переключателя `Перейти`, которого в этих формах нет. Дадим им одинаковое имя `folder_op` и одинаковое значение `open`, и все будет нормально.

Что ж, страница Files.php — наш Web-интерфейс для управления файлами — готова. Теперь можно сохранить ее, опубликовать на Web-сервере и устроить ей самую серьезную, последнюю проверку. И, если мы нигде не допустили ошибок, страница должна ее пройти!

Что дальше?

Да, с Web-интерфейсом для управления файлами пришлось повозиться... Но зато что мы получили в результате!

Последнее, что мы запланировали сделать, — это свою собственную службу почтовых рассылок. Сделаем так, чтобы все желающие получали информацию обо всех пополнениях нашей коллекции статей и файлов по электронной почте. А неисчерпаемый PHP нам в этом поможет.

Глава 16

Организация почтовой рассылки

Что ж, почти все "вкусности", что мы грозились создать на своем сайте, готовы и работают. Мы сделали нормальные, удобочитаемые сообщения об ошибках, возможность кое-каких настроек под конкретного посетителя и Web-интерфейс для отправки файлов на Web-сервер и управления ими. Это повод для гордости — не все сайты, подобные нашему, могут похвастаться такой заботой о посетителе.

Осталось совсем немного — сделать собственную службу почтовых рассылок. Пусть все желающие получают по электронной почте сведения обо всех новых статьях и файлах, добавленных в базу данных. Таким образом они смогут узнать, появилось ли на нашем сайте что-то интересное для них.

Вообще, служба почтовых рассылок — сервис довольно популярный. Его предоставляют очень и очень многие сайты самой различной направленности. Частенько на главной странице сайта можно увидеть гиперссылку, приглашающую посетить страницу подписки на рассылку, а то и прямо поле ввода почтового адреса и кнопку **Подписаться**. Нам тоже не стоит отставать от времени.

PHP — платформа очень богатая разными возможностями (бóльшая часть которых, надо признать, используется довольно редко). В числе прочего, она предоставляет встроенные средства для организации почтовых рассылок, а именно для отправки писем через любой почтовый сервер, находящийся в пределах досягаемости. Мы рассмотрим самый простой способ отправки писем, для реализации которого нам понадобится только почтовый ящик на любом сервере e-mail. А этот ящик у нас уже есть, ведь сейчас без электронной почты никуда.

Замечание

Dreamweaver поддерживает несколько способов отправки электронных писем, но мы рассмотрим только один. Остальные способы требуют наличия на серверном компьютере специальных программ, которые нужно где-то искать, устанавливать и, главное, настраивать. А на бесплатных Web-серверах этих программ может не оказаться вообще.

Введение в почтовые рассылки

Сначала, как обычно, — небольшое теоретическое вступление. Мы поговорим о почтовых рассылках вообще и о способах их реализации в частности.

Что такое почтовая рассылка

Почтовая рассылка — это массовая рассылка электронных писем по заранее составленному списку адресов (так называемому *листу рассылки*). Почтовые рассылки очень широко используются для уведомления клиентов какой-либо компании или сайта о различных событиях, обновлениях и рекламных акциях, для доставки подписчикам новостей и статей на разные темы и, в конце концов, для рекламы.

Все почтовые рассылки делятся на две большие группы. Сейчас мы их рассмотрим.

Первая группа — это *санкционированные*, или *легальные*, почтовые рассылки. Будущий получатель таких рассылок сам подписывается на них, войдя на особую страницу сайта, который их рассылает. Впоследствии, если он расхочет получать какую-либо рассылку, то может легко от нее отписаться, для чего достаточно снова посетить рассылающий их сайт. Очень часто для подписки на рассылку и отписки от нее также можно послать особым образом написанное электронное письмо на специальный почтовый адрес.

А вот вторая группа почтовых рассылок — настоящий бич современного Интернета. Это *несанкционированные почтовые рассылки*, или *спам*. От легальных рассылок они отличаются тем, что для попадания в лист рассылки от получателя вообще не требуется никаких телодвижений — достаточно опубликовать свой адрес на Web-странице, в Web-форуме, в группе новостей или ввести его в форму на каком-нибудь подозрительном сайте. Отписаться от несанкционированной рассылки практически невозможно.

Как правило, спам используется для рекламы различных сомнительных предложений и всевозможных афер. Пожалуй, всем известны письма, рассылаемые финансовыми мошенниками (так называемые "нигерийские письма", так как рассылать их начали мошенники из Нигерии). Также много нервов интернетчикам испортила эпопея с "Центром американского английского", письма от которого регулярно заваливали почтовые ящики около года назад. В последнее время спам часто используется для массового распространения вирусов и шпионских программ, что делает его не только надоедливым, но и опасным.

Для борьбы со спамом используются разные пути. Во-первых, законодатели разных стран принимают законы по борьбе со спамом. (Совсем недавно такой закон был принят и в России.) Во-вторых, практикуется судебное преследование спамеров. В-третьих, ведется активная антиспамовая пропаганда,

разъясняющая, что спам — не лучший способ рекламировать товары и услуги. В-четвертых, создаются программы, выявляющие и удаляющие присланные спамерами письма, зачастую прямо на почтовом сервере; это, пожалуй, самый эффективный путь борьбы с этой напастью.

И еще. Не следует путать легальные почтовые рассылки и спам. Легальные рассылки — очень полезный инструмент доставки информации клиентам, к тому же, они честны с получателями. Отписаться от легальной рассылки очень просто (как и подписаться). Спам же — безусловное зло; он отнимает у нас время и деньги (подключение к Интернету — штука недешевая), да еще и грозит вирусами и программами-шпионами. А избавиться от него очень трудно.

Но хватит о плохом. Давайте лучше поговорим, какими способами осуществляются эти самые почтовые рассылки и какой из них стоит выбрать нам.

Как осуществляются почтовые рассылки

Осуществить почтовую рассылку можно двумя способами. И тот, и другой применяются одинаково часто.

Первый способ — самый простой. Существуют особые сайты — *службы рассылок*, — предоставляющие средства для рассылки писем по списку адресов. Они берут на себя практически все задачи по организации рассылки: дают всем желающим возможность подписаться на нужные им рассылки и отписаться от ненужных, хранят листы рассылки и, собственно, рассылают письма. От нас требуется только зарегистрироваться на таком сайте, поместить на страницу своего сайта форму для подписки на рассылку и регулярно готовить очередные выпуски рассылки.

Таких служб рассылок существует довольно много. В Рунете больше всего популярны две из них: Subscribe.ru и Maillist.ru. Subscribe.ru — старейшая русская служба рассылок, насчитывающая миллионы подписчиков и тысячи рассылок. (Большая часть из этих рассылок, правда, уже "мертва". Обычное дело: ведущий, открывая рассылку, полон наполеоновских планов, но жизнь вносит свои коррективы, и времени на ведение рассылки не остается.) Maillist.ru помоложе предоставляет для ведущих и подписчиков меньше полезных инструментов и поэтому менее популярен.

Достоинство первого способа создать свою рассылку — простота. В самом деле, что может быть проще, чем заполнить на сайте службы рассылок небольшую форму и регулярно создавать новые выпуски рассылки. Недостатков два, и оба могут оказаться критичными. Во-первых, служба рассылки вставляет в рассылаемые подписчикам письма свою рекламу (а без этого нельзя — бесплатные ресурсы Интернета живут практически только рекламой). Во-вторых и в-главных, нам придется самим создавать новые выпуски и самим отправлять их службе рассылок, а автоматизировать этот процесс вряд ли получится.

Второй способ сложнее в реализации, но предоставляет нам больше возможностей. Мы сами можем написать программу, выполняющую рассылку писем по списку. Да, эти письма не будут содержать посторонней рекламы, и их составление можно будет автоматизировать, но написать программу для почтовых рассылок — весьма нелегкая задача даже для опытного программиста, знакомого с принципом действия почтовых серверов.

Но, как уже говорилось ранее, у нас есть под рукой PHP — замечательная платформа написания серверных Web-страниц. PHP имеет встроенные средства для организации почтовых рассылок, воспользоваться которыми совсем не сложно. По крайней мере, проще, чем писать все нужные программы самим "с нуля".

Реализация службы рассылки на PHP

Решено! Мы пишем свою службу почтовых рассылок на PHP. И сделаем ее в виде особой серверной Web-страницы.

А сейчас давайте подумаем, как ее лучше реализовать. Нам придется решить несколько проблем еще до того, как мы начнем писать код PHP.

Два способа реализовать службу рассылок на PHP

Когда мы будем писать службу рассылок, нам придется решить следующие проблемы:

- принять от желающего получать нашу рассылку почтовый адрес и сохранить его в листе рассылки;
- сформировать очередной выпуск рассылки;
- выбрать из листа рассылки все почтовые адреса;
- отправить по этим адресам сформированный ранее выпуск рассылки;
- реализовать отписку от рассылки.

Собственно, мы уже знаем, как решить большинство этих проблем. Мы сохраним лист рассылки в особой таблице нашей базы данных `site`, которую назовем, скажем, `maillist`. Принять от желающего почтовый адрес тоже несложно — достаточно создать соответствующую форму на одной из страниц нашего сайта. Точно так же можно сделать отписку от рассылки: посетитель, не желающий больше получать нашу рассылку, вводит свой почтовый адрес в другой форме (форме отписки), и сценарий PHP удаляет соответствующую запись из таблицы `maillist`. Ну, а выбрать все адреса из этой таблицы совсем просто.

Проблема в другом — как выбрать статьи и файлы для формирования письма. Точнее, как отделить "новые" записи таблицы `items` от "старых". Здесь мы можем пойти двумя путями.

Самый простой путь — проверить дату добавления данной статьи или файла. Это значение хранится в поле `added`, так что трудностей вроде бы быть не должно. Мы можем сохранить где-нибудь дату последней проверки (лучше всего ее сохранить в единственном поле единственной записи отдельной таблицы) и, если статья или файл были добавлены позже этой даты, включить ее в письмо.

Но тут нас подстерегает проблема. Предположим, что мы выполнили рассылку писем с обновлениями и сохранили сегодняшнюю дату в таблице. Далее в этот же день мы добавляем в базу данных еще несколько статей или файлов и сразу же пытаемся выполнить рассылку еще раз. Сценарий сравнит даты добавления этих статей (файлов) с сохраненной и, поскольку даты равны, посчитает, что сведения о них уже были отправлены подписчикам. И рассылка выполнена не будет.

Выходом из этого положения может быть хранение не одной даты добавления, а даты и времени, благо MySQL поддерживает такой тип данных. Для этого нам придется исправить структуру таблицы `items`, поменяв тип поля `added`. Но такое может не пройти безболезненно: вполне возможно, что нам придется переписывать все сценарии, чтобы они обрабатывали значение этого поля как дату и время. Наверное, лучше было бы изначально создавать таблицу `items` с таким предположением, и мы будем иметь это в виду.

Другой путь заключается в том, чтобы добавить в таблицу `items` новое поле логического типа с именем, например, `isnew`. Значение по умолчанию этого поля мы зададим равным `true`. Тогда сценарий, выполняющий рассылку, будет работать так:

- выбираем все записи таблицы `items`, значение поля `isnew` которых равно `true`;
- на основе этих записей формируем письмо для рассылки и рассылаем его;
- присваиваем полю `isnew` всех этих записей значение `false`.

Тогда нам не придется хранить где-то дату выполнения последней рассылки — нам будет достаточно только проверить, имеет ли поле `isnew` хоть одной записи значение `true`. Если такая запись или такие записи нашлись, они считаются "новыми", и сведения о них помещаются в последний выпуск рассылки. После этого полю `isnew` всех этих записей мы присваиваем значение `false` — фактически делаем их "старыми".

Давайте так и поступим. Запустим сервер MySQL, откроем какую-либо клиентскую программу для работы с ним и добавим в таблицу `items` поле `isnew` логического типа. Значение по умолчанию этого поля будет `true`, как мы уже договорились. И создадим на основе этого поля индекс, назвав его так же, как и поле, — `isnew`.

Еще нам понадобится таблица, где будут храниться почтовые адреса подписчиков нашей рассылки. Структура этой таблицы приведена в табл. 16.1. Дадим ей имя `maillist`, как договорились ранее.

Таблица 16.1. *Структура таблицы* `maillist` *(листа рассылки)*

Имя поля	Описание поля	Тип данных поля	Описание типа данных
`email`	Почтовый адрес	`VARCHAR(40)`	Строки длиной до 40 символов

Да-да, нам вполне хватит одного-единственного поля. Наша служба рассылки будет совсем простенькой, без всяких выдумок. А выдумки мы добавим потом, когда наберемся опыта.

Последнее, что мы сделаем, — это создадим ключевой индекс на основе поля `email`. Он сильно ускорит обработку таблицы.

Ну вот, все подготовительные действия мы закончили. Теперь можно начинать рассмотрение средств PHP для отправки почты.

Средства PHP для отправки почты

PHP может отправить наше электронное письмо двумя разными способами. (Здесь имеется в виду — без использования дополнительных компонентов.) Мы рассмотрим их по очереди.

Отправка почты через собственный почтовый сервер

Самый простой, но, в тоже время, самый хлопотный способ отправки почты средствами PHP — это использование собственного почтового сервера. Здесь слово "собственный" обозначает не "встроенный", а "работающий совместно". Такой почтовый сервер устанавливается на том же серверном компьютере, что и обработчик PHP, и настраивается на совместную с ним работу.

Замечание

Процесс настройки сервера почты и обработчика PHP для совместной работы описан в документации по PHP.

У такого способа отправки почты два преимущества. Во-первых, для отправки электронного письма достаточно вызвать всего одну функцию — `mail`. Во-вторых, иметь собственный почтовый сервер значительно удобнее, чем обращаться к услугам постороннего. Мы можем создать на нем несколько почтовых ящиков, скажем, для создателя сайта (нас), для каждого администратора и ведущего и "общий" ящик для контактов. Да и выполнять

массовую почтовую рассылку с собственного сервера проще — никто не примет нас за спамеров и не отключит наш почтовый ящик.

Недостаток всего один — нам понадобится установить программу почтового сервера и соответственно настроить ее и обработчик PHP. Да, это не очень сложно, но весь вопрос в другом: имеем ли мы доступ к серверному компьютеру, на котором размещен наш сайт? На своем собственном компьютере мы можем творить, что захотим, но если наш сайт опубликован на бесплатном Web-сервере, то прости прощай, собственная почта...

Что касается функции `mail`, то вот формат ее вызова:

```
mail(<адрес получателя>, <тема письма>, <тело письма>
↳[, <дополнительные параметры заголовка письма>]);
```

Итак, всего четыре аргумента. И все они имеют строковый тип. Рассмотрим их по порядку.

Первый аргумент задает почтовый адрес получателя письма. С ним все понятно.

Второй аргумент задает *тему письма* — особую строку с краткими сведениями о содержимом письма. В программах почтовых клиентов тема вводится в особое поле ввода, находящееся над областью редактирования самого содержимого письма и имеющее название **Тема** или **Subject**.

Третий аргумент — это само содержимое письма, или его *тело*.

А вот о четвертом, необязательном, аргументе нужно поговорить подробнее. Он задает так называемые дополнительные параметры заголовка электронного письма, без которых во многих случаях просто не обойтись.

Начнем с того, что каждое электронное письмо состоит из двух частей, следующих одна за другой. Вторая часть — это тело, из-за которого это письмо, собственно, и было отправлено. А вот первая часть — это особый набор параметров, называемый *заголовком письма* и несущий сведения об адресах отправителя и получателя, теме письма и текстовой кодировке, в которой это письмо было набрано.

Давайте рассмотрим для примера вот такое коротенькое письмо:

```
From: someuser@someserver.ru
To: otheruser@otherserver.ru
Content-Type: text/plain; charset=windows-1251
Subject: Test!!!

Это всего лишь тестовое письмо. Не пугайтесь!
.
```

Первая и вторая строка задают соответственно почтовые адреса отправителя и получателя. Третья строка задает текстовую кодировку, в которой набрано

письмо. Четвертая строка — это тема. А все эти четыре строки составляют заголовок.

После заголовка обязательно должна идти пустая строка. Она отделяет заголовок от тела письма. А заканчиваться тело (да и все письмо) должно символом точки.

Так вот, о заголовке. Первые три строки формируются самим PHP. PHP автоматически подставляет почтовый адрес отправителя, а почтовый адрес получателя и тема берутся из первых двух аргументов функции `mail`. Значение третьего аргумента подставляется в качестве тела письма.

Но текстовую кодировку PHP сам задавать не умеет. Нам придется самим передать нужную строку четвертым аргументом функции `mail`. Вот так:

```
mail("otheruser@otherserver.ru", "Test!!!",
↳"Это всего лишь тестовое письмо. Не пугайтесь!",
↳"Content-Type: text/plain; charset=windows-1251");
```

Осталось только сказать, что функция `mail` возвращает значение `true` в случае успешной отправки письма.

