Ларри Ульман

MySQL

QUICK START

VISUAL QUICKSTART GUIDE

MYSQL

Larry Ullman

Peachpit Press

QUICK START

MYSQL

Ларри Ульман

ДМК Пресс
издательство
Москва, 2003

УДК 004.65
ББК 32.973.26-018.2
У51

Ульман Л.

У51 MySQL: Пер. с англ. – М.: ДМК Пресс, 2003. – 352 с.: ил. (Quick Start).

ISBN 5-94074-229-7

Реляционная система управления базами данных MySQL разработана и до сих пор поддерживается шведской компанией MySQL AB. На сегодняшний день MySQL – одна из самых распространенных СУБД с открытыми исходными кодами. Это означает, что за рядом мелких исключений ей можно пользоваться бесплатно, а кроме того, модифицировать исходный код, который доступен в сети Internet.

В данной книге рассматривается установка MySQL в операционных системах Windows, Linux, Mac OS; подробно описываются запуск СУБД и работа с ней, причем основное внимание уделяется доступу к базе данных и администрированию из командной строки. Ряд глав посвящен программированию на языках PHP, Perl и Java. В тексте приводятся упражнения, облегчающие изучение MySQL начинающими пользователями.

В приложениях рассматриваются вопросы диагностики и устранения ошибок, приводятся справочная информация и ссылки на другие источники.

ISBN 0-321-12731-5 (англ.)

ISBN 5-94074-229-7 (рус.)

СОДЕРЖАНИЕ

Посвящается Джесс, моей любимой и единственной.

ПРЕДИСЛОВИЕ

Благодарности

Выражаю искреннюю благодарность всему замечательному коллективу издательства Peachpit Press за их стремление выпускать высококачественную литературу, за предоставленную мне возможность опубликовать эту книгу и вообще за все, что они делают. Отдельное спасибо Нэнси Олдрич-Рюнцель (Nancy Aldrich-Ruenzel), Марджори Бэер (Marjorie Baer), Киму Ломбарди (Kim Lombardi), Гэри Полу Принсу (Gary-Paul Prince) и еще двум десяткам людей, чьих имен я не знаю, хотя и должен был бы.

Благодарю редактора этой книги Ребекку Гулик (Rebecca Gulick). Работа с ней была истинным удовольствием, и книга благодаря этому стала только лучше. Мне повезло, что нам вновь привелось потрудиться вместе.

Благодарю Бренду Беннер (Brenda Benner) за тщательную корректуру и внимание к деталям.

Я глубоко признателен Конни Юнг-Миллс (Connie Jeung-Mills) за то, что она превратила разрозненный набор текстовых и графических файлов в пригодный для чтения материал.

Спасибо Оуэну Вульфсону (Owen Wolfson), отвечавшему за общую структуру книги, Карин Арригони (Karin Arrigoni), составлявшей алфавитный указатель, и Адаму Нелсону (Adam Nelson) – научному редактору.

Спасибо Джону О'Мэлли (John O'Malley), блестящему программисту и вообще очень толковому парню, за помощь при работе над главой, посвященной Java. Джон хоть и не в числе самых старых, но все равно в десятке моих самых любимых друзей.

Как всегда, у меня есть немало поводов выразить отдельную благодарность всем сотрудникам компании DMC Insights, Inc.

Благодарю читателей моих предыдущих книг, которые нашли время сообщить мне свое мнение (даже если оно сопровождалось просьбой о технической поддержке) и просили меня написать книжку про MySQL. Надеюсь, вы теперь получите именно то, что хотели.

Ну и, наконец, спасибо всем членам сообщества пользователей и разработчиков MySQL – начиная с сотрудников компании MySQL AB и заканчивая участниками различных списков рассылки. MySQL – это еще один пример того, какую огромную пользу может принести программа с открытым исходным кодом.

Соглашения

Полужирным шрифтом в этой книге отмечены пункты меню и прочие элементы интерфейса, а также клавиши, например:

New User (Новый пользователь), клавиша **Return**.

Сочетания клавиш, которые нужно нажимать одновременно, приводятся со знаком + (плюс), например: **Ctrl**+**A**.

Команды, которые требуется выполнить последовательно, записываются через стрелочку, например: **Start** ⇒ **Run** (Пуск ⇒ Выполнить).

Перенос строки кода в тексте обозначается стрелочкой ⇨, например:

```
SELECT * FROM имя_таблицы
⇨ ORDER BY RAND();
```

Моноширинным шрифтом выделяются фрагменты кода, например: `mysqladmin -u root password 'пароль'`. *Моноширинным курсивом* отмечены фрагменты кода, вместо которых следует подставить какое-либо значение (в данном примере взамен слова *пароль* необходимо ввести пароль).

Адреса сайтов в Internet выделены подчеркиванием, например: www.mysql.com.

Таким образом оформлены примечания, содержащие дополнительную информацию по теме раздела.

Помимо примечаний в конце разделов приводятся советы по работе с MySQL.

ВВЕДЕНИЕ

В середине информационной эпохи, когда все больше данных хранится в памяти компьютеров, возрастает потребность в надежных, обеспечивающих быструю работу базах данных. На протяжении многих лет такие компании, как Oracle, поставляли на рынок приложения для организации хранилищ данных, пользователями которых были главным образом крупнейшие компании из списка Fortune 500, способные заплатить за программное обеспечение и высококвалифицированный персонал, необходимый для его эксплуатации, в целях решения критических для бизнеса задач. А тем временем сообщество разработчиков программ с открытым исходным кодом (которое окончательно сформировалось в течение последних десяти лет) выпустило новое поколение компактных, надежных и недорогих СУБД. Такие продукты, как mSQL, MySQL и PostgreSQL, предоставили реальный выход из ситуации обычным пользователям и разработчикам, не располагающим большими деньгами.

На данный момент MySQL превратилась из скромной экспериментальной программы в надежную, устойчивую и простую в администрировании СУБД – и, что еще более удивительно, сохранила свою открытую природу: по-прежнему предлагается для использования и модификации совершенно бесплатно. Богатство возможностей MySQL объясняет, почему с ней работают такие крупные организации, как Yahoo!, Бюро переписей США и NASA. Низкая стоимость и свободное распространение позволят вам использовать MySQL в своих проектах. Книга, которую вы держите в руках, поможет вам в этом!

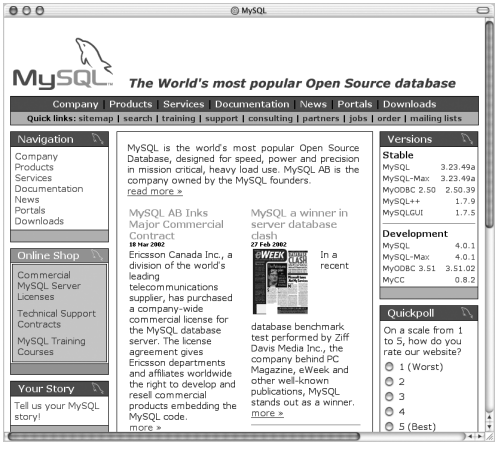

Рис. 1. Домашняя страница MySQL, расположенная по адресу www.mysql.com. Здесь можно загрузить программное обеспечение, почитать документацию и получить много другой информации

Что такое MySQL?

MySQL – это самая популярная (а некоторые добавят, что еще и самая лучшая в мире) СУБД с открытым исходным кодом. На самом деле MySQL составляет все более значительную конкуренцию таким дорогостоящим гигантам, как Oracle и Microsoft SQL Server.

MySQL создана и до сих пор поддерживается шведской компанией MySQL AB (ее сайт расположен по адресу www.mysql.com – см. рис. 1). В какой-то мере MySQL выросла из разработанной ранее СУБД mSQL, которая по-прежнему существует, хотя пик ее популярности уже в прошлом. Как ни странно, но ни в этой книге, ни в какой-либо другой вы не найдете объяснения тому, что означает аббревиатура MySQL. Что касается последней части, «SQL», тут все ясно: это Structured Query Language (структурированный язык запросов) – язык, применяемый для взаимодействия с большинством существующих баз данных. Но вот префикс «My» никак официально не расшифровывается даже в самой компании MySQL AB.

MySQL – это реляционная система управления базами данных (РСУБД). С технической точки зрения MySQL – программа, управляющая файлами, которые составляют базу данных, но часто термин «база данных» (БД) применяется и к самой программе, и к этому набору файлов. БД – это просто совокупность взаимосвязанных данных (текстовых, числовых, двоичных), за хранение и организацию которых отвечает СУБД.

О произношении

Сразу стоит уточнить, что MySQL произносится «май эс кью эль».

Есть много видов баз данных: от простейших, где данные хранятся в *плоских файлах*, до реляционных и объектно-ориентированных. *Реляционная база данных* содержит информацию в нескольких таблицах. Эта концепция была разработана в начале семидесятых годов, а до того базы данных напоминали одну огромную электронную таблицу, где хранится буквально все. Для проектирования и программирования реляционной БД требуется больше усилий, но это окупается повышенной надежностью работы и целостностью данных.

MySQL – это программа с открытым исходным кодом, равно как PHP и некоторые версии операционной системы UNIX. Это означает, что вы можете бесплатно устанавливать, запускать программу и модифицировать ее исходный код (который, как и ее саму, можно загрузить из Сети). Но в некоторых случаях вы все же должны заплатить за лицензию на MySQL и, в частности, тогда, когда получаете прибыль от включения MySQL в состав своего продукта. Подробнее об этом вы можете прочесть в документе об условиях лицензирования MySQL.

MySQL состоит из нескольких частей, в том числе сервера MySQL (программы mysqld, которая, собственно, и управляет базой данных), клиента MySQL (программы mysql, предоставляющей интерфейс к серверу) и многочисленных служебных утилит для обслуживания базы данных и иных целей. Работу с MySQL можно вести, пользуясь многими распространенными языками программирования, включая PHP, Perl и Java. Впоследствии вы узнаете, как это делается.

MySQL написана на языках C и C++ и работает под управлением различных операционных систем. Есть примеры применения MySQL для поддержания баз данных, состоящих из 60000 таблиц, насчитывающих более 5 млрд. строк. В некоторых операционных системах MySQL начиная с версии 3.23 может работать с таблицами объемом до 8 млн. Тб (терабайт), а в общем случае предельный объем составляет 4 Гб.

На момент написания этой книги текущей была не так давно вышедшая версия 4.0, а вскоре готовится выпуск версии 4.1. В четвертой версии технология MySQL вышла на качественно новый уровень и включила целый ряд возможностей, которых разработчики добивались на протяжении нескольких лет. В тексте я постараюсь отмечать, какие функции специфичны именно для версии 4 и отсутствуют в более ранних.

Основные ресурсы, относящиеся к MySQL

Поскольку MySQL – бесплатная программа, то на ваши вопросы отвечают преимущественно участники списков рассылки, посвященных MySQL, а не сотрудники компании MySQL AB. Однако компания готова заключить платный контракт на поддержку, консультирование и обучение. Этот вариант может заинтересовать вас, если вы намерены использовать MySQL в масштабе предприятия.

По MySQL имеется обширное онлайновое справочное руководство (по адресу www.mysql.com/documentation/ – см. рис. 2), включающее примечания, написанные пользователями. В нем можно производить поиск, и на большую часть своих вопросов вы найдете там ответы. На момент работы над этой книгой руководство охватывало версии вплоть до 4.1.0-alpha.

Списки рассылки по MySQL, возможно, не так активно действуют, как некоторые другие, но в них принимают участие как пользователи, так и разработчики MySQL, которые готовы быстро и информативно ответить на задаваемые вопросы. Имеются также специализированные списки рассылки, посвященные работе MySQL на платформе Windows, доступу к MySQL из программ на языке Java и другим вопросам. Большая популярность этих списков объясняет тот факт, что MySQL не посвящены сетевые конференции, подобные тем, где обсуждаются другие технологии.

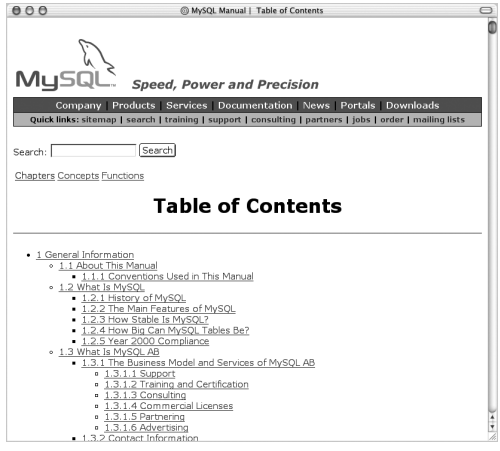

Рис. 2. Онлайновое руководство по MySQL очень подробно и хорошо организовано. Вы можете также загрузить на компьютер это руководство в других форматах

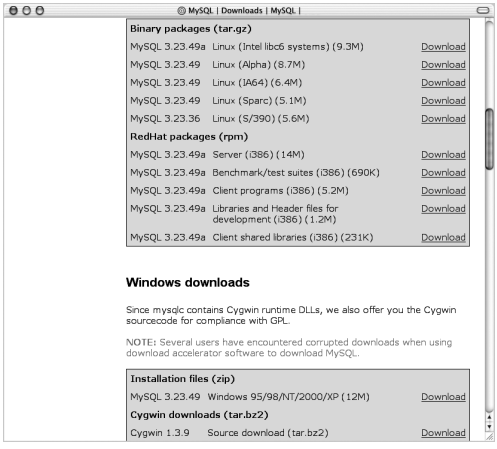

Рис. 3. Поставляются версии MySQL практически для любой операционной системы, включая Windows и различные клоны UNIX

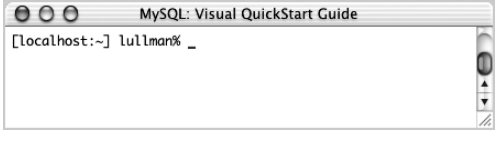

Рис. 4. В составе операционной системы Mac OS имеется приложение X Terminal, позволяющее общаться с компьютером при помощи командной строки

Технические требования

Чтобы вы могли выполнять приведенные в этой книге упражнения, ваш компьютер должен удовлетворять нескольким требованиям, впрочем, не слишком обременительным. Прежде всего, конечно, понадобится установить MySQL. К счастью, эта программа распространяется бесплатно и работает в большинстве операционных систем (рис. 3). В главе 1 будет описана процедура установки MySQL в трех широко распространенных ОС: Windows, Linux и Mac OS.