Прямая отправка письма на почтовый сервер

Для тех, кто не имеет собственного почтового сервера, PHP предлагает другой способ отправки электронных писем — напрямую на любой доступный почтовый сервер. Мы просто соединяемся с почтовым сервером и посылаем ему команды на отправку письма. Конечно, это немного сложнее, и кода придется написать заметно больше, но во многих случаях это единственный выход.

Предположим, что мы имеем свой почтовый ящик `php` на популярном бесплатном сервере Mail.ru (**http://www.mail.ru**). И с этого ящика нам нужно отправить какое-либо письмо.

Итак, сначала нам нужно установить соединение с почтовым сервером. Для этого мы используем встроенную функцию `fsockopen`. Вот формат ее вызова:

```
fsockopen(<интернет-адрес сервера>, <номер порта TCP/IP>);
```

Мы передаем функции `fsockopen` интернет-адрес почтового сервера, с которым нужно соединиться, и номер порта TCP/IP. Протокол SMTP (Simple Mail Transferring Protocol, простой протокол передачи почты), по которому серверу передается почта, использует порт 25.

Если соединение нормально установлено, функция `fsockopen` вернет *идентификатор соединения*. Его нам будет нужно сохранить в какой-либо переменной, поскольку он нам потом понадобится. Если соединение не было установлено, возвращается `false`.

```
$connection = fsockopen("smtp.mail.ru", 25);
```

Сервер приема почты по протоколу SMTP портала Mail.ru находится по адресу **smtp.mail.ru**.

Установив соединение с сервером, мы можем послать ему первую команду. Для этого используем встроенную функцию `fputs`.

```
fputs(<идентификатор соединения>, <строка, которую нужно передать>);
```

Заметим, что функция `fputs` вернет количество переданных байт или `false`, если ничего передать не удалось.

```
fputs($connection, "HELO php\r\n");
```

Это выражение посылает серверу команду `HELO php`, завершающуюся символами возврата каретки и перевода строки `\r\n` (это обязательно!). На языке протокола SMTP эта команда означает что-то вроде "Привет, сервер! Я — пользователь почтового ящика `php`".

```
fputs($connection, "MAIL FROM: php@mail.ru\r\n");
```

Команда, переданная этим выражением, задает адрес отправителя.

```
fputs($connection, "RCPT TO: otheruser@otherserver.ru\r\n");
```

А это выражение посылает серверу команду, содержащую имя получателя.

```
fputs($connection, "DATA\r\n");
```

Команда `DATA` указывает серверу, что сейчас мы передадим ему само письмо.

```
fputs($connection, "Content-Type: text/plain; charset=windows-1251\r\n");
fputs($connection, "From: php@mail.ru\r\n");
fputs($connection, "To: otheruser@otherserver.ru\r\n");
fputs($connection, "Subject: Test!!!\r\n");
fputs($connection, "\r\n");
```

Передаем заголовок письма и пустую строку, которая отделяет его от тела письма.

```
fputs($connection, "Это всего лишь тестовое письмо. Не пугайтесь!\r\n");
```

Это выражение передает серверу тело письма.

```
fputs($connection, ".\r\n");
```

Обязательно завершаем тело письма точкой.

```
fputs($connection, "RSET\r\n");
```

Команда `RSET` говорит серверу "Пока! До новых встреч".

Напоследок мы разорвем соединение с сервером, вызвав встроенную функцию `fclose`. Она принимает единственный аргумент — идентификатор разрываемого соединения — и возвращает `true`, если соединение было успешно разорвано.

```
fclose($connection);
```

Вот и все! Наше письмо отправлено.

Как отправить письмо сразу по нескольким адресам

Последняя техническая проблема, которую нам нужно решить, — как отправить одно и то же письмо сразу по нескольким адресам. Решить ее можно несколькими способами.

Проще всего послать нужное письмо несколько раз, каждый раз — новому адресату. Но это выход только на самый крайний случай. Во-первых, одновременная отправка множества писем сильно перегрузит почтовый сервер, а если мы будем и дальше так поступать, нас, чего доброго, вообще с этого сервера попросят. Во-вторых, это как-то слишком уж прямолинейно — есть и более изящные способы.

Еще можно записать всех адресатов в строку "To:" заголовка одного-единственного письма, разделив их запятой или точкой с запятой:

```
To: user1@server1.ru, user2@server2.ru, user3@server3.ru
```

Почтовый сервер, прочитав эту строку, сам отправит это письмо всем перечисленным в ней адресатам.

Однако здесь возникает вот такая проблема. Каждый получатель нашей рассылки, приняв очередное письмо, сможет прочитать в его заголовке (в строке `To:`, если быть точным) почтовые адреса всех подписчиков. А это не очень хорошо в плане конфиденциальности.

Выход из этого положения весьма прост. Вместо строки `To:` для перечисления адресатов письма нужно использовать строку `Bcc:`, которая задает адреса так называемых "слепых копий" и отличается от строки `To:` тем, что в ней всегда стоит адрес только данного конкретного получателя письма. То есть если мы разошлем письмо, содержащее в заголовке строку

```
Bcc: user1@server1.ru, user2@server2.ru, user3@server3.ru
```

то подписчик с адресом **user2@server2.ru**, получив письмо, увидит в его заголовке только

```
Bcc: user2@server2.ru
```

Остальные адресаты будут удалены из строки `Bcc:` самим почтовым сервером.

И еще. Если мы собираемся рассылать одно и то же письмо нескольким адресатам, мы должны передать все их адреса почтовому серверу в командах `RCPT TO:`

```
fputs($connection, "RCPT TO: user1@server1.ru\r\n");
fputs($connection, "RCPT TO: user2@server2.ru\r\n");
fputs($connection, "RCPT TO: user3@server3.ru\r\n");
```

Вот и все. Остальные сведения о том, куда и как рассылать письмо, почтовый сервер получит из его заголовка.

Создание соответствующих Web-страниц

Ну что ж, все проблемы мы решили. Значит, теперь можно приступать к созданию серверных Web-страниц, которые будут выполнять регистрацию нового подписчика (подписку), рассылку писем и — для тех, кто хочет нас покинуть — разрегистрацию (отписку).

Страница регистрации нового подписчика

Все театры мира начинаются с вешалок, а все службы рассылок мира — со страниц подписки. Поэтому и мы начнем создание своей службы рассылок со страницы подписки, на которой желающий получать письма обо всех обновлениях на нашем сайте сможет ввести свой почтовый адрес.

Создадим в Dreamweaver новую пустую страницу PHP, дадим ей название `Подписка`, такой же заголовок и введем какой-либо поясняющий текст. После этого сохраним страницу под именем Subscribe.php в корневой папке локальной копии нашего сайта.

Теперь создадим форму для ввода почтового адреса и дадим ей имя `subscribe`. В эту форму поместим поле ввода с именем `email` и кнопку отправки данных с именем `submit` и надписью `Подписаться`.

Подписка на нашу рассылку будет заключаться в том, что введенный посетителем адрес будет занесен в новую запись таблицы `maillist` базы данных `site`. Поэтому нам будет нужно создать серверное поведение **Insert Record**, выполняющее добавление новой записи в таблицу. Создадим ее — мы уже знаем, как это делается, из *главы 9*.

Один момент — когда мы будем вводить параметры серверного поведения **Insert Record** в одноименное диалоговое окно, то в качестве страницы, на которую будет выполнен переход в случае успешного добавления записи, введем страницу Subscribe_ok.htm, которую мы создадим потом. Эта страница будет содержать текст, говорящий о том, что подписка успешно выполнена. Таким образом мы дадим посетителю знать, что все прошло нормально.

В принципе, страница подписки готова, и ее можно протестировать. Но давайте подождем немного и подумаем.

Может случиться так, что какой-либо посетитель зайдет на наш сайт и подпишется на рассылку. Его почтовый адрес попадет в таблицу `maillist`. Через некоторое время этот же посетитель снова зайдет на наш сайт и снова попытается подписаться на рассылку. На основе поля `email` таблицы `maillist` создан ключевой индекс, значит, это поле может содержать только уникальные в пределах таблицы значения. И при попытке добавить в таблицу новую запись с тем же значением поля `email` произойдет ошибка.

Конечно, мы можем как-то перехватить эту ошибку и выдать незадачливому посетителю предупреждающий текст. Но давайте сделаем по-другому. А именно

создадим серверное поведение, которое перед добавлением записи будет проверять, не хранится ли уже в таблице этот адрес. Это серверное поведение называется **Check New Username**.

Внимание

Серверное поведение **Check New Username** может быть создано только в том случае, если на странице уже имеется серверное поведение **Insert Record**.

Выведем на экран панель **Server Behaviors**. Нажмем кнопку со знаком плюс и выберем в подменю **User Authentication** появившегося на экране меню пункт **Check New Username**. На экране появится небольшое диалоговое окно **Check New Username** (рис. 16.1).

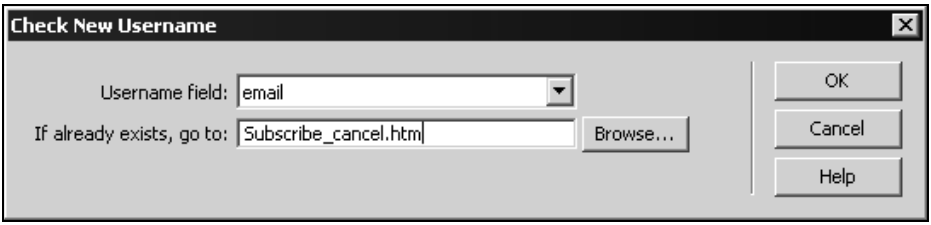


Рис. 16.1. Диалоговое окно **Check New Username**

В раскрывающемся списке **Username field** выберем поле ввода, в которое должен вводиться почтовый адрес. Это поле `email`.

В поле ввода **If already exists, go to** заносится имя страницы, на которую будет выполнен переход, если введенный почтовый адрес уже есть в таблице `maillist`. Введем туда имя пока не существующей страницы Subscribe_cancel.htm, которую создадим потом. Также мы можем нажать кнопку **Browse** и выбрать нужную страницу в диалоговом окне открытия файла Dreamweaver, но только если такая страница у нас уже есть.

Задав нужные данные, нажмем кнопку **OK**. Серверное поведение **Check New Username** будет создано.

Код PHP, формируемый при этом Dreamweaver, очень прост. В таблице `maillist` выполняется поиск записи, значение поля `email` которой равно введенному адресу. Если такая запись нашлась, осуществляется переход на заданную страницу. Если же в таблице еще нет такого адреса, выполняется сценарий, соответствующий серверному поведению **Insert Record**.

Теперь нам осталось создать вспомогательные страницы Subscribe_ok.htm и Subscribe_cancel.htm. Это обычные Web-страницы, и мы создадим их без труда. Введем в них соответствующий текст, объявляющий, соответственно, об успешной подписке на рассылку или о том, что такой адрес уже подписан. Сохраним их в корневой папке сайта, как и страницу Subscribe.php.

Последнее, что мы сделаем, — поместим гиперссылку, указывающую на готовую страницу подписки, на главную страницу нашего сайта. Создадим заголовок `Подписка`, введем под ним какой-либо поясняющий текст и поместим там же гиперссылку на страницу Subscribe.php.

Теперь сохраним все вновь созданные и исправленные страницы, опубликуем их на Web-сервере и проверим "в бою". Сначала попытаемся сами подписаться на нашу рассылку и проверим, появилась ли в таблице `maillist` запись с нашим адресом. Потом попробуем подписаться еще раз — серверное поведение **Check New Username** должно предупредить нас, что такой адрес уже подписан.

Страница выполнения рассылки

Что ж, создать страницу подписки было несложно — нам во многом помог Dreamweaver. Сейчас же мы приступим к созданию страницы выполнения рассылки, и нам придется полагаться только на собственные силы.

Создадим новую пустую страницу PHP, дадим ей название `Выполнение рассылки`, такой же заголовок и введем поясняющий текст. После этого сохраним страницу под именем Subscribe_run.php в папке Admin нашего сайта (ведь, как мы договорились ранее, выполнять рассылку могут только администраторы и ведущие).

Давайте сделаем так, чтобы рассылка писем началась после нажатия особой кнопки. А если так, то нам понадобится форма. Создадим эту форму и дадим ей такие параметры:

- имя — `subscribe_run`;
- серверная программа, которой будет отправлен файл, — Subscribe_run.php (эта же самая страница);
- метод отправки данных — `POST`;
- метод кодирования данных — `application/x-www-form-urlencoded`.

Теперь поместим в эту форму скрытое поле с именем `subscribe_run` и значением `yes`. Это поле мы используем для проверки, была ли нажата кнопка запуска рассылки или эта страница загрузилась в первый раз.

Ну и понадобится также кнопка запуска рассылки. Это будет кнопка отправки данных с неизменным именем **submit** и надписью `Выполнить`.

Чтобы дать доступ к странице Subscribe_run.php только администраторам и ведущим, создадим на ней серверное поведение **Restrict Access To Page**. Как это сделать, было описано в *главе 11*.

А теперь настал самый трудный этап работы. Сейчас мы переключимся в режим отображения кода HTML и начнем писать сценарий, выполняющий рассылку писем. Этот сценарий мы вставим после сценария, соответст-

вующего серверному поведению **Restrict Access To Page**, и перед кодом HTML. Поскольку он весьма сложен, мы рассмотрим его по частям.

```
<?php
if ((isset($_POST['subscribe_run'])) &&
↳($_POST['subscribe_run'] == "yes")) {
```

Проверяем, передан ли этой странице методом `POST` аргумент `subscribe_run`. Если передан, значит, страница была загружена после нажатия кнопки запуска рассылки и нужно выполнить весь последующий код.

```
  $rcpt_to = "";
  $bcc = "";
  $body = "";
```

Подготавливаем переменные, в которых будет храниться список адресов для команды `RCPT TO:` почтового сервера, для строки `Bcc:` заголовок письма и тело письма.

```
  selectDatabase();
  $query_S = "SELECT author, name, href FROM items WHERE isnew = 1
  ↳ORDER BY added DESC";
  $S = runQuery($query_S);
```

Создаем запрос SQL, выбирающий из таблицы `items` базы данных все "новые" статьи и файлы, и выполняем его.

```
  while ($row_S = mysql_fetch_assoc($S)) {
    $body = $body . $row_S['author'] . "\r\n";
    $body = $body . $row_S['name'] . "\r\n";
    $body = $body . $row_S['href'] . "\r\n";
    $body = $body . "\r\n";
  }
```

Формируем тело письма.

```
  if ($body != "") {
```

Проверяем, есть ли что рассылать, и, если есть, выполняем следующий код.

```
    $query_Addresses = "SELECT * FROM maillist";
    $Addresses = runQuery($query_Addresses);
```

Создаем запрос SQL, выбирающий из таблицы `maillist` все почтовые адреса, и выполняем его.

```
    while ($row_Addresses = mysql_fetch_assoc($Addresses)) {
      $rcpt_to = $rcpt_to . "RCPT TO: " . $row_Addresses['email'] .
      ↳"\r\n";
```

```
    if ($bcc != "") $bcc = $bcc . ",";
    $bcc = $bcc . $row_Addresses['email'];
  }
```

Формируем списки команд `RCPT TO:` почтового сервера и список адресов для строки `Bcc:` заголовка письма.

```
  if ($bcc != "") {
```

Проверяем, есть ли кому рассылать, и, если есть, выполняем следующий код.

```
    $connection = fsockopen("smtp.mail.ru", 25);
```

Здесь мы подразумеваем, что рассылка ведется через сервер **smtp.mail.ru**. В реальности сервер может быть другим — тогда будет нужно изменить это выражение.

```
    fputs($connection, "HELO php\r\n");
    fputs($connection, "MAIL FROM: php@mail.ru\r\n");
    fputs($connection, $rcpt_to);
    fputs($connection, "DATA\r\n");
    fputs($connection, "Content-Type: text/plain;
    ↳charset=windows-1251\r\n");
    fputs($connection, "From: php@mail.ru\r\n");
    fputs($connection, "Bcc: " . $bcc . "\r\n");
    fputs($connection, "Subject: Обновления 'Нашего архива'\r\n");
    fputs($connection, "\r\n");
    fputs($connection, $body);
```

Замечание

Вообще-то, в тело письма надо бы вставить еще несколько строк: приветствие, адрес сайта, прощание и — обязательно — гиперссылку на страницу отписки от рассылки. Это сделать нетрудно, поэтому автор не стал приводить здесь соответствующий код PHP.

```
    fputs($connection, ".\r\n");
    fputs($connection, "RSET\r\n");
    fclose($connection);
    runQuery("UPDATE items SET isnew = 0 WHERE isnew = 1");
  }
```

Отправляем готовое письмо на почтовый сервер и превращаем все "новые" записи в "старые".

```
  mysql_free_result($Addresses);
}
```

```
  mysql_free_result($S);
  header("Location: ../default.php");
}
?>
```

Напоследок удаляем все созданные наборы записей и выполняем переход на главную страницу сайта. Вот весь сценарий.