В тексте книги упор сделан главным образом на доступ к базе данных и администрирование из командной строки. Как именно осуществляется доступ к командной строке, зависит от операционной системы. В Windows это окно DOS, в Linux – интерпретатор команд (shell), а в Mac OS – окно X-терминала (рис. 4).

И наконец, для проработки глав, посвященных программированию на языках PHP, Perl и Java (главы 6, 7 и 8 соответственно), понадобятся текстовый редактор, Web-браузер и т.п. Конкретные требования будут сформулированы по ходу дела.

Об этой книге

В этой книге я предпринял попытку рассказать об основах MySQL, включив сюда сведения, которые могут понадобиться большинству пользователей. В духе серии «Visual QuickStart Guide» обучение ведется пошагово, причем каждый этап работы проиллюстрирован. Изложение материала ориентировано скорее на сообщение конкретных фактов, нежели на углубленное изучение методов разработки сложных приложений с использованием MySQL (как было бы в книге, посвященной программированию).

Большинство собранных здесь примеров предполагает работу с командной строкой (рис. 5), хотя в трех главах, посвященных программированию, а также в главе 9 дело обстоит иначе. Там я подробно остановлюсь на написании сценариев, которые затем будут протестированы с помощью подходящих средств (например, Web-браузера).

Последовательность изложения материала практически линейна. Мы начинаем с описания процедур установки и администрирования. Далее будут рассмотрены вопросы проектирования баз данных и язык SQL: сначала аспекты, общие для всех баз данных, а затем – свойственные только MySQL. Вслед за этим идут три главы, посвященные программированию, а также глава, в которой описываются некоторые специальные приемы программирования баз данных. Под конец предлагается описание ряда утилит, о которых вы должны знать. В приложениях рассматриваются вопросы диагностики и устранения ошибок, приводятся справочная информация и ссылки на другие источники.

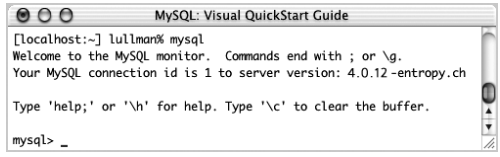

Рис. 5. В большинстве глав демонстрируется доступ к базе данных из стандартного клиента MySQL или других утилит, интерфейс которых выполнен в виде командной строки

Для вас ли эта книга?

Эта книга рассчитана на широкий круг читателей – от начального до среднего уровня. Ясно, что в какой-то мере она может пригодиться любому, кто желает применять в своей работе MySQL, но я специально ориентировался на пользователей, которые раньше вообще не работали с базами данных или работали с другой базой и хотели бы знать, как ведет себя MySQL, а также на тех, кто уже освоил эту систему, но хотел бы повысить свою квалификацию. Сразу подчеркну, что я не ставил себе задачей обучить читателя какому-либо языку программирования, а хотел лишь показать, как из программ на некоторых языках можно получить доступ к MySQL.

Сопроводительный Web-сайт

Специально для этой книги я создал Web-сайт, расположенный по адресу www.DMCinsights.com/mysql. Там вы найдете все приведенные в книге сценарии (из глав, посвященных программированию), текстовые файлы, содержащие длинные SQL-команды, а также перечень ошибок, обнаруженных после выхода книги из печати. Если вы столкнулись с ошибкой при попытке выполнить команду или сценарий и при этом уверены, что набрали текст точно так, как написано в книге, то, прежде чем рвать на себе волосы, проверьте по этому списку, что речь не идет о банальной опечатке! На сайте также приведены полезные ссылки; имеется форум, в котором читатели могут задавать друг другу вопросы (не обязательно связанные с материалом книги) и отвечать на них, и многое другое!

Вопросы, замечания, предложения

Если у вас есть вопросы по поводу MySQL, вы можете обратиться к любому из множества уже существующих сайтов, списков рассылки и перечней часто задаваемых вопросов (frequently asked questions, FAQ). В приложении 3 приведены ссылки на наиболее популярные из подобных ресурсов. Свои вопросы, замечания и предложения вы можете направлять и непосредственно мне по адресу

mysql@DMCinsights.com

Я стараюсь отвечать на все письма, хотя гарантировать этого не могу. В первую очередь и более подробно я отвечу читателям, которые просят пояснить некоторые темы, затронутые в данной книге. Ответы на другие вопросы лучше поискать в списках рассылки или задать их в читательском форуме.

1

УСТАНОВКА MYSQL

Поскольку MySQL поставляется вместе с исходным кодом, то число вариантов ее установки больше, чем у типичного коммерческого приложения. Вы можете просто запустить программу установки, а можете вместо этого задать нужные параметры компиляции и собрать свой экземпляр MySQL из исходных файлов.

В этой главе я рассмотрю инсталляцию на платформах Windows, Macintosh и Linux. Именно с ними будет иметь дело большинство пользователей MySQL, хотя эта СУБД работает и во многих других операционных системах. Прочитав об установке двоичного дистрибутива для Windows и о сборке из исходных файлов для Linux, вы получите представление о различных аспектах проблемы.

Решая вопрос о том, какой именно вариант загрузить из Сети и установить на свой компьютер, следует принимать во внимание различные факторы, из которых самый главный – нужна ли вам стабильная или разрабатываемая версия (часто помеченная как «альфа» или «бета»). MySQL проходит тщательное тестирование как внутри компании MySQL AB, ведущей разработку, так и со стороны пользователей по всему миру, поэтому, устанавливая стабильную версию, вы всегда можете рассчитывать на надежный

и высокопроизводительный продукт. Разрабатываемые версии лучше оставить специалистам, которые могут отличить ошибки в программе от неправильных действий пользователя.

Версия 4 сервера MySQL сейчас является официально выпущенной, а следующая версия (4.1) находится на уровне «альфа». В эту версию включено множество давно ожидаемых функций, в том числе повышение быстродействия и высокоскоростного сервера, прямой доступ к таблицам InnoDB и поддержка безопасных транзакций с помощью протокола SSL (Secure Socket Layer). Вслед за версией 4.0 планируется выпустить версию 4.1, которая будет поддерживать дополнительные типы данных, различные функции, вложенные подзапросы, хранимые процедуры и внешние ключи[1]. После появления этих возможностей MySQL уже можно будет сравнивать с самыми большими и дорогими СУБД. Если вам интересно, какое будущее ожидает MySQL, установите версию 4.1, но лучше не стоит использовать ее для критических приложений, пока не выйдет окончательная редакция.

В этой главе мы рассмотрим процедуру загрузки программного обеспечения, установки сервера и создания системной базы данных. Если на каком-нибудь этапе установки вы столкнетесь с проблемой, обратитесь к приложению 1 или к соответствующим разделам руководства по MySQL.

[1] Более подробно вы познакомитесь с возможностями конкретной версии MySQL и сравните их с возможностями других СУБД, посетив страницу http://www.mysql.com/information/features.html сайта разработчиков. – *Прим. ред.*

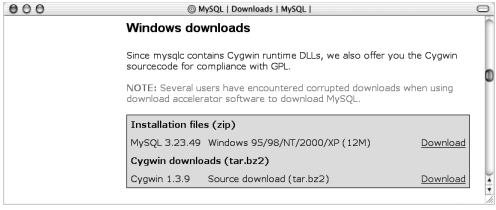

Рис. 1.1. Щелкнув по этой ссылке, вы получите список сайтов-«зеркал», из которых сможете выбрать ближайший к вам

Установка MySQL для Windows

Есть два основных способа установить MySQL в операционной системе Windows: воспользоваться заранее откомпилированным дистрибутивом или собрать собственный экземпляр из исходных файлов. Первый способ гораздо проще – именно его-то мы и рассмотрим.

MySQL работает в большинстве операционных систем семейства Windows (точнее, во всех 32-разрядных системах, таких как Windows 95, 98, Me, NT, 2000 и XP). Если речь идет о системе из семейства NT (Windows NT, 2000 или XP в любом издании – Home, Professional или Server), то MySQL можно запустить как сервис, что и будет продемонстрировано в следующей главе. В нижеприведенном примере я опишу установку последней стабильной версии MySQL на машину под управлением системы Windows XP Home.

Порядок установки MySQL для Windows

1. Загрузите zip-файл с сайта MySQL (рис. 1.1).

   На момент написания этой книги последняя стабильная версия MySQL выпускалась под номером 4.0.

2. Распакуйте загруженный файл.

   Неважно, в какое место на данном этапе распаковывается архив, но установку MySQL (см. п. 3) лучше производить в каталог C:\mysql.

3. Запустите программу setup.exe, дважды щелкнув по ее пиктограмме (рис. 1.2).

   При работе с какой-либо из операционных систем семейства Windows NT вы должны иметь права на установку программ. Выполните предлагаемые действия: выберите целевой каталог (рис. 1.3) и задайте тип установки: **Typical** (Типичная), **Compact** (Компактная) и **Custom** (Выборочная). Лично я предпочел типичную установку в каталог C:\mysql.

   Если в качестве целевого вы указали каталог C:\mysql, все исполняемые программы будут находиться в его подкаталоге bin.

4. Пользуясь программой Проводник (Windows Explorer), откройте только что созданный каталог C:\mysql\bin.

5. Запустите утилиту WinMySQLAdmin (рис. 1.4), дважды щелкнув по соответствующей пиктограмме.

   Эта утилита завершает подготовку файлов, необходимых для нормальной работы MySQL. Кроме того, на данном этапе для систем семейства Windows NT создается сервис MySQL и в системный лоток помещается значок светофора, индицирующий текущее состояние сервиса. Запомните введенные вами имя и пароль пользователя MySQL.

C Необязательно устанавливать MySQL именно в каталог C:\mysql, но я считаю, что это разумно. Если по какой-то причине вы решите произвести установку в другое место, на первом этапе работы программы setup.exe укажите имя начального каталога.

C Как бы то ни было, запускать MySQL в виде сервиса в системах семейства Windows NT проще, если установка произведена в каталог C:\mysql.

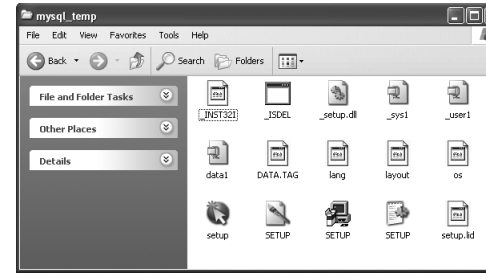

Рис. 1.2. После распаковки дистрибутива запустите программу установки MySQL, дважды щелкнув по пиктограмме приложения setup.exe

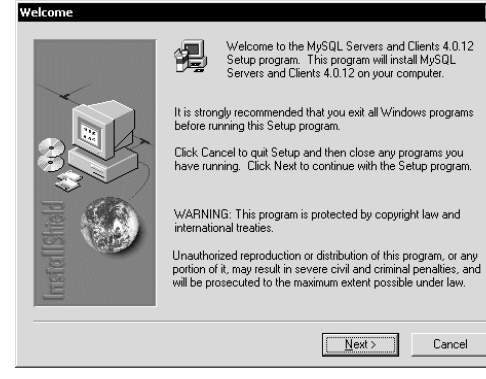

Рис. 1.3. Программа установки setup.exe предлагает выполнить две операции – в частности, указать целевой каталог

Рис. 1.4. Последний этап установки MySQL – запуск утилиты WinMySQLAdmin. Она же используется для администрирования сервера

Рис. 1.5. Сайт Марка Лянежа содержит подробную информацию об использовании MySQL на платформе Mac OS X

Установка MySQL для Mac OS X

В основе системы Mac OS X лежит операционная система FreeBSD, снабженная графическим интерфейсом пользователя Macintosh, а это означает, что к вашим услугам все возможности и надежность операционной системы UNIX, равно как и удобство интерфейса Mac. Поэтому теперь программисты все чаще пользуются операционной системой от компании Apple для разработки проектов. Стоит еще добавить, что значительная часть программ, работающих под UNIX (в том числе MySQL) будет работать и под Mac OS X.

На момент написания этой книги существовало два основных способа установить MySQL на платформу Mac OS X: собрать из исходных файлов (это будет продемонстрировано на примере Linux в следующем разделе) или воспользоваться уже готовым двоичным пакетом, который можно загрузить с сайта разработчиков MySQL. На Web-сайте Марка Лянежа (Marc Liyanage) по адресу www.entropy.ch/software/macosx/mysql подробно освещаются процессы установки MySQL на Mac OS X и обновления уже установленной версии; также приводятся различные примеры, часто задаваемые вопросы (frequently asked questions, FAQ) и даже инструкции по самостоятельной сборке пакета из исходных кодов (рис. 1.5). Я горячо рекомендую вам посетить этот сайт.

Как стать суперпользователем

В операционных системах, производных от UNIX, в частности Mac OS X и Linux, для обеспечения безопасности применяется концепция «пользователя». Суперпользователь с именем root может делать все, что угодно, – даже стереть саму систему. По этой причине вести работу от имени пользователя root нужно очень аккуратно, но при установке новых программ альтернативы просто нет.

Если учетная запись пользователя root уже существует и защищена паролем, то, чтобы зарегистрироваться под этим именем, нужно набрать команду `su root` в приложении Terminal (Терминал). После этого вам будет предложено ввести пароль пользователя root. Альтернативно можно начинать любую команду со слова `sudo`, например, `sudo ./scripts/mysql_install_db`, но тогда каждый раз придется вводить пароль.

Чтобы задать пароль пользователя root в системе Mac OS X, наберите команду `sudo passwd root`, в ответ на первое приглашение нажмите клавишу **Return**, а затем два раза подряд введите придуманный вами пароль.

Рис. 1.6. В системе Mac OS X, как и в UNIX, дистрибутивы программ могут распространяться в виде пакетов (файлов с расширением .pkg). Вот как выглядят их пиктограммы

Рис. 1.7. Панель **Users**, вызываемая из меню **System Preferences**, служит в системах Apple для управления учетными записями пользователей

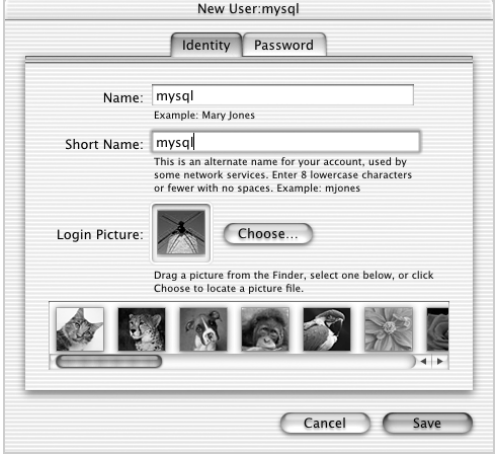

Рис. 1.8. Конкретные свойства пользователя mysql не столь существенны. Важно лишь, чтобы это имя было уникальным и легко запоминалось. Если вы ассоциируете с ним пароль, его тоже нужно запомнить

Порядок установки MySQL для Mac OS X

1. Загрузите пакет с сайта разработчиков MySQL: http://www.mysql.com/downloads/mysql-4.0.html.

   На момент написания этой книги по вышеприведенному адресу находилась версия 4.0.

2. Распакуйте загруженный файл.

   С помощью утилиты StuffIt Expander или аналогичной программы раскройте загруженный архив; в результате вы увидите пакет, содержащий MySQL (рис. 1.6).

3. Установите MySQL.

   Дважды щелкните мышью по пакету MySQL и следуйте инструкциям мастера.

4. Откройте панель управления **Users** (Пользователи) с помощью меню **System Preferences** (Системные настройки) – рис. 1.7.

   Я собираюсь создать учетную запись специального пользователя, от имени которого будет запускаться MySQL. Это мера безопасности, которую рекомендую принять и вам.

5. Добавьте нового пользователя с именем mysql (рис. 1.8). Для этого щелкните по кнопке **New User** (Новый пользователь), затем введите в поля формы полное и краткое имя `mysql` и пароль по своему выбору. Нажмите кнопку **Save** (Сохранить) и закройте панель управления **Users**.

6. Откройте приложение Terminal (Терминал) и зарегистрируйтесь как пользователь root (рис. 1.9).

   Выполнять нижеописанные действия, как правило, проще, если вы входите в систему как пользователь с именем root. Если для такого пользователя еще не создано учетной записи, можете начинать каждую команду словом `sudo`, но тогда придется всякий раз вводить пароль администратора (см. выше врезку «Как стать суперпользователем»).

7. Перейдите в каталог /usr/local/mysql:

   ```
   cd /usr/local/mysql
   ```

8. Создайте начальные базы данных (рис. 1.10):

   ```
   ./scripts/mysql_install_db
   ```

9. Измените права доступа к каталогу:

   ```
   chown -R mysql /usr/local/mysql/*
   ```

Если вы создали специального пользователя, от имени которого будет работать MySQL, то нужно изменить права доступа к файлам, входящим в состав MySQL, сделав нового пользователя их владельцем. После этого вы будете готовы к запуску системы MySQL и к работе с ней.

С Если вы решите собрать MySQL из исходных файлов (а это совсем не сложно), то придется предварительно установить средства разработки компании Apple. Их можно загрузить с сайта Apple (http://developer.apple.com) или взять с компакт-диска Developer Tools, поставляемого вместе с операционной системой.

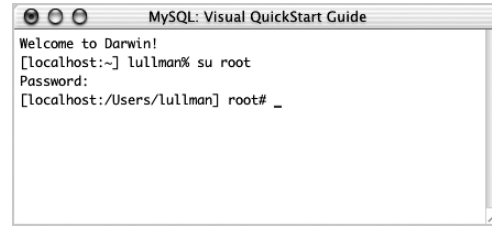

Рис. 1.9. Для завершения процедуры установки воспользуйтесь приложением Terminal и зарегистрируйтесь как пользователь root

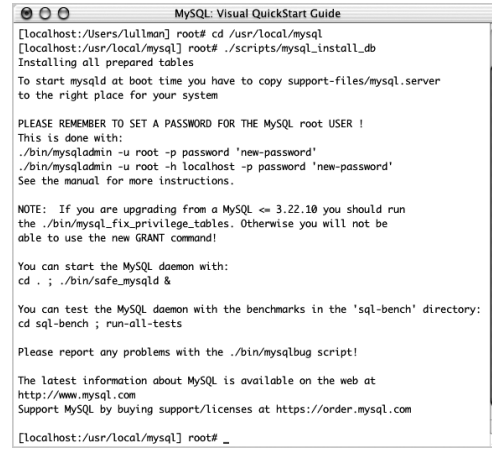

Рис. 1.10. Сценарий mysql_install_db создает две начальные базы данных – mysql и test

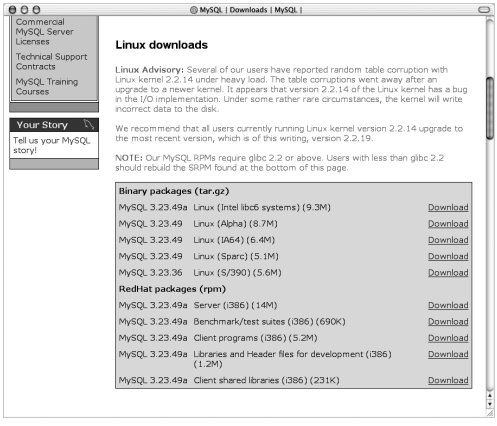

Рис. 1.11. Простейший способ установки MySQL в системе Linux – это загрузка с сайта MySQL подходящего дистрибутива и его инсталляция как любого другого RPM-пакета

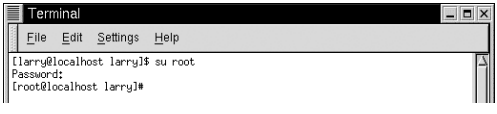

Рис. 1.12. При установке двоичного или исходного дистрибутива MySQL необходимо работать с командным интерпретатором (предпочтительно от имени пользователя root)

Установка MySQL для Linux

Компания MySQL AB считает, что для установки MySQL в системе Linux лучше всего пользоваться RPM-пакетом (RPM – Red Hat Package Manager, администратор пакетов, применяемый в дистрибутиве Red Hat Linux). На сайте MySQL имеется несколько пакетов для различных целей, скажем для установки только серверной или только клиентской части (рис. 1.11). Скорее всего, вы захотите установить и сервер, и клиент, но если нужен только доступ к серверу MySQL, работающему на другой машине, то будет достаточно одной лишь клиентской части.

По умолчанию RPM-пакет подготовлен так, что приложение устанавливается в каталог /var/lib/mysql. Кроме того, в каталог /etc/rc.d записываются служебные файлы, гарантирующие автоматический запуск сервера в виде процесса-демона при перезагрузке компьютера. Процедура установки RPM-пакета не вызывает сложностей, так что я не стану ее здесь рассматривать. Она аналогична установке любого другого RPM-пакета или запуску программы setup.exe в среде Windows.

Еще два возможных варианта – это установка заранее собранной двоичной версии и компиляция исходных файлов вручную. Необходимые при этом действия частично совпадают, поэтому я опишу обе процедуры одновременно.

Порядок установки MySQL для Linux

1. Получите доступ к серверу из командной строки (рис. 1.12).

   В этом примере описывается установка MySQL в системе Red Hat Linux 6.1 с помощью приложения Terminal. Вам будут необходимы права для манипулирования файлами и создания новых

каталогов внутри каталога /usr/local, поэтому имеет смысл войти в систему от имени пользователя root (см. выше врезку «Как стать суперпользователем»).

2. Перейдите в каталог /usr/local:

```
cd /usr/local
```

Я предпочитаю устанавливать MySQL в каталог /usr/local, как рекомендуется в руководстве. Разумеется, вы можете выбрать любое другое место, допускаемое операционной системой.

3. Загрузите двоичный или исходный дистрибутив MySQL.

Здесь есть несколько вариантов, вот самые простые:

- загрузить файл с сайта www.mysql.com, пользуясь браузером, а затем переместить его в каталог /usr/local;
- воспользоваться утилитами wget или curl, чтобы сразу загрузить файл в текущий каталог.

4. Распакуйте файлы (рис. 1.13):

```
gunzip mysql-4.0.12.tar.gz
tar mysql-4.0.12.tar
```

Файл с исходным дистрибутивом именуется в соответствии с соглашением mysql-VERSION.tar.gz. Файл с двоичным дистрибутивом именуется в соответствии с соглашением mysql-VERSION-PLATFORM.tar.gz, например: mysql-4.23.49a-pc-linux-gnu-i686.tar.gz. Выполняя эту и последующие процедуры, не забывайте изменять имена файлов в командах в соответствии с той версией, которую вы устанавливаете.

5. Создайте символическую ссылку на начальный каталог MySQL (рис. 1.14):

```
ln -s /usr/local/mysql-4.0.12 mysql
```

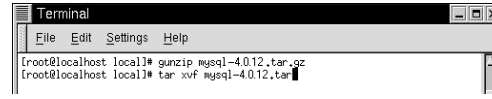

Рис. 1.13. Первый этап установки двоичного или исходного дистрибутива – распаковка загруженного файла в каталог /usr/local

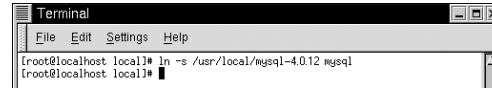

Рис. 1.14. Создание символической ссылки на начальный каталог MySQL поможет сэкономить время и избежать ошибок, хотя эта операция не обязательна

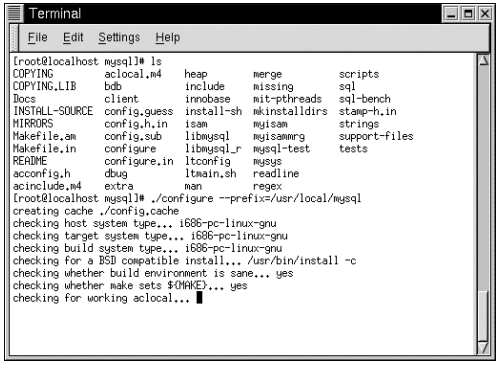

Рис. 1.15. При сборке MySQL из исходных файлов вы должны самостоятельно выполнить процедуру конфигурирования. Как минимум следует указать в команде `configure` флаг `prefix`

Это необязательная процедура, но она позволит ссылаться на файлы, входящие в состав дистрибутива MySQL, с использованием более коротких имен, например /usr/local/mysql вместо /usr/local/mysql-4.0.12.

6. Создайте пользователя и группу для работы с MySQL:

```
groupadd mysql
useradd -g mysql mysql
```

Это позволит запускать MySQL и администрировать эту СУБД от имени пользователя mysql, а не root, что повышает безопасность системы в целом.

7. Перейдите в каталог mysql:

```
cd mysql
```

Поскольку на этапе 5 была создана символическая ссылка, данная команда, по сути, делает текущим тот каталог, в который была установлена система MySQL (например, /usr/local/mysql-4.0.12/).

8. Сконфигурируйте исходные файлы, входящие в дистрибутив (рис. 1.15):

```
./configure --prefix=/usr/local/
⇨ mysql
```

Если вы производите сборку из исходного дистрибутива, необходимо самостоятельно выполнить конфигурирование, а затем команды сборки и установки (см. пп. 9 и 10). Если же вы устанавливаете двоичный дистрибутив, пропустите этот и следующие два шага.

Подробнее о процедуре конфигурирования MySQL рассказывается ниже, в разделе «Параметры конфигурации», а также в руководстве по MySQL. Флаг `prefix` очень важен: он задает каталог, в который будет установлена система MySQL после сборки.

9. По завершении конфигурирования выполните команды сборки и установки продукта (рис. 1.16).

```
make
make install
```

На выполнение этих процедур уйдет значительное время, зависящее от вашей операционной системы и быстродействия процессора. Если на этапе сборки (при выполнении команды `make`) произойдет ошибка, не запускайте команду `make install`. В случае, когда проблему не удается решить сразу, обращайтесь к руководству по MySQL или к приложению 1 либо попробуйте установить двоичный, а не исходный дистрибутив.