Поместим его в страницу Subscribe_run.php и сохраним ее. После этого вставим в самое начало страницы, перед всеми остальными сценариями PHP, вот такой небольшой сценарий:

```
<?php require('../Library.php'); ?>
```

Он подключает файл Library.php, в который мы поместили две часто используемые функции. Поскольку в этом файле уже определена функция `isAuthorized`, нам будет нужно удалить объявление этой функции из сценария, соответствующего серверному поведению **Restrict Access To Page**, иначе обработчик PHP не сможет выполнить данный сценарий.

Теперь откроем главную страницу нашего сайта default.php и найдем код HTML, выводящий абзацы с гиперссылками на административные страницы списков категорий и управления файлами. Вот он:

```
<?php if ($isAorM) { ?>
<H2>Администрирование</H2>
<P>Чтобы попасть на административную страницу списка категорий файлов,
щелкните по <A HREF="Admin/Categories_admin.php?file=1">этой
гиперссылке</A>. А <A HREF="Admin/Categories_admin.php?file=0">эта
гиперссылка</A> ведет на список категорий статей.</P>
<P><A HREF="Admin/Files.php">Эта</A> гиперссылка ведет на страницу
управления файлами.</P>
<?php } ?>
```

Вставим в него еще один абзац с гиперссылкой, указывающей на только что созданную нами страницу запуска рассылки (выделен полужирным шрифтом):

```
<?php if ($isAorM) { ?>
<H2>Администрирование</H2>
<P>Чтобы попасть на административную страницу списка категорий файлов,
щелкните по <A HREF="Admin/Categories_admin.php?file=1">этой
гиперссылке</A>. А <A HREF="Admin/Categories_admin.php?file=0">эта
гиперссылка</A> ведет на список категорий статей.</P>
<P><A HREF="Admin/Files.php">Эта</A> гиперссылка ведет на страницу
управления файлами.</P>
```

```
<P><A HREF="Admin/Subscribe_run.php">Эта</A> гиперссылка ведет на
страницу запуска почтовой рассылки.</P>
<?php } ?>
```

Сохраним обе страницы — Subscribe_run.php и default.php, — опубликуем их на Web-сервере и проверим в действии. Только не забудем сначала подключиться к Интернету...

Страница отписки от рассылки

Что ж, как говорится, насильно мил не будешь... И вполне могут найтись люди, которым почему-то не понравится наша рассылка, и они, просмотрев пару выпусков, больше не захотят ее получать. Специально для них нам будет нужно создать страницу для отписки от рассылки. А что делать — посетитель всегда прав!..

Создадим в Dreamweaver очередную пустую страницу PHP, дадим ей название `Отписка`, такой же заголовок и введем какой-либо поясняющий текст. Сохраним эту страницу под именем Unsubscribe.php в корневой папке локальной копии нашего сайта.

Чтобы посетитель смог отписаться от рассылки, он должен ввести свой почтовый адрес. Для этого нам нужно создать на странице отписки форму. Дадим ей имя `unsubscribe`. В этой форме создадим поле ввода с именем `email` и кнопку отправки данных с именем `submit` и надписью `Отписаться`. В общем, все, как на странице Subscribe.php.

Отписка от рассылки будет заключаться в том, что запись таблицы `maillist` базы данных `site`, в которой хранится введенный посетителем адрес, будет удалена. Это значит, что нам будет нужно создать серверное поведение **Delete Record**, выполняющее удаление записи из таблицы.

Для поиска удаляемой записи мы предпишем использовать поле ввода `email` формы `unsubscribe`. Для этого выберем пункт **Form Variable** раскрывающегося списка **Primary key value** диалогового окна **Delete Record**, а в расположенное правее этого списка поле ввода введем `email`. А в поле ввода **After deleting, go to**, где указывается страница, на которую будет выполнен переход после успешного удаления записи, введем имя страницы Unsubscribe_ok.htm, которую создадим потом.

Закончим создание серверного поведения **Delete Record** и приступим к странице Unsubscribe_ok.htm. Введем в нее какой-либо текст, говорящий о том, что отписка прошла нормально. После этого сохраним все вновь созданные страницы и закроем их.

Осталось только поместить гиперссылку, указывающую на готовую страницу подписки, на главную страницу нашего сайта. Найдем заголовок `Подписка` с абзацем, приглашающим подписаться на рассылку, и создадим под ним

новый абзац. Введем в него поясняющий текст, поместим там же гиперссылку на страницу Unsubscribe.php. И сохраним исправленную страницу.

Ну что, проверим все это хозяйство в действии? Опубликуем готовые страницы на Web-сервере и попробуем отписаться от нашей рассылки. После этого соответствующая запись таблицы `maillist` должна быть удалена.

Что дальше?

Мы создали свой второй сайт! Разместили на нем весьма полезную информацию. Предоставили пользователю удобный интерфейс для выборки данных, предложили подписаться на список рассылки. Пока все это делали, разобрались в хитростях Dreamweaver, PHP и MySQL. Обзавелись поклонниками и помощниками, которые будут наполнять наш сайт полезным содержимым.

На этом мы и закончим свое первое знакомство с PHP, MySQL, Dreamweaver и вообще интернет-программированием. А для тех, кто хочет его потом продолжить, автор написал заключение.

Заключение

Ну вот и закончилась книга!.. Что сказал по этому поводу знаток компьютерных и интернет-технологий, стоявший у прилавка книжного магазина и сокрушавшийся по поводу истраченной зря бумаги, нам не известно. Он ушел домой читать книгу. Нам же некогда было за ним следить.

- Мы изучили язык написания Web-страниц HTML.
- Мы узнали, как работать в Dreamweaver.
- Мы создали свой самый первый, самый простой Web-сайт.
- Мы узнали о базах данных, серверах данных и языке составления запросов SQL.
- Мы научились программировать на языке PHP.
- Соединив в одну могучую упряжку HTML, PHP, MySQL и Dreamweaver, мы создали второй сайт, гораздо более сложный.
- И — самое главное — изучив PHP-код, созданный Dreamweaver, мы выяснили, *как* пишутся серверные Web-страницы. А ведь мы и хотели это узнать, не так ли?

Вот сколько всего мы успели сделать и узнать!

Но еще больше мы не знаем.

- Мы совершенно не занимались Web-дизайном, "красотой". Из-за этого наши сайты выглядят весьма непрезентабельно. (Собственно, у нас и не было цели изучить Web-дизайн — мы занимались именно Web-программированием. Но все же...)
- Мы не использовали многие возможности MySQL, которые могут сильно облегчить жизнь программиста. Но все они описаны в документации по MySQL, которую можно найти на сайте **http://www.mysql.com**, в том числе, и на русском языке. А хороший программист должен читать руководства.
- Также мы не использовали многие новые возможности PHP, появившиеся в версиях 5.x. Это, прежде всего, расширенная модель объектно-

ориентированного программирования, сильно облегчающая задачи программиста во многих случаях. Но все это также описано в документации по PHP, которую мы со временем прочитаем.

- Мы не занимались серьезно ни оптимизацией наших сценариев PHP, ни обеспечением безопасности и устойчивости к взлому. Кое-что, правда, в этом плане мы все-таки сделали, но надо бы сделать больше.
- Мы мало работали с Dreamweaver. А ведь это мощнейший пакет, который можно изучать очень и очень долго и не уставать поражаться, сколько же в него "запихали" разработчики.
- И вообще, у нас мало опыта. Но опыт — дело практики, которая у нас еще будет.

Именно поэтому разговор о MySQL, PHP и Dreamweaver еще не закончен. Если мы хотим его продолжить, нам нужно будет читать уже совсем другие книги, более полные, ориентированные на профессионалов. Эта книга только дала начальный толчок к изучению Web-программирования, самые начальные знания, самый первый опыт.

В изучении Web-программирования нам помогут не только книги, которые еще нужно где-то найти и как-то купить, но и бесплатные ресурсы Интернета. В табл. 3.1 перечислены ссылки на некоторые из этих ресурсов, которыми пользовался и автор во время работы над этой книгой.

Таблица 3.1. *Ссылки на Web-сайты по теме книги*

Ссылка	Описание
http://www.mysql.com	Сайт MySQL AB — организации, занимающейся разработкой, поддержкой и распространением сервера данных MySQL. Дистрибутивы различных версий этого сервера, документация, дополнительные программы, полезные советы, призывы к желающим принять участие в разработке сервера
http://www.php.net	Сайт группы разработчиков PHP. Дистрибутивы, документация, полезные советы, призывы к желающим принять участие в разработке PHP, ссылки на другие полезные сайты по PHP
http://httpd.apache.org	Сайт Apache Group — организации, занимающейся разработкой, поддержкой и распространением Web-сервера Apache. Дистрибутивы, документация, дополнительные программы, полезные советы, призывы к желающим принять участие в разработке сервера

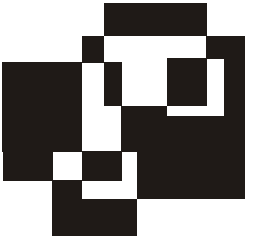
Таблица 3.1 *(окончание)*

Ссылка	Описание
http://www.macromedia.com	Сайт фирмы Macromedia — разработчика Dreamweaver. Дистрибутивы всех программ, выпускаемых этой фирмой (в том числе и самого Dreamweaver), документация, полезные советы, модули расширения, форум разработчиков
http://www.phpmyadmin.net	Сайт группы разработчиков phpMyAdmin — программы клиента данных. Дистрибутивы, документация, полезные советы, призывы присоединиться к разработчикам этой программы
http://pear.php.net	Огромная библиотека хорошо документированных примеров, готовых функций и классов, написанных на PHP. Рекомендуется всем серьезным PHP-программистам!
http://www.phpclub.ru	Сайт русской группы PHP-программистов. Статьи, готовые фрагменты кода, анонсы мероприятий, форум разработчиков
http://www.phpinside.ru	Прекрасный бесплатный электронный журнал по PHP-программированию. Распространяется в формате PDF (Portable Document Format, формат переносимых документов), разработанном фирмой Adobe (**http://www.adobe.com**) и поддерживаемом программой Adobe Reader. Также выходит в "бумажном" виде
http://www.php.spb.ru	Небольшой сайт петербургской группы PHP-программистов. Множество очень полезных статей и советов
http://subscribe.ru	Сайт популярнейшей в России службы рассылок. Среди миллионов рассылок найдется с десяток посвященных PHP
fido7.ru.php	Группа новостей, посвященная PHP-программированию

Что ж, пора прощаться. До свидания! До новых встреч!

Владимир Дронов

vlad@vgi.volsu.ru



Приложения

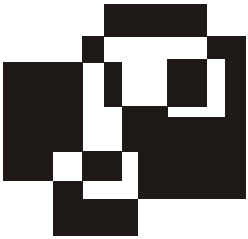
Приложение 1. Установка Web-сервера Apache

Приложение 2. Установка сервера данных MySQL

Приложение 3. Установка обработчика PHP

Приложение 4. Установка и использование клиента данных phpMyAdmin

Приложение 1



Установка Web-сервера Apache

Apache — популярный бесплатный Web-сервер, дистрибутив которого можно найти на сайте Apache Group (**http://httpd.apache.org**) — организации, занимающейся его разработкой, поддержкой и распространением. На этом сайте доступны различные версии Apache для разных операционных систем.

Процесс установки описывается на примере версии 2.0.51 для операционных систем Windows. Дистрибутив поставляется в формате Microsoft Installer (файл с расширением msi), поэтому пользователям Windows 95, 98, Me и NT нужно будет загрузить с сайта **http://www.microsoft.com** и установить на своем компьютере исполняемую среду Microsoft Installer.

Установка

После запуска файла apache_2.0.51-win32-x86-no_ssl.msi на выполнение на экране появится приглашение к установке Apache (рис. П1.1). Нажимаем кнопку **Next**.

Следующее по порядку появления окно — лицензионное соглашение (рис. П1.2). Соглашаемся с ним, включив переключатель **I accept the terms in the license agreement**, и нажимаем кнопку **Next**.

На рис. П1.3 показано следующее окно, содержащее краткие сведения о сервере Apache. Нажимаем кнопку **Next**.

Далее мы видим окно задания параметров сервера (рис. П1.4). В поле **Network Domain** вводим домен, в котором находится наш компьютер, в поле **Server Name** — доменное имя нашего компьютера, а в поле **Administrator's Email Address** — почтовый адрес администратора (то есть наш).

При установке сервера под Windows NT, 2000, XP и 2003 в этом окне будет присутствовать группа переключателей **Install Apache HTTP Server 2.0 programs and shortcuts for**. Поскольку сервер нам нужен только для тестирования нашего сайта, то включим флажок **only for the Current User, on Port 8080, when started Manually**. После этого Apache будет установлен только для текущего пользователя многопользовательской системы, будет настроен на порт 8080, а запускать мы его будем вручную.

Рис. П1.1. Приглашение к установке Apache

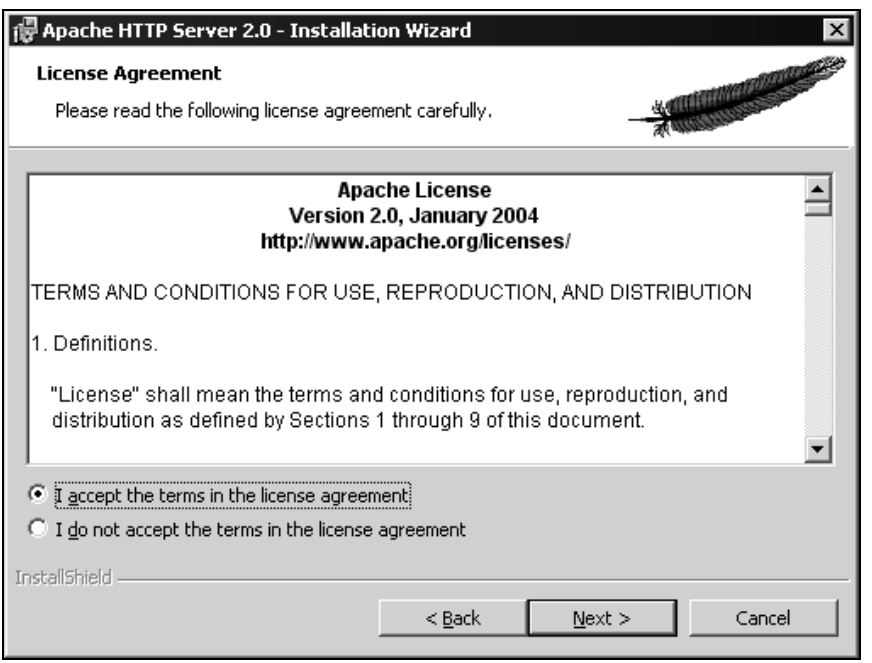


Рис. П1.2. Лицензионное соглашение

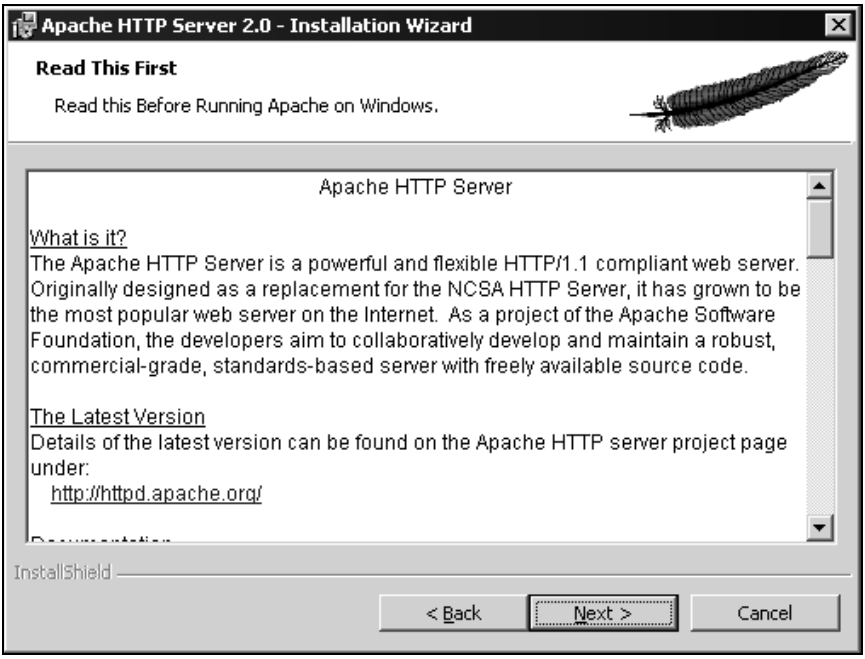


Рис. П1.3. Сведения о сервере Apache

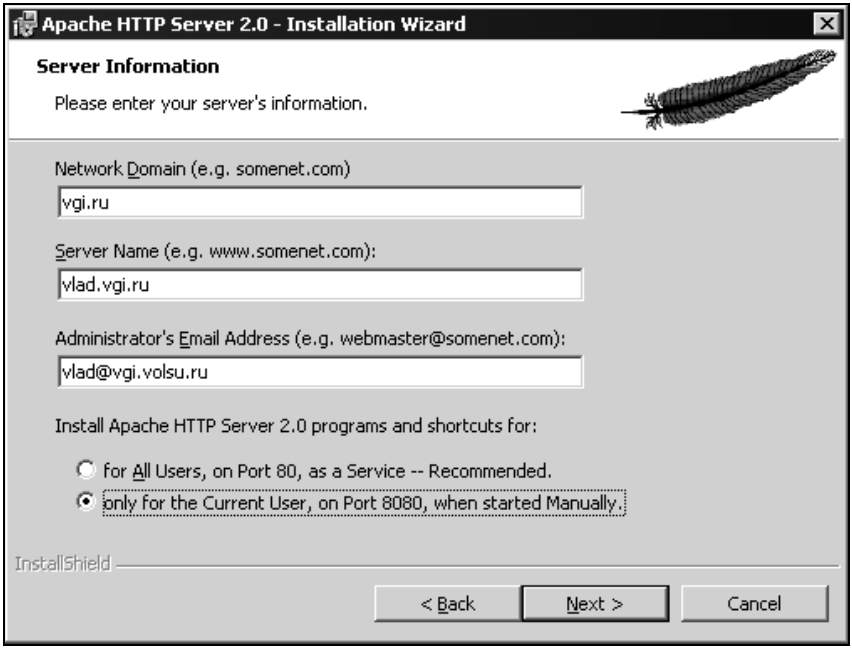


Рис. П1.4. Окно задания параметров сервера

Замечание

Флажок **for All Users, on Port 80, as a Service** стоит включать только тогда, когда мы собираемся сделать свой сайт постоянно действующим и доступным всем пользователям сети. В этом случае Apache будет запускаться вместе с операционной системой и будет настроен на порт 80 (порт по умолчанию для WWW).