10. Создайте начальные базы данных (рис. 1.17):

```
scripts/mysql_install_db
```

На этой стадии создаются база данных, необходимая для работы MySQL (она называется mysql), и тестовая база данных под названием test. Их необходимо организовать при установке как двоичного, так и исходного дистрибутива. По завершении работы вы увидите сообщения, касающиеся установленного продукта (рис. 1.18).

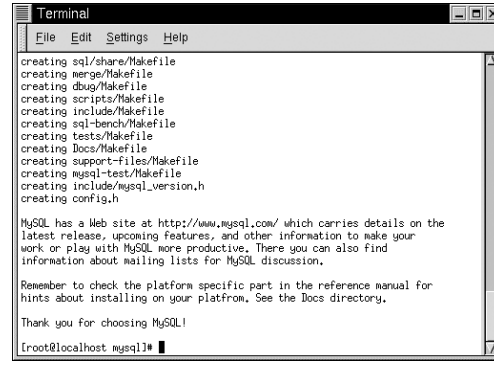

Рис. 1.16. Если процедура конфигурирования (рис. 1.15) завершилась успешно, на экране должно появиться сообщение о том, что можно приступать к сборке и установке

Рис. 1.17. Перед первым запуском сервера необходимо создать начальные базы данных – для этого предназначен сценарий mysql_install_db

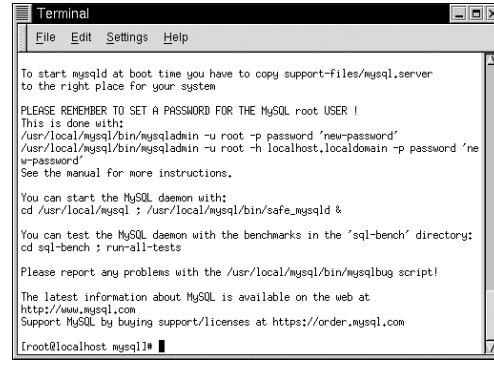

Рис. 1.18. После создания начальных баз данных вы увидите сообщения о том, что надо сделать, чтобы запустить сервер

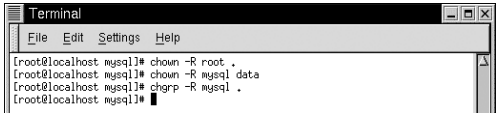

Рис. 1.19. Предоставив пользователю mysql права доступа к своим файлам, вы предотвратите запуск MySQL от имени root

11. Измените права доступа к вновь созданным файлам (рис. 1.19):

```
chown -R root /usr/local/mysql/
chown -R mysql /usr/local/mysql/data
chgrp -R mysql /usr/local/mysql/
```

Это завершающая операция. Она позволяет запускать сервер MySQL от имени вновь созданного пользователя mysql.

С В зависимости от операционной системы для добавления пользователя и группы mysql, возможно, придется использовать другие команды (например, `adduser` и `addgroup`). Если вы работаете с системой Red Hat Linux, то проще всего воспользоваться графической панелью конфигурации Linuxconf.

Параметры конфигурации

Поскольку система MySQL поставляется в исходных кодах и обладает большой гибкостью, то сконфигурировать ее можно самыми разными способами. При сборке MySQL из исходных файлов стоит задать определенные параметры, от которых зависит работа программы.

Во время установки можно с помощью флага `prefix` указать, где должны находиться двоичные файлы. Также вы вправе решить, следует ли устанавливать сервер MySQL (или оставить только клиента), и задать используемый язык. Это всего лишь некоторые примеры – полный перечень настроек вы получите, если наберете команду `./configure --help`, предварительно открыв начальный каталог MySQL (например, /usr/local/mysql) – рис. 1.20.

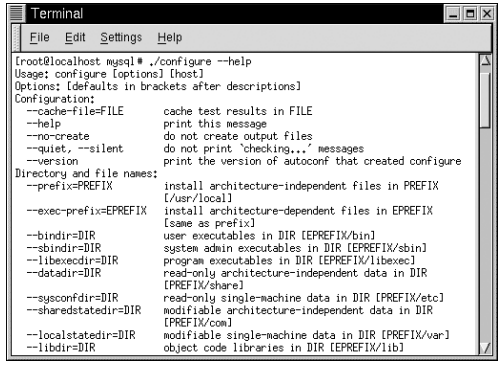

Рис. 1.20. В руководстве по MySQL описаны наиболее распространенные параметры конфигурации, а команда `./configure --help` выдает полный их перечень

По завершении установки MySQL допускается дополнительное конфигурирование путем редактирования файла my.cnf, расположенного в каталоге /etc (впрочем, это зависит от конкретной установки). Чтобы выяснить, какие возможности у вас имеются, откройте my.cnf в текстовом редакторе. Если такого файла нет, создайте его самостоятельно. Не забудьте перезапустить сервер MySQL после внесения изменений, чтобы они вступили в силу (порядок останова и запуска сервера рассматривается в главе 2).

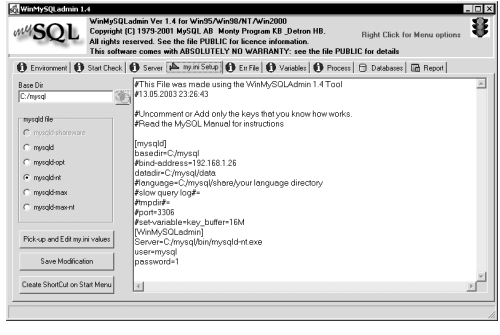

Рис. 1.21. На платформе Windows для изменения некоторых аспектов работы MySQL лучше всего воспользоваться утилитой WinMySQLAdmin

На платформе Windows изменить параметры работы MySQL проще всего с помощью утилиты WinMySQLAdmin (рис. 1.21), которую мы уже упоминали при описании процедуры установки. Результат работы WinMySQLAdmin – внесение изменений в файл my.ini или my.cnf, который сервер считывает во время запуска. Первый файл находится в каталоге Windows (обычно C:\WINDOWS или C:\WINNT), а второй – в начальном каталоге MySQL (обычно C:\mysql). Их оба можно модифицировать и вручную с помощью любого текстового редактора, например Notepad.

Подробную информацию о конфигурировании MySQL на различных платформах вы найдете на странице www.mysql.com/doc/c/o/configure_options.html.

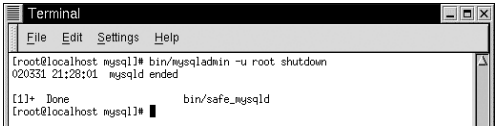

Рис. 1.22. Для останова работающего сервера MySQL можно прибегнуть к помощи утилиты mysqladmin

Обновление версии MySQL

Итак, вы успешно установили и запустили MySQL. Но позже может понадобиться обновить версию продукта, особенно когда будет выпущена версия 4.1 или 5.0. Процедура обновления MySQL не сложнее процедуры первоначальной установки, но все же лучше предпринять некоторые меры предосторожности (например, сделать резервную копию системы) на случай, если что-то пойдет не так. При установке MySQL на платформе Windows инсталлятор может уничтожить существующие файлы с данными и вообще все файлы MySQL, если вы будете работать неаккуратно.

В руководстве по MySQL подробно рассматриваются особенности обновления версии этой системы – ознакомьтесь с ними, чтобы получить представление о возможных ситуациях. Нижеприведенные рекомендации носят более общий характер.

Порядок обновления MySQL

1. Остановите работающий процесс сервера MySQL (рис. 1.22).

   В принципе остановить сервер можно, набрав команду `bin/mysqladmin -u root shutdown` при открытом каталоге mysql.

   Если вы задали пароль пользователя MySQL с именем root (а это необходимо сделать), то команда должна выглядеть так:

   ```
   bin/mysqladmin -u root -p shutdown
   ```

   При работе в системе Windows из окна сеанса DOS наберите команду `NET STOP MySQL`, если СУБД MySQL запущена как сервис (или воспользуйтесь пиктограммой-светофором в системном лотке, как показано в следующей главе).

   Подробнее об останове сервера MySQL рассказывается в главе 2.

2. Сделайте резервную копию данных MySQL (рис. 1.23).

   В главе 10 описаны утилиты, специально предназначенные для резервного копирования базы данных. Но в системах Linux и Mac OS X можно просто выполнить команду

   ```
   tar -cvf /tmp/mysql-data.tar data
   ```

   которая упакует весь каталог data в файл, сохраненный во временном каталоге tmp.

3. Установите новую версию MySQL.

   Установка производится в соответствии с вышеописанными инструкциями. Некоторые этапы, например создание нового пользователя и запуск сценария mysql_install_db, можно пропустить.

4. Запишите во вновь созданный каталог data данные из резервной копии (рис. 1.24).

   Восстановление данных из резервной копии подробнее описывается в главе 10, но если вы последовали рекомендации в п. 2 (в системе Linux), то достаточно будет набрать команду

   ```
   tar -xf /tmp/mysql-data.tar
   ```

5. Перезапустите сервер, следуя инструкциям из главы 2.

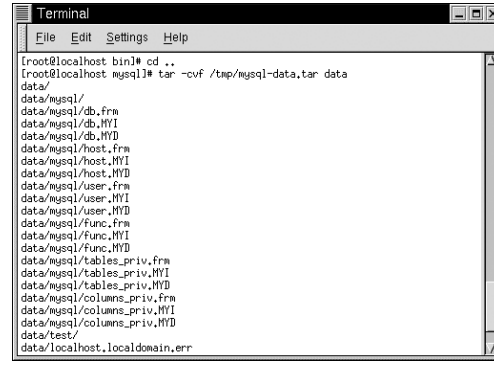

Рис. 1.23. Один из способов архивировать данные MySQL – воспользоваться командой `tar`. В главе 2 рассматриваются другие утилиты, применяемые для решения этой задачи

Рис. 1.24. Для восстановления данных из резервной копии (рис. 1.23) снова применяется команда `tar`

С Если вы работаете в системе Windows NT и СУБД MySQL запущена как сервис, то вам придется удалить существующий сервис. Для этого выполните в сеансе DOS команду `mysqld-max-nt --remove`, предварительно открыв каталог mysql\bin.

П Сценарий mysql_install_db нужно запускать только один раз, но случайный повторный запуск не катастрофичен, поскольку существующие базы данных и таблицы не будут затерты.

С В качестве дополнительной меры предосторожности при обновлении MySQL на платформе UNIX можно заранее переименовать исполняемый файл mysqld в mysqld.old или нечто подобное. Тогда, если новый сервер не заработает, вы без труда вернетесь к прежней версии.

С При обновлении версии MySQL на платформе UNIX не забудьте, что при установке RPM-пакета в каталоге /etc/rc.d создаются файлы, необходимые для автоматического запуска сервера MySQL во время загрузки. Отсюда следует, что при обновлении исходный стартовый файл в каталоге /etc/rc.d будет перезаписан, если вы заранее не сделаете копию.

Ставим «заплаты»

Для исправления ошибок или устранения брешей в системе безопасности иногда приходится ставить «заплаты» – так называемые *патчи* (patches). Компания MySQL AB поддерживает список текущих патчей на своем сайте по адресу www.mysql.com/downloads/patches.html (рис. 1.25). Имейте в виду, что «наложить заплату» можно только на исходный дистрибутив MySQL, который вы собирали самостоятельно, а не на двоичный дистрибутив или RPM-пакет.

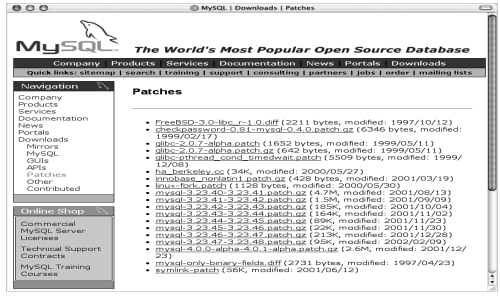

Рис. 1.25. Компания MySQL AB поддерживает список текущих патчей на своем сайте. Чтобы не отстать от жизни, периодически проверяйте, не нужно ли выполнить обновление

Порядок установки патча

1. Остановите работающий сервер MySQL.
2. Загрузите файл с патчем в тот же каталог, где находится исходный дистрибутив.
3. При необходимости распакуйте файл:

   ```
   gunzip имя_файла_патча.gz
   ```

   Патчи – это просто текстовые файлы. Другой способ загрузить патч – просмотреть его в окне браузера, а затем сохранить на диске в виде текстового файла.
4. Установите патч, набрав команду

   ```
   patch -p1 <имя_файла_патча
   ```

   Она внесет необходимые изменения в исходные файлы.
5. Удалите ранее созданную информацию о параметрах конфигурации:

   ```
   rm config.cache
   make clean
   ```

   При конфигурировании программ в системе UNIX создается файл config.cache, где сохраняется информация о параметрах конфигурации. Перед сборкой нового экземпляра продукта с иными параметрами этот файл следует удалить или переименовать.
6. Повторно выполните процедуры конфигурирования, сборки и установки, пользуясь вышеописанными инструкциями из этой главы.

2

Эксплуатация MySQL

После того как система MySQL успешно установлена, пришло время узнать, как запускать и останавливать сервер. Если исходить из того, что вы установили не только клиентскую часть, в вашем распоряжении имеется несколько утилит, предназначенных для запуска и администрирования сервера.

В этой главе я сначала расскажу о том, как запускать и останавливать MySQL на различных платформах. Затем будет описано, как создается пароль пользователя root, как работать с клиентской программой mysql и добавлять новых пользователей. Если, выполняя упражнения, вы столкнетесь с ошибками, обратитесь к приложению 1 или к соответствующим разделам руководства по MySQL.

Я продемонстрирую различные приемы на примере операционных систем Red Hat Linux, Windows 2000 и Mac OS X. На наше счастье, большинство команд одинаково во всех операционных системах, будь то Linux, Mac OS X, AIX или сеанс DOS в Windows XP. Не опасайтесь, что описываемые приемы у вас не сработают, – я постарался подробно рассказать, какие изменения могут понадобиться при решении задачи в тех или иных условиях. Что касается пользователей Windows, почти все задачи администрирования базы данных можно решить с помощью утилиты WinMySQLAdmin, кратко описанной в предыдущей главе.

Запуск MySQL

Увы, вы не узнаете, нормально ли установилась система MySQL, пока не попробуете запустить сервер. При запуске часто возникают проблемы и всякого рода путаница, особенно если вы еще не очень опытный пользователь системы UNIX. С другой стороны, раз уж СУБД MySQL запущена, она будет устойчиво и надежно работать в течение многих месяцев без перезагрузки. Если что-то на данном этапе пойдет не так, обратитесь к разделу «Запуск MySQL» в приложении 1.

Собственно сервер MySQL, то есть приложение, управляющее доступом к базе данных, находится в исполняемом файле mysqld. На платформах, отличных от Windows, процесс-демон mysqld запускается с помощью программы safe_mysqld, которая следит за тем, чтобы сервер не прекращал работу. В системах семейства Windows NT можно запустить один из нескольких исполняемых файлов, входящих в дистрибутив. Под NT программа MySQL может работать как сервис (то есть будет автоматически запускаться в процессе загрузки операционной системы). В других ОС, скажем, Windows 98 или Windows Me, программу mysqld придется запускать вручную после каждой перезагрузки компьютера.

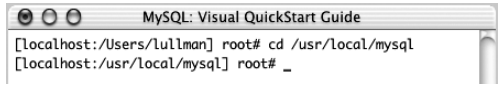

Рис. 2.1. Практически все описанные в этой главе действия должны выполняться из командной строки при условии, что открыт каталог mysql

Порядок запуска MySQL в системах UNIX

1. Войдите в систему, пользуясь командным интерпретатором.
2. Перейдите в начальный каталог MySQL (рис. 2.1):

   ```
   cd /usr/local/mysql
   ```

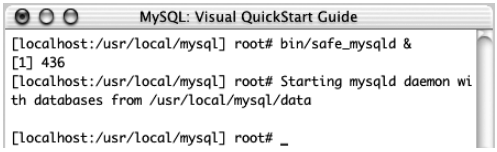

Рис. 2.2. Если процесс-демон mysqld запустился нормально, то вы увидите такие сообщения

Если при установке вы следовали инструкциям из главы 1, то все необходимые утилиты должны находиться в подкаталоге bin каталога /usr/local/mysql. В системе Red Hat Linux они будут расположены в каталоге /usr/bin[1].

3. Запустите программу safe_mysqld в фоновом режиме (рис. 2.2):

   ```
   bin/safe_mysqld &
   ```

   Эта команда гарантирует непрерывную работу сервера MySQL. Если процесс mysqld по каким-то причинам завершается, команда перезапускает его. При первом запуске safe_mysqld обнаруживает, что mysqld не работает, и сразу же вводит его в действие.

   В системах UNIX иногда трудно разобраться с тем, из какого места следует запускать приложения. Если вышеприведенная команда не дает желаемого результата, попробуйте ввести `./bin/safe_mysqld &` или перейдите в каталог bin и наберите `safe_mysqld &` либо `./safe_mysqld &`.

   Если mysqld не запускается, запомните напечатанное сообщение об ошибке и обратитесь к приложению 1, где рассмотрены типичные проблемы.

4. Приступайте к администрированию базы данных.

Далее в этой главе будет рассказано о том, как обращаться к базе данных MySQL, создавать новых пользователей и выполнять многие другие задачи.

[1] Если установка производилась из RPM-пакета. – *Прим. переводчика.*

Запуск MySQL на платформе Windows гораздо проще, чем в UNIX. Нужно только определиться с тем, какую версию mysqld запускать, – а тут есть несколько вариантов, причем в системах семейства Windows NT (Windows NT, 2000 и XP) их больше, так как существует возможность запуска MySQL в виде сервиса, которую вы, вероятно, и захотите использовать. Ниже описывается, как запустить MySQL под Windows, если вы решили обойтись без рассмотренной в главе 1 утилиты WinMySQLAdmin (она автоматически запускает MySQL как сервис).

Порядок запуска MySQL в системах Windows

1. Решите, какой тип сервера использовать (рис. 2.3).

   В дистрибутив MySQL включено несколько версий сервера MySQL. Выбор зависит от того, какие цели вы ставите. Имеются следующие варианты:

   – mysqld – стандартный сервер;
   – mysqld-opt – версия, оптимизированная для достижения максимальной производительности;
   – mysqld-nt – аналогична mysqld, но предназначена специально для ОС семейства Windows NT;
   – mysqld-max – аналогична mysqld-opt, но включает поддержку таблиц типа InnoDB и DBD. Эти типы таблиц будут рассмотрены в главе 11.
   – mysqld-nt-max – функциональное объединение mysqld-nt и mysqld-max.

   Средний пользователь, скорее всего, не заметит разницы между этими вариантами. Я рекомендую начать с самого простого – mysqld или mysqld-nt, а позже при необходимости перейти к другой версии для оптимизации производительности. Выбор сервера не отражается на данных, хранящихся в базе.

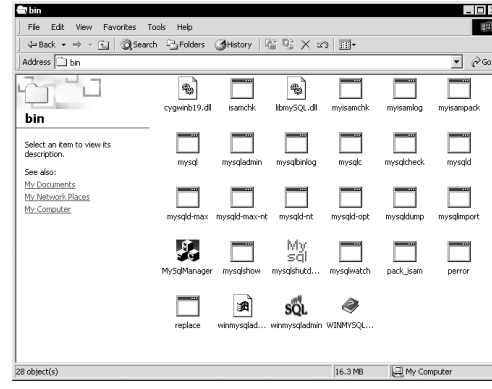

Рис. 2.3. В дистрибутив MySQL для Windows входит несколько заранее сконфигурированных исполняемых файлов сервера; все они находятся в каталоге mysql/bin

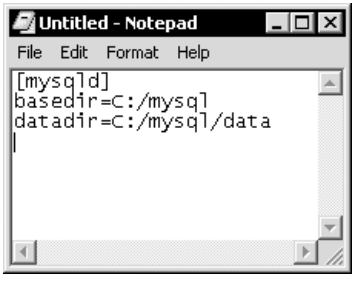

Рис. 2.4. Файл my.ini можно создать вручную, пользуясь программой Notepad или любым другим текстовым редактором

2. Запустите Notepad или любой другой текстовый редактор для создания конфигурационного файла.

   Конфигурационный файл требуется для работы MySQL в нескольких случаях, и, коль скоро вы не пользовались утилитой WinMySQLAdmin, его придется создать вручную. Это необходимо, если вы хотите изменить местоположение файлов MySQL, собираетесь пользоваться вариантами сервера mysqld-opt, mysqld-max или mysqld-max-nt (см. п. 1) или как-то иначе изменить параметры работы сервера MySQL. На платформе Windows можно создать либо файл my.ini в каталоге WINDOWS или WINNT, либо файл my.cnf в каталоге C:\. Вы вправе использовать либо тот, либо другой, но не оба сразу; я рекомендую остановиться на my.ini.

   Хочу все же отметить, что если вы следовали инструкциям в главе 1 и запустили утилиту WinMySQLAdmin, то она создала конфигурационный файл автоматически.

3. Наберите в текстовом редакторе следующие строки (рис. 2.4):

   ```
   [mysqld]
   basedir=C:/mysql
   datadir=C:/mysql/data
   ```

   Это лишь малая часть того, что может храниться в конфигурационном файле. Приведенные выше строки сообщают серверу MySQL, где находятся файлы MySQL (`basedir`) и файлы данных (`datadir`). Если вы изменили параметры во время установки, скорректируйте команды.

В файле могут присутствовать комментарии, например содержащие дату его создания и имя автора. Строка комментария должна начинаться со знака #.

4. Сохраните файл, назвав его my.ini.

   Файл следует сохранять в системном каталоге Windows (обычно C:\WINDOWS или C:\WINNT).

5. Запустите сервер (рис. 2.5).

   Находясь в сеансе DOS, наберите следующие строки:

   ```
   cd C:\mysql\bin
   mysqld --standalone
   ```

   Если вы работаете не в системе семейства Windows NT, то флаг `--standalone` можно не ставить.

6. Приступайте к администрированию базы данных.

Далее в этой главе будет рассказано о том, как обращаться к базе данных MySQL, создавать новых пользователей и выполнять другие важные операции.

Рис. 2.5. Чтобы запустить сервер mysqld на платформе Windows, не прибегая к утилите WinMySQLAdmin, перейдите в каталог mysql/bin и наберите команду `mysqld` (следует добавить флаг `--standalone`, если работа ведется в системе семейства Windows NT)

П Приложения mysqld-nt и mysqld-max-nt, предназначенные для работы в системах Windows NT, 2000 или XP, могут работать и в системах Windows 95, 98, Me при условии, что установлен стек TCP/IP.

С Чтобы открыть сеанс DOS, выберите пункт **Run** (Выполнить) из меню **Start** (Пуск) и введите имя команды `cmd`.

С Некоторые пользователи системы Red Hat Linux сталкивались с трудностями, пытаясь запустить MySQL согласно вышеизложенным инструкциям. Если СУБД MySQL устанавливалась с помощью RPM-пакета, то она, вероятно, уже сконфигурирована как сервис. В таком случае запустить mysqld можно, либо введя команду `service mysqld start`, либо воспользовавшись панелью управления Linuxconf. Ну и на крайний случай подойдет команда `/etc/rc.d/init.d/mysqld start`.

П Следует помнить, что в Windows разделителем каталогов в пути к файлу служит обратная косая черта (\), тогда как в других операционных системах – прямая (/).

С Программе mysqld (а, стало быть, и сценарию safe_mysqld) можно передать в командной строке аргументы, модифицирующие способ ее работы. Обычному пользователю это, как правило, не требуется, но изучить дополнительную информацию не помешает. Для этого обратитесь к странице www.mysql.com/doc/C/o/Command-line_options.html.

Останов MySQL

Уж если вам удалось запустить сервер MySQL, то остановить его совсем просто. Впрочем, вам вряд ли придется часто останавливать сервер – разве что перед обновлением версии или на период длительного обслуживания (все приложение в целом спроектировано так, чтобы работать непрерывно).

Порядок останова сервера MySQL одинаков для Windows и UNIX/Linux/Mac OS X, если не считать возможности использования утилиты WinMySQLAdmin. Сначала я приведу общие рекомендации, затем остановлюсь на WinMySQLAdmin, а в примечаниях познакомлю вас еще с двумя методами.

Порядок останова MySQL

1. Войдите в систему, пользуясь командным интерпретатором.

   Для выполнения этой операции необязательно иметь полномочия администратора (входить под именем root).

2. Перейдите в каталог `mysql/bin`, воспользовавшись одной из команд:

   `cd /usr/local/mysql/bin` (UNIX)

   `cd C:\mysql\bin` (Windows)

Снова повторим, что в разных операционных системах действуют свои правила, определяющие, откуда можно запускать программы. Измените вышеуказанные строки, если возникнут проблемы. Например, в некоторых версиях Linux следует перейти в каталог /usr/local/mysql и добавить к имени следующей ниже команды `bin/`. В других версиях Linux, например Red Hat, нужно перейти в каталог /usr/bin.

3. Наберите команду `mysqladmin shutdown` (рис. 2.6).

   Если пользователю root уже присвоен пароль, в дополнение необходимо указать еще два флага:

   ```
   mysqladmin -u root -p shutdown
   ```

После нажатия клавиши **Return** в ответ на приглашение введите пароль пользователя root (рис. 2.7).

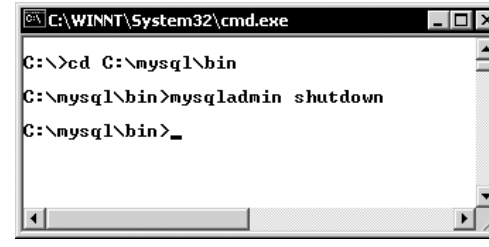

Рис. 2.6. Если вы еще не задали пароль для пользователя root, то можете остановить сервер MySQL командой `mysqladmin shutdown`

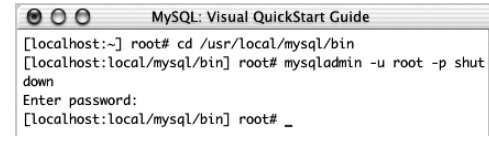

Рис. 2.7. Если для пользователя root требуется пароль, добавьте к команде `shutdown` флаги `-u root -p`

Порядок останова MySQL с помощью утилиты WinMySQLAdmin

1. Щелкните по пиктограмме-светофору в системном лотке (рис. 2.8).

   Утилита WinMySQLAmin запускает MySQL как сервис и одновременно помещает в системный лоток (расположенный по умолчанию в правом нижнем углу экрана) пиктограмму-светофор. При щелчке по ней открывается контекстное меню.

2. Выберите пункт **Win NT** (рис. 2.9).

   Если вы работаете с системой не из семейства Windows NT, соответствующий пункт, конечно, будет называться иначе.

3. Щелкните по пункту **Stop the Service** (Остановить сервис) – рис. 2.9.