Завершив настройку, нажмем кнопку **Next**. Далее на экране появится окно задания типа установки: типичная или выборочная (рис. П1.5). Включим переключатель **Typical**, задающий типичную установку, — ее для наших целей будет вполне достаточно — и нажмем кнопку **Next**.

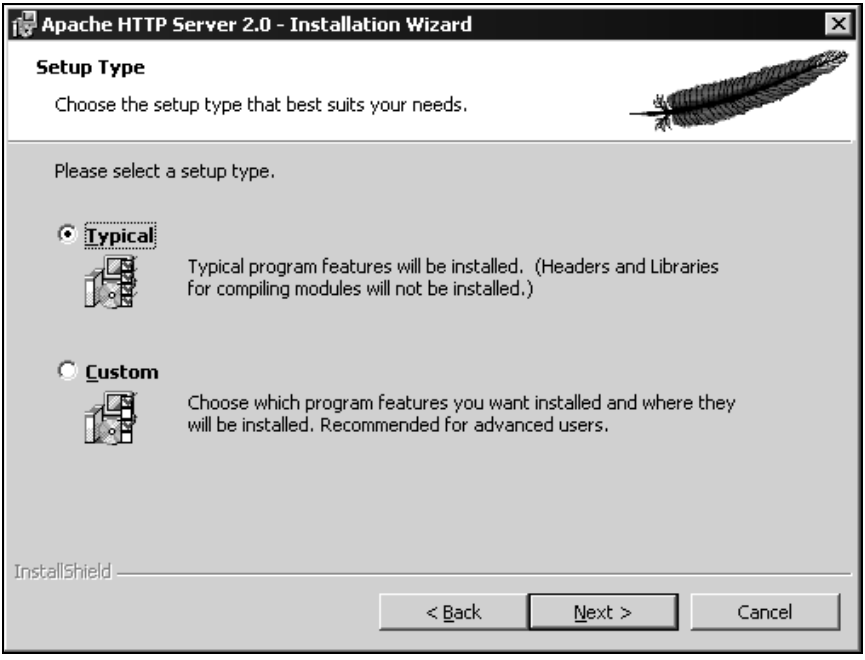


Рис. П1.5. Окно задания типа установки

Далее мы увидим окно задания папки, в которую будет установлен Apache (рис. П1.6). По умолчанию это папка Apache Group, находящаяся в системной папке Program Files. Обычно не возникает особой нужды менять эту папку, поэтому нажимаем кнопку **Next**.

Последнее окно, которое появится на экране, сообщает, что сейчас начнется установка. Нажмем кнопку **Next** и подождем, пока она не завершится.

Сигналом того, что установка успешно завершилась, будет окно, показанное на рис. П1.7. Чтобы закрыть его, нажмем кнопку **Finish**.

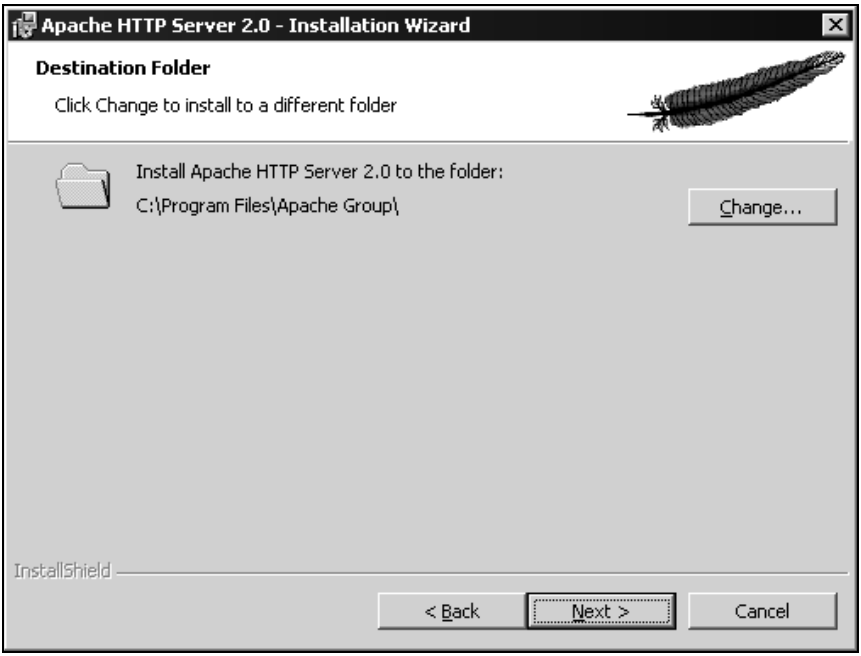


Рис. П1.6. Окно задания папки, в которую будет установлен сервер

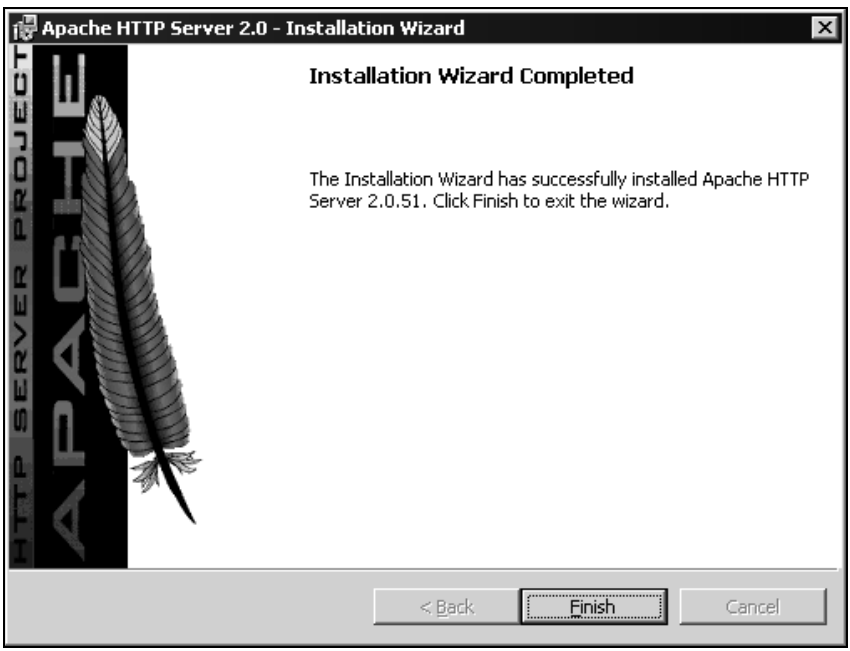


Рис. П1.7. Окно, сообщающее о завершении установки

Запуск и остановка

Чтобы запустить Apache, нажмем хорошо знакомую нам кнопку **Start** (Пуск), выберем в открывшемся меню пункт **Apache HTTP Server 2.0.51**, в появившемся на экране подменю выберем пункт **Control Apache Server**, а в другом подменю — пункт **Start Apache in Console**. На экране появится окно сеанса MS-DOS (в случае Windows 95, 98 или Me) или окно консоли (в случае Windows NT, 2000, XP или 2003) с заголовком **Start Apache in Console**. Все, сервер запущен.

Теперь давайте проверим только что установленный Web-сервер в действии. Запустим Web-обозреватель и наберем в строке адреса **http://localhost:8080/**. В ответ Web-обозреватель выведет небольшую страничку, сообщающую о том, что сервер нормально установлен и нормально работает (рис. П1.8).

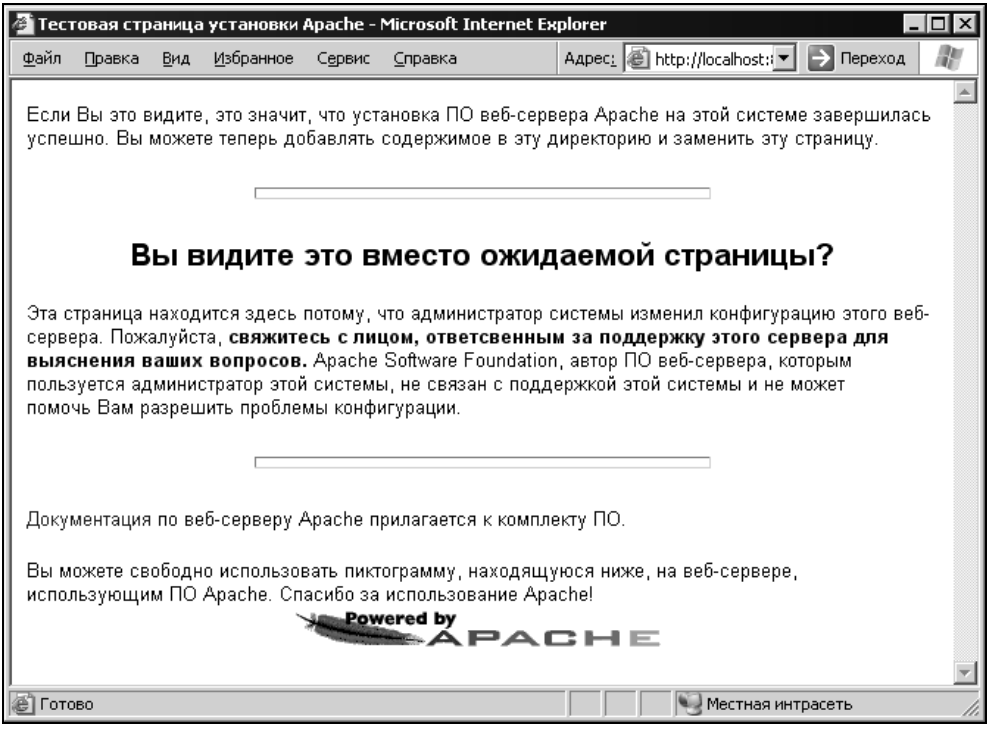


Рис. П1.8. Web-страница, сообщающая о нормальной работе Web-сервера

Для остановки Web-сервера нужно сначала закрыть все окна Web-обозревателя, в которых открыты страницы с этого сервера, а потом закрыть окно **Start Apache in Console**. Через несколько секунд оно пропадет — сервер будет остановлен.

Предварительная настройка

В принципе, Apache не требует дополнительной настройки после установки, так как все нужные параметры мы задали еще при его установке, в окне задания его параметров (см. рис. П1.4). Но кое-какая минимальная настройка все-таки понадобится.

Прежде чем настраивать Apache, остановим его, если он запущен. Далее запустим Проводник Windows или другую программу для управления файлами и откроем папку conf, находящуюся в папке, в которой был установлен Apache. Откроем в любом текстовом редакторе, например, Блокноте, файл httpd.conf и найдем в нем такую строку:

```
DirectoryIndex index.html index.html.var
```

Эта строка задает имя файла, в котором хранится страница по умолчанию. Видно, что изначально задано имя index.html. Давайте добавим сюда имена страниц по умолчанию, которые используются в этой книге, — default.htm и default.php. Также нам нужно добавить туда еще одно имя — index.php, — чтобы нормально работал phpMyAdmin. Перепишем эту строку, чтобы она выглядела так:

```
DirectoryIndex default.htm default.php index.php index.html
↳index.html.var
```

Теперь необходимо добавить в Apache поддержку PHP. Для этого в любое место файла httpd.conf добавим вот такие строки:

```
LoadModule php4_module <путь, где установлен PHP>/php4apache.dll
AddType application/x-httpd-php .php
```

Замечание

Процесс добавления поддержки PHP в Apache (и другие программы Web-серверов) описан в файле install.txt, находящемся в папке, где установлен PHP.

И последняя настройка. Найдем в файле httpd.conf вот такие строки:

```
<Directory "C:/Program Files/Apache Group/Apache2/htdocs">
    Options Indexes FollowSymLinks
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Будем внимательны — эти строки "разбавлены" комментариями (в файле конфигурации Apache комментарии начинаются со знака решетки #). Нам

будет нужно изменить третью сверху строку, чтобы она выглядела вот так (выделено полужирным шрифтом):