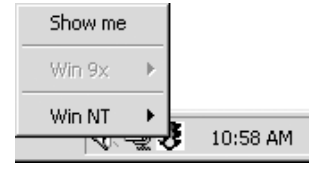

Рис. 2.8. Щелчок по пиктограмме-светофору предоставляет доступ к утилите WinMySQLAmin

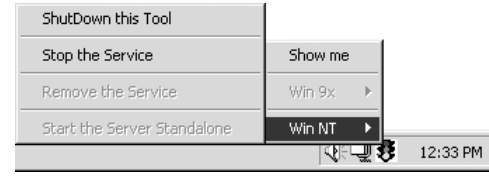

Рис. 2.9. При наведении указателя мыши на пункт **Win NT** открывается второе меню, в числе прочего содержащее команду останова сервера MySQL

Рис. 2.10. Окно подсказки дает вам шанс передумать и отменить останов сервера

Рис. 2.11. Красный цвет «светофора» свидетельствует о том, что MySQL не работает. Зеленый означает, что сервис работает, а желтый – что выполняется процедура останова

4. Нажмите кнопку **Yes** (Да) во всплывающем окне (рис. 2.10).

   В отличие от UNIX, версия MySQL для Windows сначала предупредит, что останов сервиса приведет к закрытию всех установленных соединений (то есть работающие пользователи будут отключены).

5. Убедитесь, что сервис MySQL остановлен, посмотрев на пиктограмму-светофор (рис. 2.11).

   Если сервис MySQL работает, то горит зеленый свет. Пока идет процедура останова, светофор горит желтым. А когда сервис остановлен, включается красная лампочка.

П В системах UNIX (в частности, Linux) сервер также останавливается по команде `/etc/rc.d/init.d/mysqld stop`.

С Если MySQL работает как сервис под Windows NT, то остановить программу можно командой `NET STOP mysql` (независимо от того, какой каталог при этом открыт).

С В большинстве систем Windows можно пропустить один шаг при выполнении вышеописанных действий, введя команду `C:\mysql\bin\mysqladmin shutdown` без предварительного перехода в начальный каталог MySQL.

Применение утилиты mysqladmin

Как следует из ее названия, утилита mysqladmin предназначена для администрирования базы данных. В круг задач, решаемых при помощи этой программы, входит останов сервера MySQL (см. предыдущий раздел), установка паролей пользователей и др. Ряд функций mysqladmin продублирован в утилите WinMySQLAdmin, разработанной для платформы Windows. Наконец, многое из того, что делает mysqladmin, можно выполнить непосредственно из клиентской программы mysql, о которой пойдет речь в следующем разделе данной главы.

Одно из самых первых применений mysqladmin – назначение пароля пользователя root. При установке MySQL этот пароль не задается. Разумеется, здесь налицо брешь в системе безопасности, которую необходимо заделать до запуска сервера. Напомним, что к базе данных, равно как и к операционной системе, может обращаться несколько пользователей. Однако пользователи MySQL и пользователи ОС – это не одно и то же, даже если их имена совпадают. Так, пользователь MySQL root не имеет ничего общего с суперпользователем ОС под именем root: у них разные полномочия и даже пароли могут быть разные (последнее желательно, хотя и необязательно).

Нужно понимать, что для работы утилиты mysqladmin сервер MySQL (mysqld) должен быть запущен. Если он остановлен, запустите его, как было показано выше.

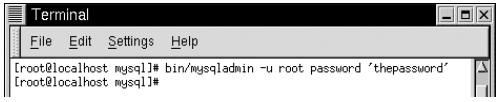

Рис. 2.12. Поскольку я не могу вызвать утилиту mysqladmin непосредственно из каталога bin, то для изменения пароля пришлось вызвать ее как `bin/mysqladmin`, открыв каталог mysql

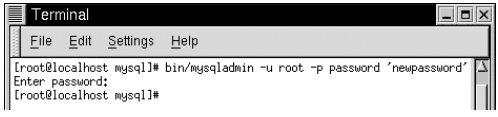

Рис. 2.13. Для изменения существующего пароля следует добавить флаг `-p` при вызове утилиты mysqladmin (рис. 2.12), чтобы можно было ввести текущий пароль

Порядок установки пароля пользователя root

1. Войдите в систему из командного интерпретатора.
2. Перейдите в каталог mysql/bin или просто в каталог mysql в зависимости от операционной системы:

   `cd /usr/local/mysql/bin` (UNIX)

   `cd C:\mysql\bin` (Windows)

3. Введите следующую команду, заменив слово *пароль* желаемым паролем (см. рис. 2.12):

   ```
   mysqladmin -u root password
   ⇨ 'пароль'
   ```

   Помните, что MySQL различает регистр символов в паролях, так что `Kazan` и `kazan` – это не одно и то же. Слово `password`, предшествующее самому паролю, заключенному в одинарные кавычки, означает, что система MySQL должна зашифровать следующую строку.

Изменить первоначально заданный пароль немногим сложнее.

Порядок изменения пароля пользователя root

1. Войдите в систему из командного интерпретатора.
2. Перейдите в каталог mysql/bin или просто в каталог mysql в зависимости от операционной системы.
3. Введите следующую команду, заменив слово *новый пароль* желаемым паролем (рис. 2.13):

   ```
   mysqladmin -u root -p password
   ⇨ 'новый пароль'
   ```

Поскольку у пользователя root уже есть пароль, то утилита mysqladmin потребует его указать. Флаг `-p` дает mysqladmin команду запросить текущий пароль.

Способы применения утилиты mysqladmin на этом не исчерпываются: она обеспечивает, в частности, создание и удаление баз данных. Эти вопросы мы рассмотрим позже, в главе 4, при изучении клиентской программы mysql. Напоследок поговорим о том, как mysqladmin используется для опроса текущего состояния сервера.

Проверка текущего состояния сервера MySQL

1. Войдите в систему из командного интерпретатора.
2. Перейдите в каталог mysql/bin или просто в каталог mysql в зависимости от операционной системы.
3. Введите следующую команду (рис. 2.14):

```
mysqladmin -u root -p status
```

Получив приглашение `password:`, введите действующий пароль пользователя root. Если пароль набран правильно, то вы увидите, сколько времени (в секундах) процесс mysqld проработал после загрузки, а также другие статистические данные.

С Чтобы узнать, с какой версией MySQL вы работаете, введите команду

```
mysqladmin -u root -p version
```

С Чтобы проверить, работает ли сервер MySQL, не просматривая всю статистику, введите команду `mysqladmin -u root -p ping` (рис. 2.15).

П К сожалению, вы можете забыть пароль root, тем самым закрыв доступ самого важного пользователя к базе данных. О том, как вести себя в этой ситуации, рассказывается в приложении 1.

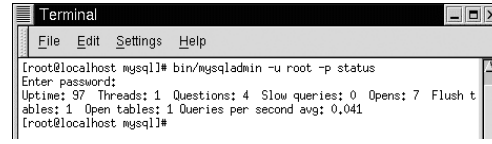

Рис. 2.14. При задании аргумента `status` утилита mysqladmin выводит информацию о текущем состоянии процесса mysqld

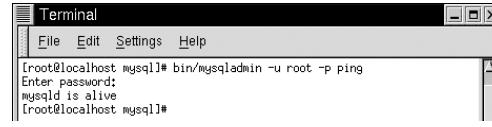

Рис. 2.15. При задании аргумента `ping` утилита mysqladmin сообщает, работает ли в данный момент сервер MySQL

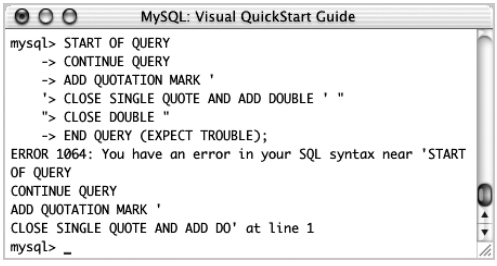

Рис. 2.16. Клиент mysql индицирует свое представление о текущем состоянии ввода различными приглашениями

Таблица 2.1. Четыре приглашения монитора, проиллюстрированные на рис. 2.16, показывают, чего ожидает mysql в данный момент

Приглашение	Расшифровка
`mysql>`	Готовность к вводу новой команды
`->`	Продолжение команды
`'>`	Необходимо закончить строку одиночной кавычкой
`">`	Необходимо закончить строку двойной кавычкой

Работа с клиентской программой mysql

Самый распространенный способ общения с сервером mysqld (если не говорить о программировании) – использование клиентской программы mysql (ее еще иногда называют монитором). Это приложение позволяет установить соединение с сервером, работающим на той же или на другой машине. Почти все упражнения в данной книге ориентированы на применение программы mysql (слово mysql, написанное строчными буквами, означает конкретное клиентское приложение, тогда как MySQL – весь комплекс программ).

Клиент mysql распознает некоторые аргументы в командной строке, в том числе имя пользователя, пароль и имя хоста (компьютера, на котором работает сервер). Вот как они задаются:

`mysql -u` *`имя_пользователя`* `-p -h` *`имя_хоста`*

Флаг `-p` дает mysql команду открыть окно ввода пароля точно так же, как в случае mysqladmin. При желании пароль можно задать прямо в командной строке, сразу после флага `-p`, но тогда он будет виден на экране, что небезопасно.

В мониторе mysql каждое предложение (команду SQL) нужно завершать точкой с запятой. Отсюда следует, что одно предложение может для наглядности размещаться на нескольких строках. Поэтому при работе вы можете встретиться с несколькими приглашениями, что проиллюстрировано на рис. 2.16. Полный список возможных приглашений приведен в табл. 2.1.

Чтобы познакомить вас с клиентом mysql, я покажу, как запустить его, выбрать рабочую базу данных и закончить сеанс. Естественно, для этого сервер (mysqld) должен быть запущен.

Порядок работы с монитором mysql

1. Войдите в систему из командного интерпретатора.
2. Перейдите в каталог mysql/bin или просто в каталог mysql в зависимости от операционной системы:

   `cd /usr/local/mysql/bin` (UNIX)

   `cd C:\mysql\bin` (Windows)
3. Введите следующую команду (рис. 2.17):

   `mysql -u` *`имя_пользователя`* `-p`

   Упоминаемый выше аргумент `-h` *`имя_хоста`* необязателен, и я предпочитаю указывать его лишь тогда, когда не могу соединиться с сервером иначе. Если вы уже задали пароль для пользователя root, как было показано ранее, то теперь в качестве имени пользователя можете ввести `root` и соответствующий пароль.
4. Выберите базу, с которой вы хотите работать (рис. 2.18):

   `USE test;`

   Команда `USE` сообщает MySQL, с какой базой данных вы собираетесь дальше работать (чтобы не приходилось вводить ее имя каждый раз). Сценарий mysql_install_db, который запускался в ходе процедуры установки, создал две базы данных: mysql и test.

   Если вы заранее знаете, с какой базой будете работать, то можете сразу указать ее имя в командной строке:

   `mysql -u` *`имя_пользователя`* `-p`
   ⇨ *`имя_базы_данных`*
5. Закончите работу с `mysql` (рис. 2.19):

   `exit`

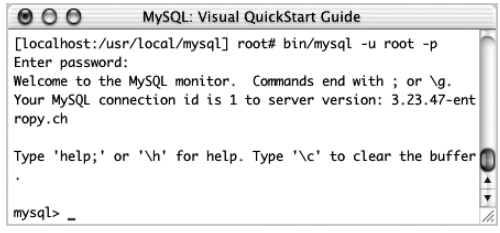

Рис. 2.17. Монитор mysql будет чаще всего использоваться в этой книге для доступа к базе данных

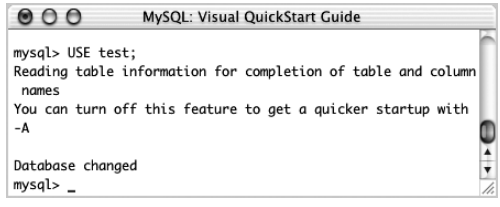

Рис. 2.18. Первый шаг при работе с клиентом mysql – выбор рабочей базы данных

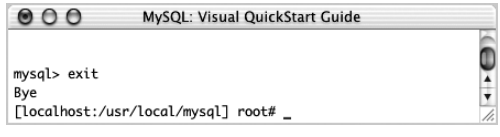

Рис. 2.19. Команды `exit` и `quit` завершают работу с монитором mysql

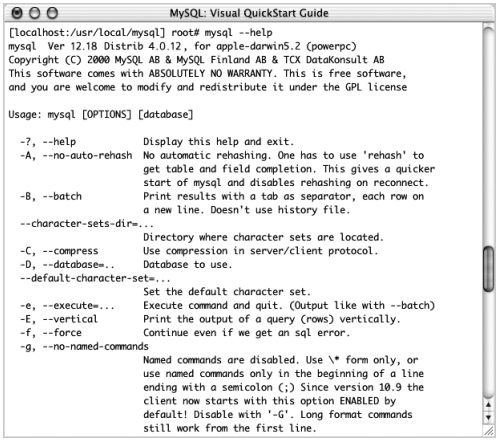

Рис. 2.20. Подробную справку о порядке запуска mysql предоставляет команда `mysql --help`

Для выхода из монитора можно также использовать команду `quit`. Эти две команды, в отличие от всех остальных, не требуют завершающей точки с запятой.

С Чтобы получить справочную информацию о возможностях программы mysql, наберите команду `mysql --help` (рис. 2.20).

П Клиент mysql использует функцию `readline` из стандартной библиотеки UNIX, которая позволяет применять клавиши со стрелками вверх и вниз для прокрутки списка ранее введенных команд. Это может существенно сократить время набора команд.

С Ускорить работу с монитором mysql помогает также клавиша **Tab**, позволяющая завершать неполные слова (наберите символ # и нажмите клавишу **Return**, чтобы узнать, какие слова можно завершать), а кроме того, сочетания **Ctrl**+**A** и **Ctrl**+**E**, перемещающие курсор в начало и конец строки соответственно.

С Если вы сделали ошибку в длинном предложении, то для отмены наберите `\c` и нажмите клавишу **Return**. Если mysql считает, что отсутствует одиночная или двойная кавычка, придется сначала ввести ее.

П Хотя вводимые в mysql команды завершаются точкой с запятой, запросы, посылаемые серверу из сценариев (написанных на PHP, Perl и других языках), не нуждаются в данном символе. Это типичная (хотя и безобидная) ошибка.

С В зависимости от того, как установлена система MySQL, для Windows допусти́м запуск монитора, равно как и других утилит, рассматриваемых в этой главе, двойным щелчком мыши по исполняемому файлу в папке mysql/bin. Кроме того, монитор можно запускать из меню **Start** ⇒ **Run** (Пуск ⇒ Выполнить).

С Если вы не уверены в себе, запускайте mysql с параметром `--i-am-a-dummy`[1]. Тогда попытка выполнить некоторые сомнительные операции будет пресечена самим монитором.

[1] Буквально «Я чайник» (англ.). – *Прим. переводчика.*

Пользователи и их права

После успешной установки и запуска MySQL, а также указания пароля пользователя root настало время добавить и других пользователей. В целях безопасности я настоятельно рекомендую создать учетные записи дополнительных пользователей, а не работать все время от имени root.

Система прав доступа в MySQL спроектирована так, что для выполнения определенных команд над определенными базами данных требуются полномочия. Именно эта технология обеспечивает посетителям Web-сайта безопасную работу с различными базами данных. У каждого пользователя MySQL есть набор прав доступа к определенным базам с конкретных хостов (компьютеров). Пользователь root – не операционной системы, а MySQL – наделен всеми правами и может создавать учетные записи других пользователей, которые в принципе могут обладать такими же правами (хотя это и не рекомендуется).

Когда пользователь пытается что-то сделать, MySQL сначала проверяет, разрешено ли ему вообще устанавливать соединение с сервером (при этом имя пользователя и его пароль сравниваются с информацией, хранящейся в таблице user базы данных mysql). Затем выясняется, есть ли у пользователя право выполнять запрошенную команду в указанной базе данных, например осуществлять выборку или вставку данных, создавать новые таблицы и т.д. Для этого MySQL сверяется с таблицами db, host, user, tables_priv и columns_priv (также в базе mysql). В табл. 2.2 перечислены права доступа, которые можно ассоциировать с каждым пользователем. Использование большинства из них будет продемонстрировано в главе 4.

Таблица 2.2. Список прав доступа, которые можно назначать пользователям MySQL

Право	Разрешает
SELECT	Выбирать данные из таблицы
INSERT	Добавлять новые строки в таблицу
UPDATE	Изменять существующие данные в таблице
DELETE	Удалять данные из таблицы
INDEX	Создавать и удалять индексы над таблицами
ALTER	Модифицировать структуру таблицы
CREATE	Создавать новые таблицы или базы данных
RELOAD	Перезагружать таблицы полномочий (как следствие, вступают в силу изменения в наборе прав доступа пользователя)
SHUTDOWN	Останавливать сервер MySQL
PROCESS	Просматривать список и останавливать процессы MySQL
FILE	Импортировать данные в таблицы из текстовых файлов
GRANT	Создавать учетные записи новых пользователей и наделять их правами
REVOKE	Отзывать права у пользователей

MySQL предоставляет несколько способов добавить пользователя и назначить ему права доступа; лично я предпочитаю делать это вручную, выполняя команду GRANT в мониторе mysql. Вот ее синтаксис:

```
GRANT права ON имя_базы_данных.* TO
⇨ имя_пользователя IDENTIFIED BY 'пароль'
```

Вместо слова *права* вы можете перечислить конкретные права из табл. 2.2 либо указать сразу все с помощью ключевого слова ALL (не рекомендуется). Сочетание *имя_базы_данных*.* определяет те базы данных и таблицы, с которыми может работать пользователь. Можно указать конкретную таблицу в виде *имя_базы_данных*.*имя_таблицы*, или разрешить доступ ко всем базам и всем таблицам (это тоже не рекомендуется). Наконец, следует задать имя и пароль пользователя.

Максимальная длина имени пользователя – 16 символов. Имена не должны содержать пробела (применяйте вместо него символ подчеркивания); кроме того, MySQL различает в них регистр символов. На длину паролей ограничений не накладывается, прописные и строчные буквы в них также различаются. Пароли могут храниться в зашифрованном виде, не допускающем восстановления исходного представления. Если опустить фразу IDENTIFIED BY '*пароль*', то пользователь сможет обращаться к серверу без пароля (категорически не рекомендуется).

И наконец, у вас есть возможность разрешить доступ пользователя только с определенных хостов. Имя хоста – это либо имя компьютера, на котором работает сервер MySQL (типичное значение – localhost), либо доменное имя компьютера, с которого пользователь осуществляет доступ. Вместо доменного имени можно указывать IP-адрес. Чтобы разрешить доступ

с одного хоста, наберите команду в такой форме:

```
GRANT права ON имя_базы_данных.* TO
⇨ имя_пользователя@localhost
⇨ IDENTIFIED BY 'пароль'
```

Чтобы разрешить доступ с любого хоста, воспользуйтесь метасимволом % (знак процента):

```
GRANT права ON имя_базы_данных.* TO
⇨ имя_пользователя@'%'
⇨ IDENTIFIED BY 'пароль'
```

Для примера я опишу добавление двух новых пользователей с различными правами.

Создание учетной записи пользователя

1. Войдите в систему из командного интерпретатора.
2. Перейдите в каталог mysql/bin или просто в каталог mysql в зависимости от операционной системы.
3. Войдите в монитор mysql:

   ```
   mysql -u имя_пользователя -p
   ```

4. Создайте две базы данных (рис. 2.21):

   ```
   CREATE DATABASE alpacas;
   CREATE DATABASE movies;
   ```

   Хотя мы еще не обсуждали создание баз данных, вышеприведенный синтаксис вполне понятен, а при наличии двух демонстрационных баз этот пример проще рассматривать.

5. Создайте учетную запись пользователя с административными правами в базе alpacas (рис. 2.22):

   ```
   GRANT SELECT, INSERT, UPDATE, DELETE,
⇨ CREATE, DROP, ALTER, INDEX, FILE ON
⇨ alpacas.* TO llama@localhost
⇨ IDENTIFIED BY 'camel';
   ```

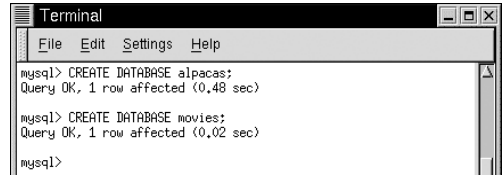

Рис. 2.21. Прежде чем добавлять пользователей, создадим две базы данных с помощью команды SQL `CREATE DATABASE имя_базы_данных`

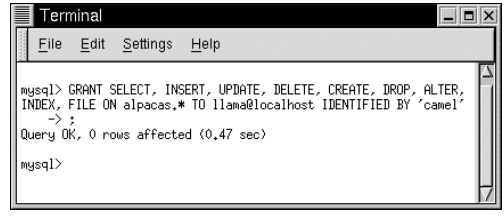

Рис. 2.22. Первый созданный нами пользователь имеет все права работы с базой данных alpacas

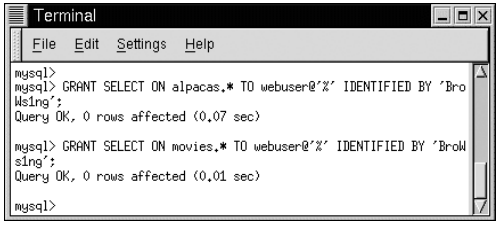

Рис. 2.23. Пользователю webuser разрешено только читать данные из таблиц

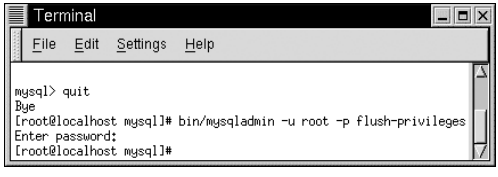

Рис. 2.24. Чтобы новые права вступили в силу, нужно актуализировать их с помощью утилиты mysqladmin

Пользователь llama может создавать и изменять таблицы, добавлять и обновлять данные и т.д. в базе alpacas. В общем, он обладает всеми правами, кроме создания новых пользователей. Обязательно укажите для llama пароль (лучше бы посложнее, чем придуманный мной); также рекомендую ограничить доступ конкретным хостом.

6. Создайте учетную запись пользователя с доступом только для чтения в обеих базах данных (рис. 2.23):

```
GRANT SELECT ON alpacas.* TO
⇨ webuser@'%' IDENTIFIED BY
⇨ 'BroWs1ng';
GRANT SELECT ON movies.* TO
⇨ webuser@'%' IDENTIFIED BY
⇨ 'BroWs1ng';
```

Пользователь webuser может выбирать данные из таблиц (выполнять предложения SELECT), но не модифицировать их. Для создания учетной записи пользователя, которому разрешено читать содержимое любой базы данных, быстрее было бы написать `GRANT SELECT ON *.* TO webuser@'%' IDENTIFIED BY 'BroWs1ng';`. Но при этом webuser смог бы осуществлять выборку и из базы mysql, что вряд ли покажется разумным. При назначении прав доступа всегда следует проявлять консерватизм, оставляя лишь необходимый минимум полномочий.

7. Выйдите из программы mysql:

```
quit
```

8. Актуализируйте изменения с помощью утилиты mysqladmin (рис. 2.24):

```
bin/mysqladmin -u root -p flush-
⇨ privileges
```

Только что внесенные изменения не вступят в силу, пока вы не попросите

MySQL перечитать список пользователей и прав доступа. Именно для этого и предназначена вышеприведенная команда. Того же эффекта можно добиться с помощью команды `bin/mysqladmin -u root -p reload` (мы часто будем встречать обе). Если о такой процедуре забыли, добавленный пользователь не может получить доступ к базе данных. Это весьма распространенная ошибка.

9. Протестируйте нового пользователя (рис. 2.25):

```
bin/mysql -u llama -p alpacas
```

С помощью этой команды вы сумеете войти в монитор и приступить к созданию таблиц в базе данных.

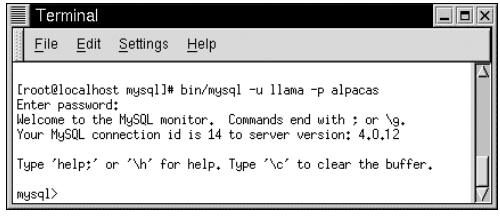

Рис. 2.25. После того как новому пользователю даны права, от его имени можно получить доступ к базе данных из программы mysql

C Любую базу данных, имя которой начинается с `test_`, может модифицировать любой пользователь, имеющий доступ к серверу MySQL. Поэтому называйте таким образом только базы, создаваемые в целях тестирования.

C Если вам не нравится создавать учетные записи пользователей и назначать им права вручную, как я показал выше, попробуйте применить утилиту mysqlaccess (это сценарий, написанный на языке Perl), которая находится в каталоге bin. Более подробную информацию вы получите, обратившись к руководству по MySQL или набрав команду `mysqlaccess --howto`.

C Есть даже более «ручной» способ создания учетной записи пользователя: прямая вставка строк (при выполнении команды `INSERT`) в таблицу user и другие таблицы базы данных mysql. Но лучше предоставить это более опытным пользователям, которые понимают отношения между таблицами user, db и прочими.

C Актуализировать изменения прав доступа можно также, набрав команду `FLUSH PRIVILEGES` в мониторе mysql.

ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

3

При работе с любой реляционной СУБД (и с MySQL, в частности) созданию и использованию базы данных должна предшествовать разработка ее структуры. Правильное проектирование базы данных или, как еще говорят, *моделирование данных* необходимо, если вы хотите добиться успеха при длительном хранении информации и управлении ею. С помощью процедуры *нормализации* устраняется избыточность и решаются другие проблемы, которые могли бы отрицательно сказаться на целостности данных.

Те методы, с которыми вы познакомитесь в настоящей главе, помогут обеспечить «живучесть», производительность и надежность ваших баз данных. Конкретный пример, который будет использоваться в последующих главах, связан с учетом деловых операций (счетов-фактур и затрат), однако изложенные принципы нормализации применимы к любому приложению, где задействованы базы данных.

Нормализация

Концепцию нормализации разработал сотрудник фирмы IBM Э. Ф. Кодд (E. F. Codd) в начале 1970-х годов (он же, кстати, является автором самой идеи реляционных баз данных). Реляционная база данных – это просто набор данных, организованный определенным образом. Доктор Кодд сформулировал несколько правил, именуемых *нормальными формами*, которые помогают ввести такую организацию. В этой главе я рассмотрю первые три нормальные формы, поскольку их достаточно для проектирования большинства баз данных.

Перед тем как приступать к нормализации данных, следует определить цели разрабатываемого приложения. Вне зависимости от того, будете ли вы в деталях обсуждать эту тему с заказчиком или примете решение самостоятельно, для построения модели нужно четко представлять себе, как будет организован доступ к информации. Поэтому при чтении данной главы вам понадобятся, скорее, ручка и бумага, а не сама СУБД MySQL (еще раз подчеркну, что принципы проектирования баз данных применимы к любой СУБД, а не только к MySQL).