```
<Directory "C:/Program Files/Apache Group/Apache2/htdocs">
    Options Indexes FollowSymLinks
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
```

Это необходимо, чтобы Apache обрабатывал файлы .htaccess, хранящиеся в корневой папке. По умолчанию он их игнорирует.

Ну, вот и все. Сохраним файл httpd.conf и закроем. При следующих запусках Apache будет использовать заданные нами настройки.

Внимание

Прежде чем изменять какие-либо параметры в файле конфигурации httpd.conf Apache, нужно остановить его. Apache прочитывает файл httpd.conf при запуске и в процессе работы к нему не обращается.

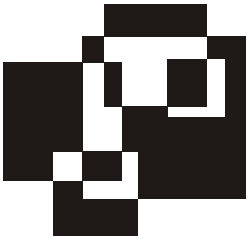
Доступ к справочной системе Apache

Вместе с Apache поставляется весьма подробная справочная система, описывающая его возможности и настройку. Чтобы получить к ней доступ, необходимо:

1. Запустить сам Web-сервер Apache.
2. Запустить Web-обозреватель.
3. Набрать в строке адреса Web-обозревателя **http://localhost:8080/manual/**.

К сожалению, справочная система Apache переведена на русский язык частично — бóльшая ее часть написана по-английски.

Приложение 2



Установка сервера данных MySQL

MySQL — популярный бесплатный сервер данных, дистрибутив которого можно найти на сайте MySQL AB (**http://www.mysql.com**) — организации, занимающейся его разработкой, поддержкой и распространением. На этом сайте доступны различные версии MySQL для разных операционных систем.

Процесс установки описывается на примере версии 4.0.18 для операционных систем Windows. Дистрибутив поставляется в архиве ZIP, который нужно распаковать в любую папку.

Установка

После запуска файла setup.exe на выполнение на экране появится приглашение к установке MySQL (рис. П2.1). Нажимаем кнопку **Next**.

Далее появится окно, содержащее сведения по настройке сервера (рис. П2.2). Читаем их и снова нажимаем кнопку **Next**.

Следующее окно служит для задания папки, в которую будет установлен MySQL (рис. П2.3). По умолчанию это папка mysql, находящаяся в корневой папке диска С. Конечно, можно ее не менять, но по правилам хорошего тона все программы Windows должны быть установлены в папку Program Files. Так что лучше давайте изменим путь установки сервера.

Щелкнем по кнопке **Browse**. На экране появится небольшое диалоговое окно **Choose Folder**, показанное на рис. П2.4. Занесем в поле ввода **Path** этого окна путь C:\Program Files\mysql и нажмем кнопку **OK**. Снова оказавшись в диалоговом окне задания пути установки (см. рис. П2.3), нажмем кнопку **Next**.

После этого мы увидим окно задания типа установки: типичная, компактная или выборочная (рис. П2.5). Включим переключатель **Typical**, задающий типичную установку, — ее для наших целей будет вполне достаточно — и нажмем кнопку **Next**.

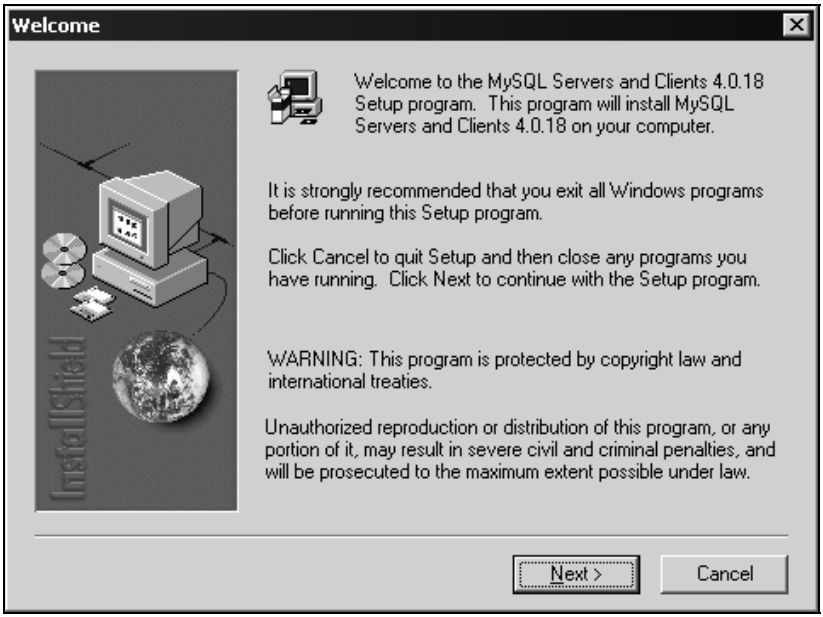


Рис. П2.1. Приглашение к установке MySQL

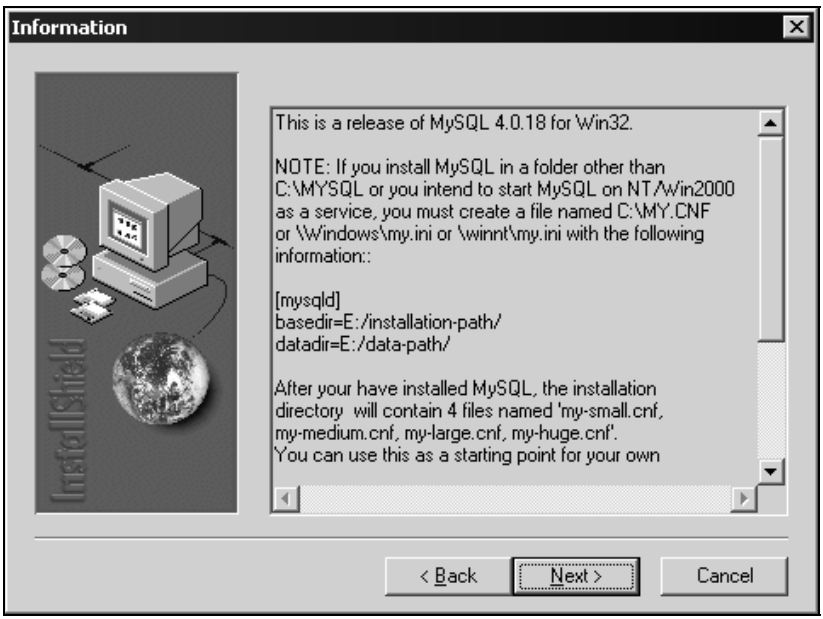


Рис. П2.2. Сведения о настройке

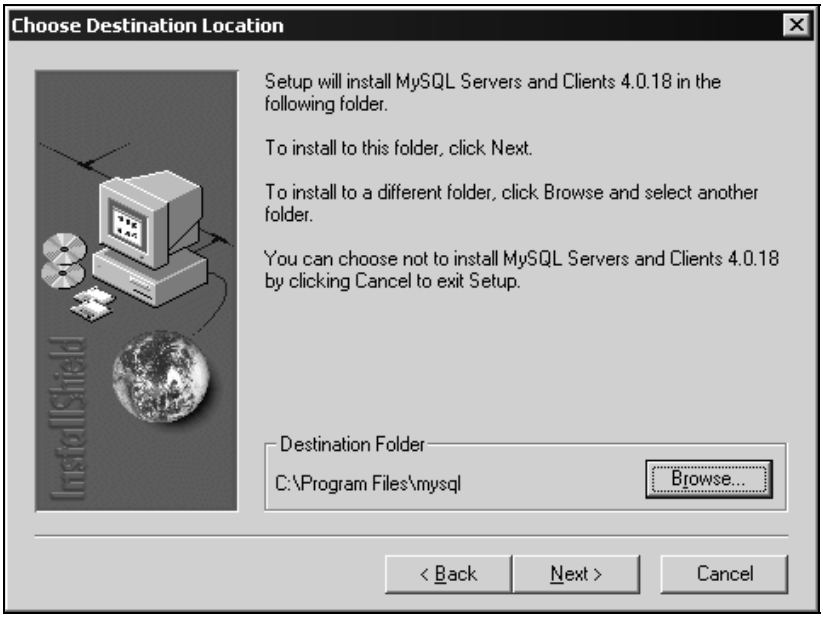


Рис. П2.3. Окно задания папки, в которую будет установлен сервер

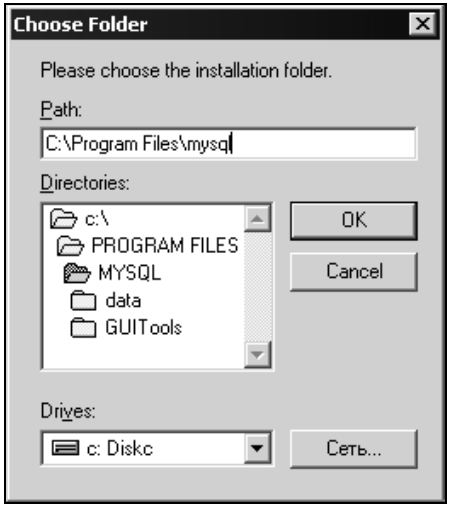


Рис. П2.4. Диалоговое окно **Choose Folder**

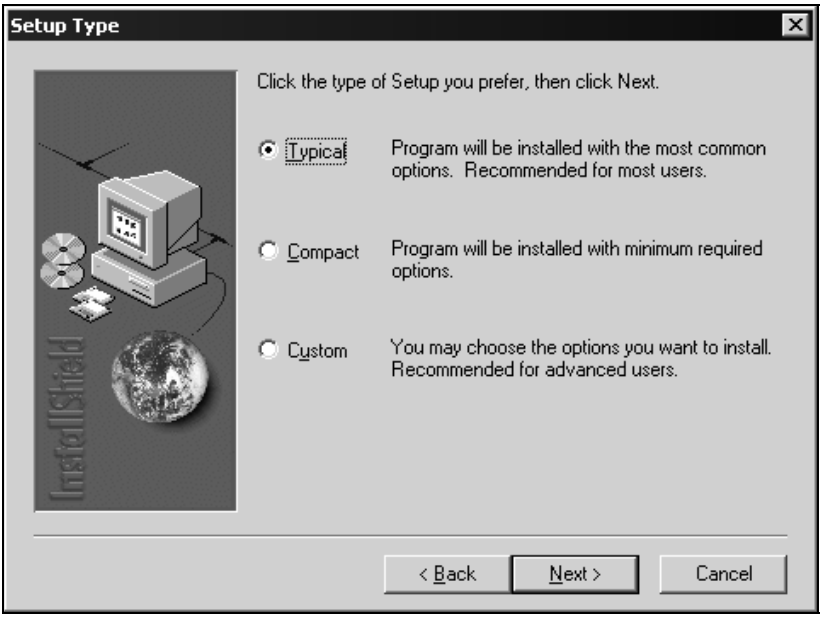


Рис. П2.5. Окно задания типа установки

Рис. П2.6. Окно, сообщающее о завершении установки

Сразу же после этого начнется сам процесс установки. Подождем, пока он завершится, и нажмем кнопку **Finish** последнего окна, которое появится на экране (рис. П2.6). Все, сервер данных MySQL установлен!

При установке сервера будут автоматически созданы базы данных `mysql` и `test`. Первая база данных содержит служебные данные — списки баз данных, таблиц, полей, индексов, пользователей и их прав — так что трогать ее не стоит. Вторая база данных — просто пример, поэтому при желании ее можно удалить.

Также сразу после установки будут созданы три пользователя. Двоих из них следует сразу же удалить, оставив только пользователя `root@localhost` — администратора, имеющего право подключаться с локального компьютера. От его имени мы потом создадим нашу базу данных `site`.

Предварительная настройка

В отличие от "покладистого" Apache, MySQL требует перед запуском предварительной настройки. Если же ее не выполнить, сервер может работать с ошибками или не работать вообще.

Запустим Проводник Windows или другую программу для управления файлами и откроем папку, в которой установлена сама система Windows (обычно эта папка находится в корневой папке диска C и имеет имя Windows или WinNT). Создадим в этой папке текстовый файл my.ini, откроем его в любом текстовом редакторе, например, Блокноте, и впишем такой текст:

```
[mysqld]
basedir=<путь, по которому установлен MySQL>
datadir=<путь, по которому установлен MySQL>/data
default-character-set=cp1251
```

В нашем случае содержимое файла my.ini будет выглядеть так:

```
[mysqld]
basedir=C:/Program Files/mysql
datadir=C:/Program Files/mysql/data
default-character-set=cp1251
```

Обратим внимание на то, что здесь для разделения папок используется символ "прямой слеш" (/) вместо принятого в Windows "обратного слеша" (\). Дело в том, что этот же синтаксис файла my.ini применяется для настройки версий MySQL для операционных систем семейства Unix, в которых для разделения папок используется как раз символ "прямого слеша".

Проверим, не допустили ли мы ошибок, сохраним файл my.ini и закроем его. Все, самые основные параметры сервера мы задали.

Вкратце рассмотрим, что мы только что написали. Понятно, что параметр `basedir` задает папку, в которую установлен сервер MySQL; он нужен, чтобы сервер смог найти свои вспомогательные компоненты. Параметр `datadir` задает папку, в которой хранятся все базы данных; по умолчанию это папка data, находящаяся в папке, где установлен MySQL.

А сейчас — особое внимание! Этот момент очень плохо документирован в руководстве по MySQL, но его обязательно нужно учесть. Это касается строки

```
default-character-set=cp1251
```

без которой сервер, в принципе, заработает, но не совсем правильно.

Когда сервер данных (любой — не только MySQL) выполняет сортировку записей по полю строкового типа, он использует порядок сортировки, принятый для заданного в его настройках языка. Поскольку мы будем хранить в базах данных MySQL строки на русском языке, нам будет нужно задать соответствующий порядок сортировки. Так вот, приведенная ранее строка как раз и задает этот порядок — соответствующий русской кодировке 1251, принятой в системах семейства Windows. (О кодировках рассказывается в *главе 2*.)

Итак, файл настройки my.ini мы создали. Теперь сервер данных MySQL может быть запущен.

Запуск и остановка

Запуск и остановка сервера данных MySQL будут выполняться по-разному, в зависимости от версии Windows, в которой он установлен. Так что мы рассмотрим этот процесс отдельно для Windows 95, 98, Me, Windows NT и Windows 2000, XP, 2003.

Запуск и остановка под Windows 95, 98 и Me

Для запуска MySQL под версиями 95, 98 и Me системы Windows нужно воспользоваться Командной строкой MS-DOS — эта скромная, но полезная утилита поставляется в составе системы. Запустим ее и наберем в ней такой текст, после каждой строки нажимая клавишу <Enter>:

```
cd \
cd Program Files
cd mysql
cd bin
```

Эти команды выполняют переход в папку bin, находящуюся в папке, где установлен сервер. Теперь набираем вот такую команду (опять же, не забыв нажать после ее ввода клавишу <Enter>):

```
mysqld --standalone
```

Она и запустит наш сервер данных.

Нужно иметь в виду, что при запуске сервера на экране не появится никаких окон. MySQL работает "тихо".

Чтобы остановить MySQL, нужно использовать командную строку. Опять наберем приведенные ранее четыре команды перехода в папку bin, а следом за ними — вот такую команду:

```
mysqladmin -u root shutdown
```

и сервер будет тут же остановлен и выгружен из памяти.

Замечание

В приведенной ранее команде останова сервера используется утилита MySQLAdmin. Это поставляемая в составе MySQL сервисная утилита, применяемая для настройки некоторых параметров сервера, его запуска и остановки, а также просмотра сведений об открытых базах данных и подключенных пользователях.

Запуск и остановка под Windows NT

Под версией NT системы Windows MySQL может быть запущен как служба. *Служба* — это особая программа Windows, которая вообще не взаимодействует с пользователем, а занимается только тем, что обслуживает другие программы.

Внимание

Службы поддерживаются только системами Windows NT, 2000, XP и 2003. В Windows 95, 98 и Me службы лучше не запускать — они не работают правильно.

Любая служба перед запуском должна быть правильным образом установлена. И сделать это нам придется самим. Запустим стандартно поставляемую в составе системы утилиту Командная строка и наберем в ней четыре уже знакомых нам команды и одну незнакомую (не забываем после ввода всех команд нажимать клавишу <Enter>):

```
cd \
cd Program Files
cd mysql
cd bin
mysqld-max-nt --install
```

Последняя — незнакомая команда — как раз и установит MySQL в системе как службу.

Запустить же установленный как служба MySQL можно двумя способами. Если мы еще не закрыли Командную строку, то можем набрать в ней вот такую команду:

```
net start MySQL
```

Это первый способ. Второй способ — использование специального апплета **Службы** (Services) Панели управления. Откроем Панель управления, дважды щелкнем по значку **Службы** (Services), и на экране появится окно **Службы** (Services) — апплет будет запущен. Выберем в списке установленных служб пункт MySQL и щелкнем кнопку запуска. После этого можно закрыть апплет и саму Панель управления.

Остановить MySQL также можно двумя способами. Во-первых, можно запустить командную строку и набрать в ней команду

```
net stop MySQL
```

Во-вторых, можно запустить апплет **Службы** (Services), выбрать в списке MySQL и щелкнуть кнопку остановки службы.

Запуск и остановка под Windows 2000, XP, 2003

Установка MySQL как службы в системах Windows 2000, XP и 2003 выполняется так же, как и в Windows NT. Точно так же в них выполняется запуск и остановка службы MySQL с помощью Командной строки. Единственное отличие: для работы со службами в этих версиях Windows используется не апплет Панели управления, а специальная оснастка **Службы** (Services) системы управления Microsoft Management Console.

Откроем Панель управления, дважды щелкнем по значку **Администрирование** (Administration) и в появившемся на экране окне **Администрирование** (Administration) дважды щелкнем значок **Службы** (Services). Одноименная оснастка будет запущена, и на экране появится ее окно. Здесь, как и в случае апплета **Службы** (Services) в Windows NT, выберем в списке установленных служб MySQL и нажмем кнопку запуска.

Если же нам будет нужно остановить MySQL, мы снова запустим оснастку **Службы** (Services), выберем в списке MySQL и щелкнем кнопку остановки службы.

Доступ к справочной системе MySQL

Полное справочное руководство на английском языке поставляется в составе MySQL. Чтобы получить к нему доступ, нужно найти папку Docs, находящуюся в папке, в которой установлен MySQL, и открыть Web-страницу

manual_toc.html в Web-обозревателе. Эта страница содержит оглавление руководства.

Справочное руководство по MySQL на русском языке (и весьма качественно переведенное!) можно найти на сайте **http://www.mysql.com**. Оно также представляет собой набор Web-страниц в архиве ZIP. После распаковки этого архива в какую-либо папку нужно открыть в Web-обозревателе страницу manual.ru_toc.html, содержащую оглавление.

Вспомогательные программы

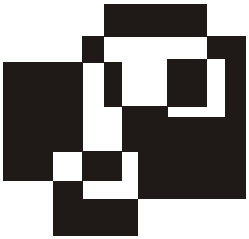
Кроме самого сервера данных MySQL, на сайте **http://www.mysql.com** можно найти две вспомогательные программы, которые могут помочь нам в работе. Давайте посмотрим, что это за программы.

MySQL Control Center — это весьма мощный и удобный в использовании клиент данных MySQL. С его помощью можно быстро выполнить всю работу по созданию базы данных. В настоящее время доступна версия 0.9.4.

Замечание

Мы для создания своей базы данных будем использовать клиент phpMyAdmin. Его установка и использование описаны в *приложении 4*.

MySQL Administrator — утилита, призванная администрировать и обслуживать сам сервер MySQL. Правда, пользы от нее не очень много; вероятно, это из-за того, что пока доступна только предварительная версия этой программы (бета-версия) 1.0.7 beta.



Приложение 3

Установка обработчика PHP

Здесь будет описан процесс установки обработчика активных серверных страниц PHP на локальный компьютер. При этом имеется в виду, что Web-сервер уже установлен, настроен и нормально работает. (Процесс установки Web-сервера Apache описан в *приложении 1.*)

PHP — популярная бесплатная платформа для создания активных серверных Web-страниц, дистрибутив которой можно найти на сайте группы разработчиков PHP (**http://www.php.net**). На этом сайте доступны различные версии PHP для разных операционных систем.

Процесс установки описывается на примере версии 4.3.6 для операционных систем Windows.

Установка

Процесс установки PHP очень прост. Дистрибутив PHP поставляется в виде архива ZIP, который нужно распаковать в папку Program Files (все программы Windows должны быть установлены в данную папку — это своего рода стандарт де-факто). После чего в папке Program Files будет создана папка php-4.3.6-win32, в которой находятся все файлы, входящие в комплект поставки PHP.

После этого нам обязательно будет нужно скопировать файл php4ts.dll из папки, в которой установлен PHP, в папку, в которую установлена Windows. (Обычно эта папка находится в корневой папке диска C и имеет имя Windows или WinNT.) Этот файл содержит ядро обработчика PHP. А поскольку все программы в поисках нужных им библиотек DLL просматривают, кроме всего прочего, и папку, в которой установлена Windows, Web-сервер всегда сможет его найти и загрузить.

Внимание

Руководство по установке PHP 4.3.6 утверждает, что при работе совместно с Apache 2.x не будет выполняться отправка файлов на Web-сервер через Web-интерфейс. (Об этом см. *главу 15*.) Автор испытывал PHP 4.3.6 совместно с Apache 2.0.51 и не столкнулся ни с какими проблемами.

Предварительная настройка

PHP перед первым запуском также нужно настроить. Иначе он может не запуститься, и Web-сервер при попытке открыть любую страницу PHP будет выдавать сообщения об ошибках.

Прежде чем настраивать PHP, нам обязательно будет нужно остановить Web-сервер Apache, если он запущен. После этого запустим Проводник Windows или другую программу для управления файлами и откроем папку, в которой установлен PHP. Найдем в ней файл php.ini-dist, скопируем его в папку, в которую установлена система Windows, и изменим его имя на php.ini. Потом откроем этот файл в любом текстовом редакторе, например, Блокноте, и найдем в нем строку:

```
;include_path = ".;c:\php\includes"
```

Изменим ее таким образом:

```
include_path = "c:\Program Files\php-4.3.6-win32\includes"
```

(Здесь подразумевается, что мы установили PHP в папку Program Files\ php-4.3.6-win32 на диске C.) Обратим особое внимание на то, что точки с запятой в начале исправленной строки быть не должно.

Следующая строка, которую нам нужно изменить, выглядит так:

```
extension_dir = "./"
```

А должна она выглядеть так:

```
extension_dir = "c:\program files\php-4.3.6-win32\extensions"
```

А вот и третья по счету строка, которую нам нужно исправить:

```
;upload_tmp_dir =
```

Правим ее аналогично:

```
upload_tmp_dir = "C:\Program Files\php-4.3.6-win32\tmp"
```

И последняя строка:

```
;session.save_path = /tmp
```

должна выглядеть так:

```
session.save_path = "C:\Program Files\php-4.3.6-win32\tmp"
```

Исправив эти четыре строки, сохраним файл php.ini и закроем его.

Внимание

Прежде чем изменять какие-либо параметры в файле конфигурации php.ini, нужно остановить Web-сервер. При этом Web-сервер завершит и выгрузит из памяти обработчик PHP.

Последнее, что нам осталось сделать, — это проверить, существует ли в папке, где установлен PHP, папка tmp. Если же такой папки нет, нам будет нужно ее создать. В ней обработчик PHP будет сохранять файлы с переменными уровня сессии (о них было рассказано в *главе 11*).

Запуск и остановка

Специально запускать обработчик PHP не нужно. Web-сервер сам загрузит и запустит его, как только мы откроем в Web-обозревателе любую страницу PHP. При запуске обработчик PHP прочитает файл конфигурации php.ini и использует заданные в нем настройки.

Чтобы завершить и выгрузить из памяти обработчик PHP, нам придется остановить сам Web-сервер. Увы — другого способа сделать это не существует.

Доступ к справочной системе PHP

Справочное руководство по PHP нужно будет загрузить с сайта **http://www.php.net** отдельно — вместе с самой программой оно не поставляется. На этом сайте можно найти руководства на многих языках, в том числе на русском. К несчастью, русскоязычное руководство переведено не полностью — многие разделы так и остались написанными по-английски.

Справочное руководство по PHP поставляется в разных форматах: набор Web-страниц, CHM и PDF. Автор рекомендует пользоваться руководством в формате CHM — на его взгляд, оно самое удобное. Поставляется оно в виде архива ZIP, содержащего файл php_manual_en.chm (это английское руководство; русское находится в файле php_manual_ru.chm).

Приложение 4

Установка и использование клиента данных phpMyAdmin

Здесь будет описан процесс установки клиентской программы для работы с базами данных MySQL phpMyAdmin на локальный Web-сервер.

Дистрибутив популярного бесплатного клиента данных phpMyAdmin можно найти на сайте группы его разработчиков (**http://www.phpmyadmin.net**). Клиент данных phpMyAdmin представляет собой набор серверных страниц PHP, поэтому подходит для всех операционных систем и всех Web-серверов, имеющих поддержку PHP.

Все бесплатные и платные Web-серверы, предоставляющие клиентам возможность работы с базами данных MySQL, как правило, уже имеют установленный phpMyAdmin. Как получить к нему доступ, должно быть описано на сайте поддержки. Если же администратор какого-либо Web-сервера не побеспокоился об установке phpMyAdmin, мы сами сможем без труда его установить.

Установка и настройка

Процесс установки и использования клиента данных phpMyAdmin описывается на примере версии 2.5.7.

Если мы собираемся устанавливать phpMyAdmin на локальный компьютер, то должны удостовериться, что на нем установлены Web-сервер, сам MySQL и обработчик PHP. Если же это не так, то будет нужно обратиться к *приложениям 1—3* этой книги, где объясняется процесс установки и настройки данных программ.

Дистрибутив phpMyAdmin поставляется в архиве ZIP, который нужно распаковать в любую папку. При этом будет создана папка phpMyAdmin-2.5.7-pl1, в которой будут находиться все файлы, входящие в состав phpMyAdmin.

Первое, что нужно сделать сразу после распаковки архива, — это выполнить настройку. Для этого зайдем в папку phpMyAdmin-2.5.7-pl1 и откроем

в Блокноте или аналогичном текстовом редакторе файл config.inc.php. Этот файл содержит объявления переменных, используемых в страницах PHP, которые и составляют phpMyAdmin.

Сначала найдем в этом файле такое выражение:

```
$cfg['PmaAbsoluteUri'] = '';
```

Оно задает полный интернет-адрес phpMyAdmin. Исправим его, чтобы оно выглядело таким вот образом:

```
$cfg['PmaAbsoluteUri'] = '<интернет-адрес нашего сайта>/
↳phpMyAdmin-2.5.7-pl1/';
```

Например, если мы установим phpMyAdmin на локальный Web-сервер, то это выражение примет вид:

```
$cfg['PmaAbsoluteUri'] = 'http://localhost:8080/phpMyAdmin-2.5.7-pl1/';
```

Ну, а в случае установки на удаленный Web-сервер (когда мы будем публиковать наш сайт в Сети) нужно будет подставить в него интернет-адрес удаленного Web-сервера, который нам выделят.

Далее прокручиваем содержимое файла config.inc.php вниз и находим такое выражение:

```
$cfg['Servers'][$i]['auth_type'] = 'config';
```

Оно задает способ подключения пользователя к серверу MySQL. Если там указано значение `config`, то имя пользователя и пароль берутся из того же файла config.inc.php. А указаны там пользователь `root` — администратор сервера MySQL — с "пустым" паролем!

Понятно, что это огромная "дыра" в безопасности. Ведь любой посетитель нашего сайта сможет зайти в phpMyAdmin, автоматически подключиться к серверу под именем `root` и натворить там все что захочет. Поэтому нам нужно "заткнуть" эту "дыру". Для этого достаточно исправить приведенное ранее выражение так, чтобы оно приняло вид:

```
$cfg['Servers'][$i]['auth_type'] = 'http';
```

После этого при попытке входа на phpMyAdmin он через Web-обозреватель запросит у нас имя пользователя и пароль. А это значит, что никто посторонний не сможет натворить дел в нашей базе данных.

Теперь можно сохранить файл config.inc.php и закрыть его. Настройка закончена.

Теперь перенесем всю папку phpMyAdmin-2.5.7-pl1 в корневую папку нашего Web-сервера — локального или удаленного (для этого можно воспользоваться программой клиента FTP). В последнем случае нам придется подождать — phpMyAdmin весьма велик.

Замечание

Папке phpMyAdmin-2.5.7-pl1 стоит дать более короткое имя, например, pma (как это сделал автор книги). Короткое имя проще набирать.

Использование

А здесь мы опишем использование phpMyAdmin для работы с базами данных MySQL. Мы научимся создавать базы данных, таблицы, поля и индексы и управлять правами доступа к ним.

Вход

Войти в phpMyAdmin очень просто. Для этого достаточно запустить Web-обозреватель и набрать в строке адреса **http://localhost:8080/phpMyAdmin-2.5.7-pl1**.

Внимание

Здесь подразумевается, что phpMyAdmin установлен на локальном Web-сервере, а папка, в которую он установлен, не была переименована. Если это не так, то нужно внести соответствующие изменения в приведенный ранее интернет-адрес.

После этого Web-обозреватель выведет на экран стандартное диалоговое окно ввода имени пользователя и пароля. Введем имя и пароль и нажмем кнопку **OK** этого окна. Если все было сделано правильно, через несколько секунд в Web-обозревателе появится главная страница phpMyAdmin, показанная на рис. П4.1.

Клиент данных phpMyAdmin имеет одну особенность — он автоматически опознает язык операционной системы и выводит свою главную страницу на этом языке. Так, если мы работаем под русской версией Windows, то получим главную страницу phpMyAdmin на русском языке.

Если мы установили MySQL на локальный компьютер, то должны будем использовать для входа на сервер пользователя `root` с "пустым" паролем. Если мы создаем базу данных на удаленном Web-сервере, то должны будем использовать имя и пароль, которые нам выделят.

В случае если мы ошибемся при вводе имени пользователя или пароля, Web-обозреватель снова выведет стандартное диалоговое окно ввода имени пользователя и пароля, и мы сможем ввести все правильно.

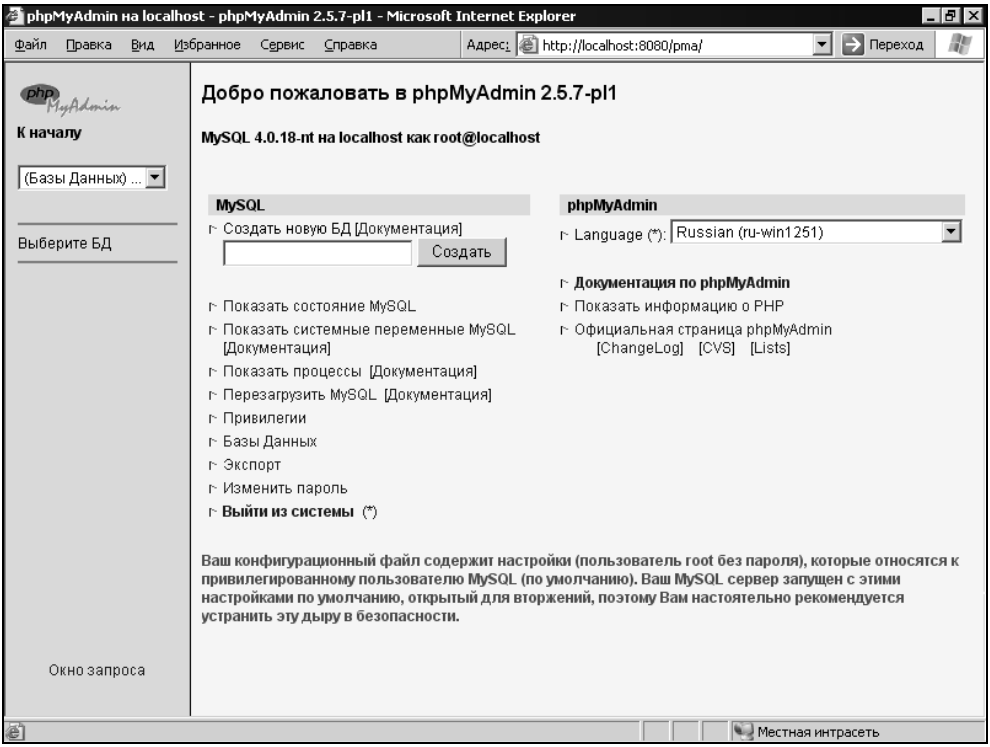


Рис. П4.1. Главная страница phpMyAdmin

Создание базы данных

Итак, мы находимся в главном окне phpMyAdmin (см. рис. П4.1). И нам пора приступать к созданию нашей базы данных.

Посмотрим в верхний левый угол страницы. Там должен быть раскрывающийся список баз данных, уже созданных на сервере и доступных для данного пользователя. (Если мы зашли под именем `root`, то нам доступны все базы данных.) Проверим, выбран ли в этом списке пункт **(Базы Данных)**, а если нет, выберем его.

Создать базу данных в phpMyAdmin проще простого. Введем имя создаваемой базы в поле ввода **Создать новую БД** и нажмем кнопку **Создать**. После этого созданная нами база данных появится в левой колонке страницы, а в правой ее части мы увидим два поля ввода: **Имя** (имя создаваемой таблицы) и **Поля** (количество таблиц).

Теперь можно приступать к созданию таблиц.

Создание таблиц

Чтобы создать таблицу, сначала нужно проверить, выбрана ли нужная нам база данных. Для этого посмотрим на левую колонку страницы phpMyAdmin — в раскрывающемся списке должен быть выбран пункт, имя которого совпадает с базой данных, в которую мы хотим поместить новую таблицу. Если же это не так, выберем его.

Все, мы выбрали базу данных. Вводим имя создаваемой нами таблицы в поле ввода **Имя** в правой части страницы. А в поле ввода **Поля** введем количество полей в этой таблице. После чего нажмем кнопку **Пошел**.

Новая таблица будет создана сразу же после этого, и phpMyAdmin выведет нам новую страницу, где мы сможем задать параметры ее полей.

Может получиться так, что на странице phpMyAdmin не будет ни поля **Имя**, ни поля **Поля**. Чтобы они появились, нужно выбрать в раскрывающемся списке в верхнем левом углу окна пункт **(Базы Данных)** и сразу же после этого — нашу базу данных. Упомянутые ранее поля тотчас появятся на странице, и мы сразу же сможем создать еще одну таблицу.

Создание полей

Когда мы создавали новую таблицу, то в поле ввода **Поля** ввели количество полей, которые должны в ней присутствовать. После этого phpMyAdmin выведет в правой части страницы форму в виде таблицы, в которой мы сможем задать параметры полей (рис. П4.2).

Поле	Тип [Документация]	Длины/Значения*	Атрибуты	Ноль	По умолчанию*
id	INT		UNSIGNED	not null	
name	VARCHAR	20		not null	
file	TINYINT	1		not null	

Рис. П4.2. Форма для задания параметров полей вновь созданной таблицы

Эта форма-таблица имеет следующие колонки:

- **Поле** — имя поля;
- **Тип** — раскрывающийся список для задания типа данных, которые будут храниться в поле;
- **Длины/Значения** — максимальная длина (строковых и целочисленных полей);
- **Атрибуты** — раскрывающийся список для задания дополнительных атрибутов поля. Для нас будут полезны пункт **UNSIGNED** (целочисленный

тип данных без знака) и "пустой" пункт (целочисленный тип данных со знаком);

- **Ноль** — раскрывающийся список, позволяющий задать, может ли поле принимать значение `NULL`. Доступны пункты **not null** (не может принимать; выбран по умолчанию) и **null** (может принимать);
- **По умолчанию** — значение поля по умолчанию;
- **Дополнительно** — раскрывающийся список, позволяющий создать поле счетчика (для этого достаточно выбрать пункт **auto_increment**);
- **Первичный** — переключатель, задающий создание ключевого индекса на основе данного поля;
- **Индекс** — переключатель, задающий создание обычного индекса на основе данного поля;
- **Уникальное** — переключатель, задающий создание уникального индекса на основе данного поля;
- **---** — переключатель, отменяющий создание индекса на основе данного поля;
- **ПолнТекст** — флажок, задающий создание полнотекстового индекса. Такой индекс содержит все слова, встречающиеся в значении данного поля, и может быть применен для поиска слов в полях. Полнотекстовый индекс может быть создан только на основе строковых полей.

К сожалению, раскрывающийся список **Тип** не содержит пункта **BINARY** — логический тип данных. Нам придется выбрать в нем пункт **TINYINT** и ввести в поле ввода **Длины/Значения** число 1.

Завершив ввод данных в форму-таблицу, нажмем кнопку **Сохранить**. После этого phpMyAdmin создаст новую таблицу, выведет ее имя в левой колонке страницы и автоматически выберет ее. При этом в правой части страницы мы увидим то, что показано на рис. П4.3.

Если же нам нужно добавить новое поле в уже созданную таблицу, нам сначала нужно будет ее выбрать. Для этого выберем в раскрывающемся списке в верхнем левом углу страницы пункт, соответствующей нашей базе данных. После этого ниже появится список таблиц, созданных нами в этой базе; щелкнем нужную таблицу, чтобы ее выбрать.

Выбрав таблицу, мы сможем ввести количество добавляемых полей в поле ввода **Добавить новое поле** внизу страницы и нажать кнопку **Пошел**. На странице появится форма-таблица, аналогичная показанной на рис. П4.2, в которую нам будет нужно ввести параметры добавляемого поля (полей). После этого останется нажать кнопку **Сохранить**.

Поле	Тип	Атрибуты	Ноль	По умолчанию	Дополнительно	Действие
id	int(10)	UNSIGNED	Нет		auto_increment	
name	varchar(20)		Нет			
file	tinyint(1)		Нет	0		

Отметить все / Снять отметку со всех *С отмеченными:*

Индексы : [Документация]

Имя ключа	Тип	Количество элементов	Действие	Поле
PRIMARY	PRIMARY	0		id

Создать индекс на 1 колонках

Пошел

Используемое пространство :

Тип	Использование
Данные	0 Bytes
Индекс	1,024 Bytes
Всего	1,024 Bytes

Статистика ряда :

Выражения	Значение
Формат	динамический
Ряды	
Далее Autoindex	
Создание	Окт 29 2004 г., 11
Последнее обновление	Окт 29 2004 г., 11

Рис. П4.3. Структура выбранной таблицы, отображаемая phpMyAdmin

Создание индексов

Индексы можно создать, включив соответствующие переключатели в форме-таблице задания параметров полей (см. рис. П4.2). Но такие индексы будут включать только одно поле. Если же нам нужно создать индекс, содержащий значения двух и более полей (например, индекс `namefile` в таблице `categories`), придется использовать другой способ.

Введем в поле ввода **Создать индекс на** количество полей, которые будут включены в индекс, и нажмем кнопку **Пошел**. Клиент данных phpMyAdmin выведет нам страницу с формой, с помощью которой выполняется создание индекса (рис. П4.4).

Имя создаваемого индекса задается в поле ввода, которое так и называется — **Имя индекса**. Если нужно создать ключевой индекс, туда нужно ввести строку `PRIMARY` (без кавычек).

Раскрывающийся список **Тип индекса** задает его дополнительные атрибуты. Это выполняется выбором одного из трех его пунктов:

- **INDEX** — простой индекс;
- **UNIQUE** — уникальный индекс;
- **FULLTEXT** — полнотекстовый индекс.

Ниже находится небольшая табличка, с помощью которой задаются входящие в индекс поля. Она имеет такие столбцы:

- **Поле** — раскрывающийся список, в котором выбирается поле;
- **Размер** — поле ввода для задания количества символов строкового поля, которое будет включено в индекс. MySQL позволяет создавать индексы,

содержащие не полные значения строковых полей, а их фрагменты. (Так, в частности, можно создавать индексы на основе полей типа blob.)

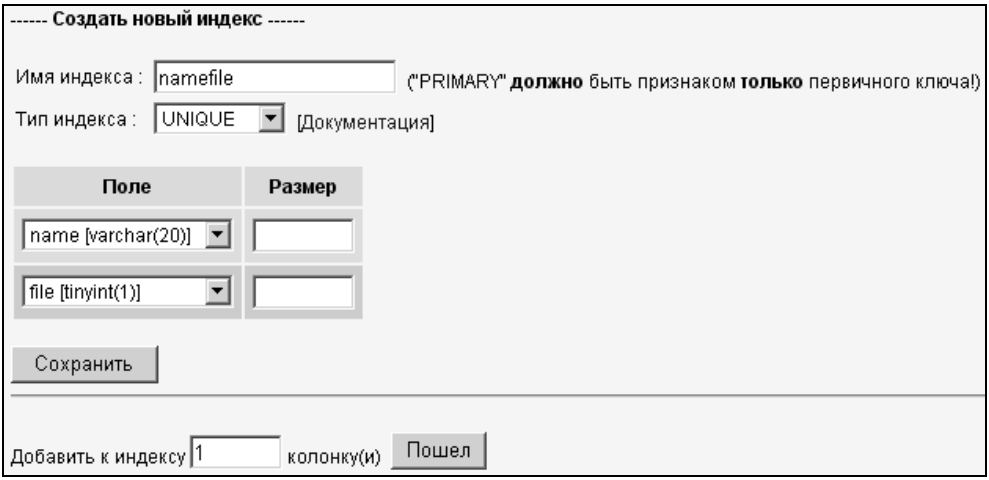


Рис. П4.4. Форма для создания нового индекса

Задав параметры индекса, нужно нажать кнопку **Сохранить**. После этого новый индекс будет создан, и сведения о нем появятся на странице, отображающей структуру таблицы (см. рис. П4.3).

Чтобы добавить новый индекс в уже существующую таблицу, сначала выберем в раскрывающемся списке в верхнем левом углу страницы пункт, соответствующий нашей базе данных. Ниже появится список таблиц, созданных нами в этой базе; найдем ту, что нам нужна, и щелкнем по ней. После этого на странице появятся сведения о структуре выбранной таблицы, а также необходимые нам поле ввода **Создать индекс на** и кнопка **Пошел**.

Правка и удаление полей, индексов, таблиц и баз данных

Увы — все люди совершают ошибки... Особенно, если они делают что-то в первый раз. И мы не исключение.

К счастью, phpMyAdmin предоставляет достаточно мощные средства для исправления ошибок. Мы можем исправить параметры любого поля, любого индекса и любой таблицы. Также мы можем удалить ненужное или лишнее поле, индекс или таблицу.

Правка и удаление полей

Чтобы изменить или удалить какое-либо поле таблицы, сначала выберем ее. Процесс все тот же: выбираем в раскрывающемся списке в верхнем левом

углу страницы базу данных, в которой находится эта таблица, и щелкаем по нужной нам таблице. После чего phpMyAdmin выведет на странице сведения о структуре этой таблицы.

Список полей и их параметров приведен в верхней части страницы. Он имеет вид таблицы, в колонках которой перечислены заданные нами при создании этого поля параметры.

Чтобы исправить какое-либо поле, щелкнем по значку , находящемуся в строке, соответствующей нужному полю. Клиент данных phpMyAdmin выведет форму-таблицу задания параметров поля (см. рис. П4.2), где мы сможем их изменить. Сделав нужные изменения, не забудем нажать кнопку **Сохранить**.

А удалить ненужное поле совсем просто. Для этого нужно щелкнуть по значку , находящемуся в соответствующей строке. На экране появится окно-предупреждение, спрашивающее, действительно ли мы хотим удалить это поле. Чтобы удалить его, нужно нажать кнопку **OK**; нажатие кнопки **Отмена** (Cancel) отменяет удаление поля.

Правка и удаление индексов

Чтобы изменить или удалить индекс таблицы, опять же, сначала выберем эту самую таблицу. Выберем в раскрывающемся списке в верхнем левом углу страницы базу данных и щелкнем по нужной нам таблице. После чего phpMyAdmin выведет на странице сведения о структуре этой таблицы.

Список индексов и их параметров приведен в нижней части страницы. Он имеет вид таблицы, в колонках которой перечислены заданные нами при создании этого индекса параметры.

Чтобы исправить какой-либо индекс, щелкнем по значку , находящемуся в строке, соответствующей нужному индексу. Клиент данных phpMyAdmin выведет форму-таблицу задания параметров индекса (см. рис. П4.4), где мы сможем их изменить.

Чтобы добавить поле в индекс, введем количество добавляемых полей в поле ввода **Добавить к индексу** и нажмем кнопку **Пошел**. В таблице полей появится еще одна строка, в которой мы сможем задать нужное поле.

А вот удаление ненужного поля из индекса выполняется довольно-таки неочевидно. В таблице, где перечислены поля, входящие в индекс, находим то, которое хотим удалить, и выбираем в раскрывающемся списке **Поле** пункт **-- Игнорировать --**. Все — теперь это поле удалено из индекса.

Как обычно, сделав нужные изменения, не забудем нажать кнопку **Сохранить**.

Ненужный индекс удаляется из таблицы так же, как и ненужное поле, — щелчком по значку , находящемуся в соответствующей строке. На экране появится окно-предупреждение, спрашивающее, действительно ли мы хо-

тим удалить этот индекс. Чтобы удалить его, нужно нажать кнопку **OK**; нажатие кнопки **Отмена** (Cancel) отменяет удаление индекса.

Правка и удаление таблиц

Правка таблиц заключается в добавлении, правке и удалении входящих в них полей и индексов. Как это делается, мы уже знаем.

А чтобы удалить ненужную таблицу, необходимо сначала выбрать ее (мы не будем описывать это процесс — он уже не раз был описан). Когда phpMyAdmin выведет сведения о структуре этой таблицы, найдем в верхнем правом углу страницы гиперссылку **Уничтожить**. (Она выделена ярко-красным цветом, так что не заметить ее очень трудно.) Если по ней щелкнуть, на экране появится окно-предупреждение, спрашивающее, действительно ли мы хотим избавиться от этой таблицы. Нажатие кнопки **OK** удаляет ее, а кнопка **Отмена** (Cancel) оставляет "в живых".

Правка и удаление баз данных

Правка баз данных заключается в добавлении, правке и удалении таблиц. Так что перейдем сразу к удалению.

Чтобы удалить ненужную таблицу, сначала необходимо выбрать в раскрывающемся списке в верхнем левом углу страницы пункт **(Базы Данных)**. Когда phpMyAdmin выведет свою главную страницу, щелкнем по гиперссылке **Базы Данных**. На экране появится страница с формой для управления базами данных (рис. П4.5).

Рис. П4.5. Форма для управления базами данных

Видно, что эта форма имеет вид таблицы. В самой левой ее колонке находится флажок, включение которого вовлекает соответствующую базу данных в выполняемую нами операцию (в нашем случае — удаление). После этого нам останется нажать кнопку **Пошел** под надписью **Удалить выбранные БД**.

Обычно в таких случаях на экране появляется хорошо нам знакомое окно-предупреждение Windows. Но сейчас phpMyAdmin решил немного разнообразить свое поведение и вывел страницу с формой, содержащей текст предупреждения и две кнопки — **Да** и **Нет**. Понятно, что нажатие кнопки **Да** удалит многострадальную базу данных, а **Нет** — отменит удаление.

Управление пользователями

Мы создали все нужные нам таблицы. Теперь самое время разобраться с пользователями и их правами.

Средства управления пользователями phpMyAdmin

Чтобы переключиться в список пользователей, нужно щелкнуть гиперссылку **К началу**, которая постоянно находится в верхнем левом углу страницы, выше раскрывающегося списка баз данных. После того как phpMyAdmin выведет главную страницу, щелкнем гиперссылку **Привилегии** и попадем прямо на страницу с формой для управления пользователями (рис. П4.6).

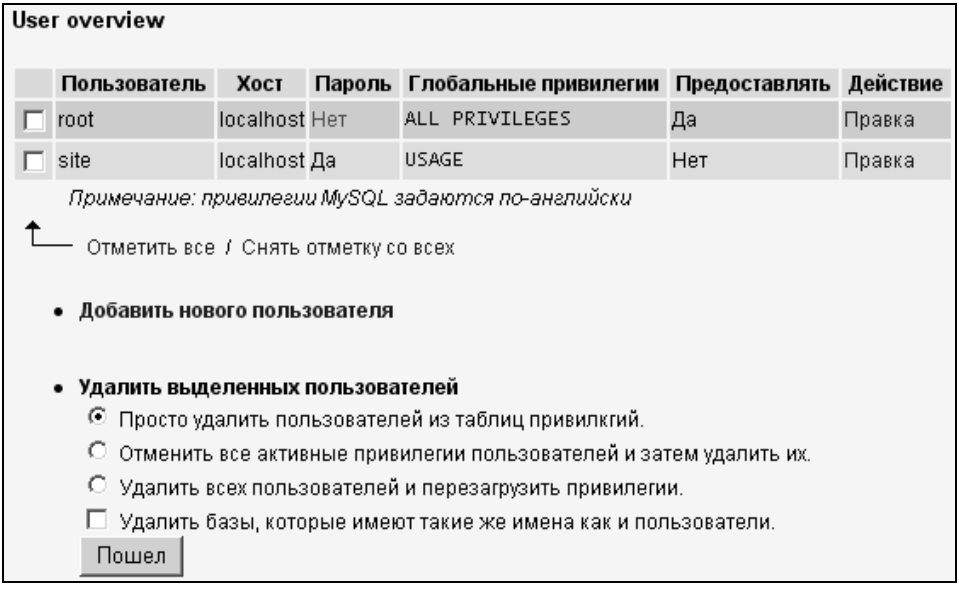


Рис. П4.6. Форма для управления пользователями

Эта форма содержит таблицу из нескольких колонок. Давайте их перечислим.

- Самая левая колонка содержит флажок, включение которого вовлекает данного пользователя в какую-либо операцию (например, удаление).
- **Пользователь** — имя пользователя. Если там стоит слово **Любой**, имеется в виду пользователь с любым именем. (В списке пользователей MySQL такой пользователь имеет имя `%`.)
- **Хост** — интернет-адрес компьютера, с которого данный пользователь имеет право подключаться к MySQL. Значок `%` обозначает любой интернет-адрес, то есть любой компьютер.
- **Пароль** — пароль пользователя. Если там стоит слово **Нет**, то пользователь имеет "пустой" пароль.
- **Предоставлять** — может ли данный пользователь управлять правами других пользователей (слово **Да**) или нет (слово **Нет**).
- **Действие** — гиперссылка **Правка**, ведущая на страницу правки параметров пользователя.

Сразу же после установки MySQL в его списке присутствуют четыре пользователя:

- `%@%` — пользователь с любым именем, который может подключиться с любого компьютера и имеет очень ограниченные права;
- `%@localhost` — пользователь с любым именем, который может подключиться с локального компьютера и имеет практически все права на создание и использование баз данных;
- `root@%` — пользователь с именем `root`, который может подключиться с любого компьютера и имеет права администратора;
- `root@localhost` — пользователь с именем `root`, который может подключиться с локального компьютера и имеет права администратора.

Все эти пользователи имеют "пустые" пароли. Но зачем они нужны?

Затем, чтобы мы смогли сразу после установки MySQL подключиться к нему и установить такие права доступа к базам данных, какие нам нужны. MySQL открывает нам "дверь", чтобы мы вошли и смогли поставить на нее "замок". Если же мы этого не сделаем, в "дверь" сможет войти кто угодно и натворить с нашими базами данных таких дел, что лучше об этом не думать.

Итак, первое, что нам нужно сделать, — это добавить в список пользователя с именем `site` и паролем `site`. Вторым нашим шагом будет удаление всех пользователей, кроме `root@localhost` и `site@localhost`. За работу!

Создание пользователя

Чтобы добавить нового пользователя, нам достаточно будет щелкнуть по гиперссылке с "говорящим" названием **Добавить нового пользователя**. В от-

вет на это phpMyAdmin выведет страницу с формой для задания параметров добавляемого пользователя (рис. П4.7), которую нам будет нужно заполнить.

Рис. П4.7. Форма для задания параметров пользователя

Для задания имени пользователя необходимо выбрать в раскрывающемся списке **Имя пользователя** пункт **Использовать текстовое поле** и ввести нужное имя в расположенное правее поле ввода. Чтобы ввести пользователя с любым именем (`%`), достаточно просто выбрать в этом списке пункт **Любой пользователь**.

Компьютер, с которого данный пользователь имеет право подключаться к MySQL, задается аналогично. Выбираем в раскрывающемся списке **Хост** пункт:

- **Любой хост** — для задания любого компьютера (`%`);
- **Local** — для задания локального компьютера (`localhost`);

- **Использовать текстовое поле** — для задания произвольного интернет-адреса. Нужный адрес вводится затем в расположенное справа поле ввода.

Теперь осталось задать пароль. Выбираем в раскрывающемся списке **Пароль** пункт **Использовать текстовое поле**, вводим нужный пароль в поле ввода, расположенное правее этого списка, а потом тот же пароль — в поле ввода **Подтверждение**. Если нужно ввести "пустой" пароль, то достаточно будет выбрать в списке **Пароль** пункт **Без пароля**.

Ниже всех этих списков и полей ввода расположен большой набор флажков, задающих права пользователя на доступ к серверу, то есть всем его базам данных. Эти флажки имеют названия, совпадающие с названиями аналогичных ключевых слов языка SQL (этот язык вкратце описан в *главе 6*). Так, чтобы дать пользователю возможность чтения данных из всех баз, нужно включить флажок **SELECT**.

Поскольку мы не собираемся давать нашему новому пользователю `site` возможность читать данные откуда то ни было, кроме базы `site`, то сразу нажмем кнопку **Пошел**. После этого phpMyAdmin выведет новую страницу, где мы сможем задать права доступа к отдельным базам данных (а также изменить права доступа к серверу, имя и пароль, если у нас будет такое желание). Эта страница, точнее, находящаяся на ней форма для задания прав доступа к отдельным базам данных, показана на рис. П4.8.

Рис. П4.8. Форма управления правами доступа к отдельным базам данных

Эта форма организована в виде таблицы с колонками, которые мы сейчас перечислим.

- **БД** — имя базы данных, которой даны особые права.
- **Привилегии** — список ключевых слов SQL, с помощью которых создаются разрешенные пользователю команды и запросы (`SELECT`, `INSERT`, `UPDATE`, `DELETE` и пр.).
- **Предоставлять** — может ли данный пользователь управлять правами других пользователей (слово **Да**) или нет (слово **Нет**).
- **Привилегии, специфичные для таблицы** — права, заданные для отдельных таблиц базы данных.
- **Действие** — гиперссылки **Правка** и **Отменить**. Вряд ли нужно объяснять, что они делают.

Изначально эта форма-таблица пуста, так как мы еще не задали никаких прав для доступа к базе данных `site`. Поэтому выбираем в раскрывающемся

списке **Добавить привилегии на следующую базу** пункт **site** (нашу базу данных) и сразу же попадаем на страницу задания прав доступа к базе данных (рис. П4.9).

Рис. П4.9. Форма задания прав доступа к отдельной базе данных

Здесь мы снова видим набор флажков, задающих права на выполнение соответствующих запросов SQL. Для нашего пользователя `site` мы должны включить все флажки в группе **Данные** и флажок **LOCK TABLES** в группе **Администрирование**. Почему? Сейчас мы об этом поговорим.

Флажки в группе **Данные** дают пользователю возможность выполнять запросы на получение данных, добавление, правку и удаление записей. Это флажки **SELECT**, **INSERT**, **UPDATE** и **DELETE**, уже знакомые нам по *главе 6*. Поскольку пользователь `site` будет работать с данными, он должен иметь права на выполнение всех этих запросов.

Группа флажков **Структура** разрешает или запрещает выполнение запросов SQL на создание баз данных, таблиц и индексов, а также на их правку и удаление. Нашу базу данных `site` со всеми ее таблицами и индексами мы создали от имени пользователя `root` — администратора сервера, имеющего права абсолютно на все. Пользователю же `site` не придется создавать таблицы и индексы — он будет работать только с хранящимися в них данными. А значит, права на создание, правку и удаление всех этих баз данных, таблиц и индексов ему не пригодятся.

Группа флажков **Администрирование** дает права на выполнение служебных операций. Из этой группы нам понадобятся права только на выполнение запроса `LOCK TABLES` (блокировку таблиц), поэтому нам нужно включить только этот, одноименный, флажок.

Замечание

Ниже группы флажков задания прав доступа к базе данных находится форма-таблица, с помощью которой можно управлять правами доступа к отдельным таблицам этой базы. Она аналогична форме, показанной на рис. П4.8, и используется так же. Сами же права доступа к таблице задаются с помощью формы, похожей на форму задания прав доступа к базе данных (см. рис. П4.9).

Права заданы. Нажимаем кнопку **Пошел**. phpMyAdmin посылает MySQL соответствующие запросы SQL, а MySQL выполняет все операции по заданию прав. После этого ищем в верхней части страницы гиперссылку **Привилегии**, щелкаем по ней и попадаем на страницу управления пользователями (см. рис. П4.6).

Правка и удаление пользователей

Теперь нам нужно удалить всех пользователей, кроме **root@localhost** и **site@localhost**. Сделать это проще простого: включаем соответствующие флажки в левой колонке формы-таблицы управления пользователями (см. рис. П4.6) и нажимаем кнопку **Пошел**.

Внимание

Удаление пользователей, а также прав доступа к отдельным базам данных и таблицам выполняется без предупреждения.

Изменить же параметры какого-либо пользователя можно, щелкнув гиперссылку **Правка** в колонке **Действие** формы-таблицы управления пользователями. Правка параметров пользователя выполняется с помощью хорошо знакомых нам форм, показанных на рис. П4.7—П4.9.

Работы с данными

К сожалению, phpMyAdmin не содержит удобных средств для работы с самими данными: добавления, правки и удаления записей, а также их выборки. Точнее, кое-какие средства у него все-таки имеются, но нам самим придется вводить нужные запросы SQL в соответствующую форму.

Итак, выбираем в раскрывающемся списке в верхнем левом углу страницы нашу базу данных. После того как phpMyAdmin выведет список таблиц, находим в верхней части страницы гиперссылку **SQL** и щелкаем по ней. Все — форма для ввода запросов SQL перед нами (рис. П4.10).

Здесь все просто. Вводим код SQL-запроса в большую область редактирования и нажимаем кнопку **Пошел**. Клиент данных phpMyAdmin передает этот запрос MySQL и принимает от него подтверждение об успешном его выполнении или сообщение об ошибке. Поскольку введенный нами запрос на добавление записи (см. рис. П4.10) не содержит ошибок, он будет выполнен.

Выполнить SQL запрос(ы) на БД [Документация] :

```
insert into categories (name, file) values ("Интернет", 1);
```

☑ Показать данный запрос снова

Или Месторасположение текстового файла :

Обзор... (Maximum size: 2,048KB)

Сжатие: ◉ Автодетект ○ Нет ○ архивировать в gzip

Пошел

Рис. П4.10. Форма для ввода запросов SQL

Таким образом мы сможем добавить в таблицы созданной нами базы данных `site` несколько отладочных записей, на которых мы потом будем отлаживать наши серверные страницы. Также мы можем выполнить любой запрос на извлечение данных (`SELECT`), изменение (`UPDATE`) и удаление (`DELETE`) записей.

В принципе, с помощью этой формы можно выполнить любой запрос SQL, поддерживаемый сервером MySQL: создания таблиц, индексов, пользователей, управления правами и пр. Хотя, конечно, для этих целей phpMyAdmin предоставляет другие, более удобные инструменты.

Выход

После завершения работы с базами данных и пользователями нужно выполнить операцию выхода. При этом phpMyAdmin разорвет все соединения с сервером данных и освободит оперативную память компьютера.

Чтобы выйти из phpMyAdmin, нужно щелкнуть гиперссылку **К началу**, которая присутствует в верхнем левом углу любой его страницы. После этого phpMyAdmin выведет свою главную страницу, на которой нужно щелкнуть гиперссылку **Выйти из системы**. Web-обозреватель выведет на экран стандартное диалоговое окно ввода имени пользователя и пароля, в котором нам будет нужно просто нажать кнопку **Отмена** (Cancel).

Если после этого в окне Web-обозревателя появится строка **Ошибочный логин/пароль. В доступе отказано.**, значит, операция выхода была выполнена успешно.

Другие программы клиентов данныхMySQL

Программа phpMyAdmin — не единственный доступный клиент для работы с базами данных MySQL. Существуют и другие программы, которые мы здесь рассмотрим.

В *приложении 2* этой книги упоминалась программа MySQL Control Center. Это весьма мощный клиент данных, с помощью которого можно выполнить операции по созданию баз данных, таблиц и индексов, а также "наполнению" их данными. В частности, отладочные данные удобнее всего заносить в таблицы MySQL именно с помощью MySQL Control Center.

MySQL Control Center — это обычная программа Windows, не требующая для работы никаких дополнительных программ (за исключением, разумеется, самого сервера MySQL). Ее дистрибутив можно найти на сайте MySQL AB (**http://www.mysql.com**). На момент написания этой книги была доступна версия 0.9.4.

Существует еще одна программа клиента данных MySQL — EMS MySQL Manager. Она имеет практически те же возможности, что и MySQL Control Center, плюс некоторые дополнительные и обладает несколько более привлекательным интерфейсом. Найти эту программу можно на сайте группы разработчиков **http://www.mysqlmanager.com**. У автора этой книги имеется только старая версия 1.9, наверняка уже существуют более новые.

Однако у обеих этих программ имеется один большой недостаток. Дело в том, что для их нормальной работы на компьютер нужно будет установить также клиентскую часть сервера MySQL. А эта самая клиентская часть использует для связи с сервером MySQL порт 3306 (о портах TCP/IP см. *главу 1*). Если мы "общаемся" с сервером, установленным на локальный компьютер, то никаких проблем не будет. Но если мы попытаемся подключиться к удаленному серверу, то можем столкнуться с тем, что интернет-провайдер заблокировал этот порт в целях безопасности. (Это, кстати, обычная практика — блокировать неиспользуемые порты.)

В случае phpMyAdmin такой проблемы нет. Как мы уже знаем, он представляет собой набор серверных Web-страниц PHP, устанавливается на том же сервере, что и MySQL, и "общается" с пользователем через порт 80, доступ к которому всегда открыт. К тому же, как правило, phpMyAdmin уже установлен на тех удаленных Web-серверах, что предоставляют клиентам возможность использования MySQL, поэтому навыки работы с ним пригодятся нам в дальнейшем.

Предметный указатель

"

"Живой" просмотр 197

A

ARPANET 8
ASP 124

C

CGI 125
ColdFusion 124
Cookie 332
CSS 38

D

DNS 17

F

FTP 9, 14
  пассивный 111

G

GIF 33

H

HTML-редактор 54
HTTP 14

I

IP-адрес 16
ISAPI 125

J

JDBC 139
JPEG 33
JSP 124

N

NSAPI 125
NULL 156

O

ODBC 139

P

PHP 123
PNG 33
POP3 14

R

RGB 65

S

SFTP 112
Shockwave/Flash 34
SMTP 14, 371
SQL 140

T

TCP/IP 13

W

W3C 31
Web-обозреватель 18
  примеры программ 21
  история 37
Web-редактор 54
  визуальный 54
  гибридный 55
  невизуальный 54
Web-сайт 18, 49
  локальная копия 94
  планирование 49
  полезное содержимое 51
  регистрация 93, 197
  удаленная копия 94
Web-сервер 18
  примеры программ 23
  расширение 125
Web-страница 18, 28
  административная 241
  активная серверная 123
  архив новостей сайта 51
  главная 51
  динамическая 122
  закрытие 71
  карта сайта 52
  корректная 36
  название 37
  начальная 51
  новости сайта 51
  открытие 81, 99
  по умолчанию 19
  пользовательская 241
  правильно оформленная 36
  просмотр 89
  результатов поиска 271
  сведения о контакте 52
  сведения о разработчиках 51
  создание 58, 99
  сообщения об ошибке 404, 318
  сохранение 60
  статическая 119
  текущая 20, 34
World Wide Web Consortium 31
WWW 9
WWWC 31

X

XHTML 31

А

Автономный режим 21
Администратор 19, 153
Активный документ 85
Аргумент 149, 163, 185
  необязательный 186
  регистрация 254
  фактический 186
  формальный 185
Архитектура 13
  двухзвенная 13
  клиент-сервер 13
  однозвенная 13
  трехзвенная 122
Атрибут:
  без значения 256
  индекса 158

необязательный 32
обязательный 32
поля 156
прав 156
стиля 39
тега 32

Б

База данных 127
нереляционная 128
регистрация 199
реляционная 127
Блок 177
Блокировка 311
Брандмауэр 112

В

Вложенность тегов 29
Выражение 164, 171, 177, 179

Г

Гиперссылка 34
внешняя 103
мертвая 102
оборванная 102
почтовая 83
создание 82, 103
цель 83
Гость 241
Граница таблицы 73
Группа новостей 9
Группировка 148

Д

Декремент 169
Диалоговое окно
Attach External Style Sheet 87
Check New Username 375
Choose Home Page 97
CSS Style Definition 86
Define Access Levels 293
Delete Record 263
Dependent Files 105
Display Total Records 233
Dynamic CheckBox 254
Dynamic Data 255
Dynamic List/Menu 268
Email Link 83
Insert Record 246
Log In User 284
Log Out User 297
Manage Sites 93
MySQL Connection 200
New CSS Style 85
New Document 58
Please Provide a Test Value 219
Preferences 55
Recordset 205
Repeat Region 209
Reports 100
Restrict Access To Page 292
Select Database 201
Select File 82
Select Image Source 79
Server Compatibility 111
Show If Recordset
Is Not Empty 230, 231
Split Cell 78
Synchronize Files 106
Table 71
Test SQL Statement 206
Update Files 99
Update Record 258
URL Variable 254
Домен 16
Доменная зона 16
Доменное имя 16
Доступ 153

З

Запись 129
Запрос клиентский 11

Запросы SQL 152
Значение:
  аргумента по умолчанию 186
  атрибута 32, 39
  поля 129
  поля по умолчанию 131
  пункта списка 269
  стиля 39

И

Идентификатор 203
  папки 348
  соединения 371
Иерархия 37
Изображение 33
Имя:
  индекса 135
  кодировки 47
  константы 193
  набора записей 205
  переменной 165
  пользователя 109
  поля 129
  соединения 200
  стиля 39
  таблицы 129
  формы 244
  функции 185
  элемента управления 237
Индекс 132
  внешний 137
  ключевой 133
  простой 133
  сложный 133
  суррогатный 134
  уникальный 158
Инкремент 169
Инструментарий
  документа 60
  панели Files 97
Интернет 7
  сервис 9
Интернет-адрес 15, 20
Интернет-пейджер 9
Интернет-провайдер 8
Интранет 24
Информация 118

К

Каскадная таблица стилей 38
Клиент 10
  данных 137
Клиентская часть сервера 139
Ключ 133
Ключевое слово 140, 164
Кнопка:
  выключатель 62
  переключатель 61, 67
Код:
  ошибки 318
  символа 47
Кодировка 47
Кодовая таблица 47
Комментарий 194
Компьютер
  клиентский 10
  серверный 10
Константа 193
  встроенная 194
Критерий 133
  группировки 148
  связывания 146
Кэш 21

Л

Лист рассылки 365
Логин 109
Локальный хост 95

М

Маркер 67
  изменения размера 80
Массив 190 192
Метаданные:
  таблицы 130

Метод:
кодирования данных 238
передачи данных 215, 239, 240

Н

Набор записей 204
Новости 9

О

Область:
необязательная 230
повторяющаяся 207
Обработчик 123
Обсуждение 9
Объединение ячеек 77
Окно:
выбора цвета 65
документа 58
списка синхронизируемых файлов 107
справки 89
Оператор 163, 168
арифметический 163, 169
вывода 161
комментария 194
логический 145, 172
объединения строк 170
приоритет 175
простого присваивания 165, 170
сложного присваивания 170
сравнения 144, 171
условный 179
Оптимизация 304
Ответ серверный 11
Ошибка 404, 102

П

Пакет 13
Палитра 65, 66
Панель 61
Bindings 204
CSS Styles 84
Databases 199
Files 97
Link Checker 103
Reference 91
Server Behaviors 209
Site Reports 101
закрытие 102
Папка:
виртуальная 19
корневая 18
создание 100
тестовая 199
Переменная 164
SQL-запроса 219
глобальная 188, 291
локальная 188
объявление 165
статическая 189
уровня сессии 287
уровня страницы 188
Письмо 370
Поведение серверное 209
Подтверждение 12
Поисковая машина 271
Поле 129
индексированное 132
ключевое 133
обязательное 131
скрытое 248, 250, 257
уникальное 131
Поле ввода:
пароля 283
файла 340
Полоса навигации 51
Порт 15
Потоковое вещание 9
Почтовая рассылка 365
Права 153, 154
Правила 131
Представление 118
Приложение HTML 343
Приоритет 145

Проверка:
HTML-кода 100
гиперссылок 102
Программа серверная 122
Прокси-сервер 112
Протокол 13, 14
Процессор данных 137
универсальный 139
Псевдоним поля 147
Публикация 24

Р

Разрыв строк 70
Редактор свойств 61
Режим отображения 60
Результат 164, 185
Рунет 47

С

Связь 136 137
Сеанс 12
Секция:
Web-страницы 37
строки статуса тегов 62
Селектор цвета 65
Сервер 10
Сервер данных 138
Сессия 287
Символ:
служебный 47
специальный 69, 167
Синхронизация 106
Система управления базами данных 127
Служба 403
Служба рассылок 366
Соединение 12
Сообщение об ошибке 11
Спам 365
Список HTML 67
Справочная система 89
Ссылочная целостность 280
Стиль 39
встроенный 42
гибридный 42
конфликт 44
переопределения тега 39
создание 85
стилевой класс 40
Строка статуса 62
Структура сайта логическая 50
СУБД 127, 128, 138
Сценарий 123, 161
Счетчик цикла 181

Т

Таблица:
вторичная 136
дочерняя 136
первичная 136
родительская 136
создание 71
структура 130
Таблица стилей 38
внешняя 42
внутренняя 39
Таймаут 287
Тег 28, 36, 46
видимый 36
дочерний 38
закрывающий 28
логического форматирования 46
невидимый 36
одинарный 28
открывающий 28
парный 28
потомок 38
родительский 38
содержимое 28
физического форматирования 46
Текст динамический 207
Текст-замена 32
Тело:
функции 185

цикла 181
Тип MIME 342
Тип данных 129
memo 130
NULL 168
логический 166
преобразование 173
приведение 174
с плавающей точкой 166
совместимость 173
строковый 167
счетчик 130
целочисленный 166
Триггер 137

У

Условие 177

Ф

Файл:
заголовок 138
локальной
конфигурации 319
Файлообменная сеть 13
Фильтрация 133
Форма 122, 236
Формат Unix 333
Функция 185
агрегатная 148
встроенная 189
вызов 186
объявление 185

Х

Хостинг-провайдер 25
Хранимая процедура 137
Хэш 192

Ц

Целостность 280
Цель гиперссылки 35
Цикл 181
бесконечный 183
перезапуск 184
прерывание 184
просмотра 192
с постусловием 182
с предусловием 183
со счетчиком 181

Ч

Чат 10

Ш

Шаблон 59, 221
Шрифт стандартный 64

Э

Элемент Web-страницы 31
внедренный 32
нетекстовый 71
текстовый 32