В книгах, посвященных проектированию баз данных, обычно приводятся примеры, касающиеся библиотек или фонотек (не стал исключением и я, когда писал книгу «PHP Advanced for the World Wide Web: Visual Quick Pro Guide»), но сейчас я хочу создать базу, в большей степени ориентированную на деловые приложения. Ее основная цель – учет счетов-фактур и затрат, но без особого труда она может быть приспособлена для учета рабочего времени, потраченного на проект, равно как и для выполнения других задач. В табл. 3.1 приведен предварительный перечень данных, которые нужно хранить в базе.

Таблица 3.1. Здесь приведена информация, которую предстоит хранить в базе данных

Элемент данных	Пример
Invoice Number (Номер счета)	1
Invoice Date (Дата счета)	4/20/2002
Invoice Amount (Сумма счета)	$30.28
Invoive Description (Описание счета)	Проектирование HTML
Date Invoice Paid (Дата оплаты счета)	5/11/2002
Client Information (Информация о клиенте)	Acme Industries, 100 Main Street, Anytown, NY, 11111, (800) 555-1234
Expense Amount (Сумма затрат)	$100.00
Expense Category & Description (Категория и описание затрат)	Контракт с компанией Web Hosting Fees-Annual за хостинг сайта www.DMCinsights.com
Expense Date (Дата, когда была произведена затрата)	1/26/2002

С Один из лучших способов понять, что за информация должна храниться в базе, – это прояснить для себя, какие запросы будут предъявляться и какие данные следует вернуть в ответ.

П В этой книге я продемонстрирую «ручной» подход к проектированию базы данных, однако для этой цели существуют специализированные программы, некоторые из которых перечислены в приложении 3.

Ключи

Ключом называется элемент данных, идентифицирующий строку в таблице (строку также называют *записью*). Есть два типа ключей: *первичные* и *внешние*. Первичный ключ – это уникальный идентификатор, который должен удовлетворять определенным правилам, а именно:

- у ключа должно быть значение (он не может содержать `NULL`);
- это значение должно оставаться постоянным (никогда не изменяется);
- значения ключей для разных записей в таблице различны.

Реальный пример первичного ключа – это номер социального страхования в США. Хотя я слыхал истории о присвоении одинаковых номеров, но принцип состоит в том, что у каждого человека есть уникальный номер социального страхования, который не изменяется на всем протяжении его жизни. Номер социального страхования – это искусственная конструкция, призванная идентифицировать людей. Точно так же и вы зачастую будете сталкиваться с ситуацией, когда включение в таблицу искусственного первичного ключа оказывается наилучшим решением.

Внешние ключи представляют в таблице B первичные ключи из таблицы A. Если есть база данных movies (кинофильмы) с таблицами movie (фильм) и director (режиссер), то первичный ключ таблицы director будет выступать в роли внешнего в таблице movies. О том, как этот механизм используется, мы подробнее поговорим при рассмотрении процедуры нормализации.

На нынешний момент в MySQL внешние ключи реализованы только для таблиц типа InnoDB (подробнее о различных типах таблиц рассказывается в главе 11),

а в остальных случаях игнорируются. Таким образом, внешние ключи в MySQL присутствуют только теоретически, не принося никакой практической пользы, хотя в будущих версиях это положение должно измениться.

База данных accounting пока что выглядит как единственная таблица, но, чтобы приступить к нормализации, мне придется выделить хотя бы первичный ключ (внешние выявятся позже).

База данных accounting
Invoice Number **(PK)**
Invoice Date
invoice Amount
Invoice Description
Date Invoiced
Client Information
Expense Amount
Expense Category & Description
Expense Date

Рис. 3.1. Первый этап нормализации базы данных – выделение начального первичного ключа

Назначение первичного ключа

1. Найдите в таблице все поля, удовлетворяющие трем условиям первичного ключа.

   В данном примере единственное поле, которое является уникальным и имеет неизменяемое и непустое значение, – это Invoice Number (Номер счета). Пометьте это поле сочетанием PK (primary key, первичный ключ) – рис. 3.1.
2. Если никакого естественного первичного ключа нет, придется добавить искусственный.

Часто приходится создавать искусственный первичный ключ, поскольку лучшего решения не найти. Даже если вы имеете дело с номерами социального страхования и стандартными международными номерами книг (ISBN, International Standardized Book Number), которые безусловно удовлетворяют всем критериям, создание фиктивного поля для формирования первичного ключа иногда более целесообразно.

MySQL допускает наличие только одного первичного ключа в таблице, хотя он может быть образован из нескольких колонок (в данной книге мы эту возможность рассматривать не станем).

В идеале первичный ключ должен быть целым числом: это обеспечивает максимальную производительность MySQL. Вот еще одна причина, по которой номер социального страхования или ISBN (тот и другой содержат дефисы) – не лучшие кандидаты в первичные ключи.

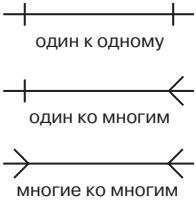

Рис. 3.2
Такие линии, помеченные стрелками, используются для представления связей между таблицами в моделях данных

Связи

Говоря о связях в базе данных, я имею в виду то, как данные из одной таблицы соотносятся с данными из других. Между таблицами бывают следующие типы связей: «один к одному», «один ко многим» и «многие ко многим».

Связь типа «один к одному» означает, что каждый элемент в таблице A соответствует одному и только одному элементу в таблице B (например, каждый гражданин США имеет один номер социального страхования и, наоборот, каждый номер социального страхования принадлежит лишь одному гражданину США).

Связь типа «один ко многим» подразумевает, что с одним элементом таблицы A может соотноситься несколько элементов таблицы B. Так, слова «мужчина» и «женщина» применимы ко многим людям, но каждый человек является либо мужчиной, либо женщиной. Связь типа «один ко многим» чаще всего встречается в базах данных.

Наконец, связь «многие ко многим» предполагает, что несколько элементов в таблице A сопоставлены с несколькими элементами таблицы B. Например, альбом записей может содержать песни нескольких исполнителей и, наоборот, у одного исполнителя может быть несколько альбомов. Следует по возможности избегать связей типа «многие ко многим», поскольку они ведут к избыточности данных и вызывают проблемы с контролем их целостности.

Связи и ключи используются совместно; обычно ключ в одной таблице соотносится с некоторым полем в другой, как уже говорилось выше. Теперь, разобравшись с уникальными идентификаторами и связями, давайте приступим к нормализации базы данных.

При моделировании данных применяются некоторые соглашения для представления структуры базы, которые будут встречаться на рисунках из этой главы. На рис. 3.2 приведены символы всех трех типов связей.

В результате проектирования базы данных создается диаграмма «сущность–связь» (ER-диаграмма), на которой таблицы представлены прямоугольниками, а связи – символами, показанными на рис. 3.2.

Термин «реляционная» происходит от английского слова relation – «отношение».

Первая нормальная форма

Чтобы база данных находилась в первой нормальной форме (1НФ), каждая колонка должна содержать ровно одно значение (часто это свойство называют *атомарностью*). Таблица, в одном поле которой содержится почтовый адрес, не может считаться приведенной к первой нормальной форме, поскольку адрес состоит из нескольких логических элементов: квартиры, дома, города, штата, почтового индекса и, возможно, страны. Не пройдет тест и поле, содержащее имя с фамилией (хотя есть люди, отстаивающие ту точку зрения, что полное имя само по себе атомарно).

Продолжим процедуру нормализации, исследовав ранее предложенную структуру на предмет совместимости с первой нормальной формой.

База данных accounting
Invoice Number **(PK)**
Invoice Date
Invoice Amount
Invoice Description
Date Invoiced
Client Name
Client Street Address
Client City
Client State
Client Zip
Client Phone
Expense Amount
Expense Category
Expense Description
Expense Date

Рис. 3.3. В результате приведения к первой нормальной форме я разбил два поля на несколько

Приведение базы данных к первой нормальной форме

1. Выявите поля, содержащие несколько элементов данных.

   В табл. 3.1 мы находим две колонки, не соответствующие требованиям первой нормальной формы: Client Information (Информация о клиенте) и Expense Category & Description (Категория и описание затрат). Поля, содержащие даты, состоят из дня, месяца и года, но дробление до такого уровня детализации уже излишне.

2. Разбейте поля, выявленные на этапе 1, на отдельные компоненты (рис. 3.3).

   Для решения проблемы я разобью поле Client Information на Client Name (Имя клиента), Client Street Address (Почтовый адрес клиента), Client City (Город проживания клиента), Client State (Штат, где проживает клиент), Client Zip (Почтовый индекс клиента) и Client Phone

(Телефон клиента). Далее, вместо поля Expense Category & Description я организую два новых: Expense Category (Категория затрат) и Expense Description (Описание затрат).

3. Еще раз убедитесь, что созданные на этапе 2 поля удовлетворяют требованиям первой нормальной формы.

Вторая нормальная форма

База данных находится во второй нормальной форме (2НФ), если она уже приведена к первой нормальной форме (нормализация производится в определенном порядке) и каждая неключевая колонка в таблице соотносится только с одним первичным ключом. Самое очевидное указание на то, что база данных не приведена ко второй нормальной форме, – это возможность существования нескольких записей с одинаковыми значениями в некоторой колонке. Например, если указывать для каждого альбома выпустившую его фирму, это значение будет многократно повторено в записях об альбомах.

Пристальное изучение базы данных accounting (рис. 3.3) вскрывает ряд проблем. Во-первых, одна и та же информация о клиенте может быть связана с несколькими счетами (одному клиенту иногда выставляют много счетов). Во-вторых, информация о затратах также не связана однозначно со счетом.

Чтобы привести базу ко второй нормальной форме, мне нужно вынести «проблемные» колонки в отдельные таблицы, где каждое значение будет представлено только один раз. На самом деле нормализацию можно описать как процесс создания все новых и новых таблиц до тех пор, пока не будет устранена всякая избыточность.

Приведение базы данных ко второй нормальной форме

1. Выявите все поля, которые не соотносятся непосредственно с первичным ключом.

   Выше я уже отмечал, что информация о клиенте и о затратах не связана непосредственно со счетом.

2. Создайте новые таблицы (рис. 3.4).

   Наиболее логичной представляется модификация существующей структуры, которая сводится к созданию отдельных таблиц Clients (Клиенты), Invoices (Счета) и Expenses (Затраты). На картинке я создал по одному прямоугольнику для каждой таблицы, поместив ее имя в заголовок, а названия колонок расположив ниже.

3. Назначьте первичные ключи из числа существующих полей или создайте искусственные ключи (рис. 3.5).

   С помощью техники, описанной выше, добейтесь, чтобы в каждой новой таблице имелся первичный ключ. Поскольку в таблицах Clients и Expenses нет подходящих уникальных идентификаторов, придется создать искусственные: Client ID (Идентификатор клиента) и Expense ID (Идентификатор затрат). Можно возразить, что значение поля Client Name (Имя клиента) должно быть уникально и, стало быть, может претендовать на роль первичного ключа, но для этой цели все же лучше использовать целые числа.

4. Повторите действия, описанные в пп. 1–3.

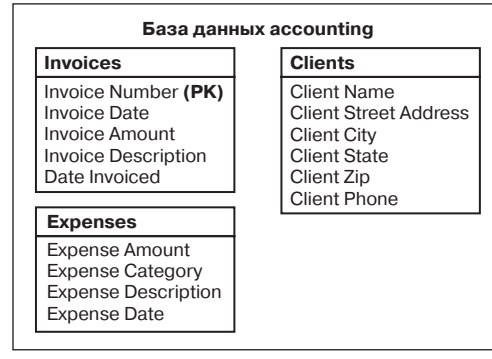

Рис. 3.4. Для нормализации базы данных следует вынести избыточную информацию – в данном случае о клиенте и о затратах – в отдельные таблицы

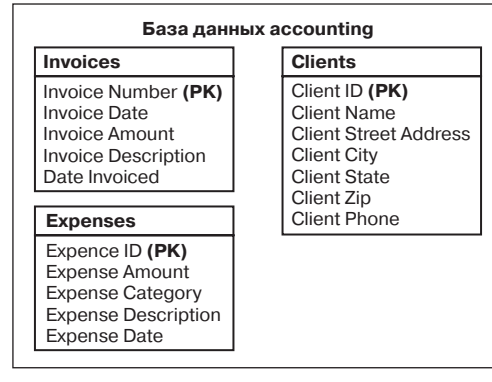

Рис. 3.5. У каждой таблицы в базе данных должен быть свой первичный ключ, искусственный (как поле Client ID) или естественный (как поле Invoice Number)

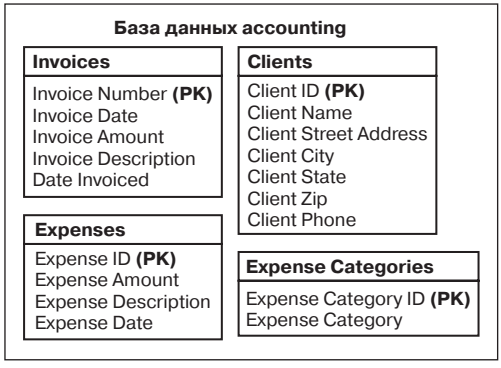

Рис. 3.6. Поле Expense Category, бывшее частью таблицы Expenses, вынесено в отдельную таблицу

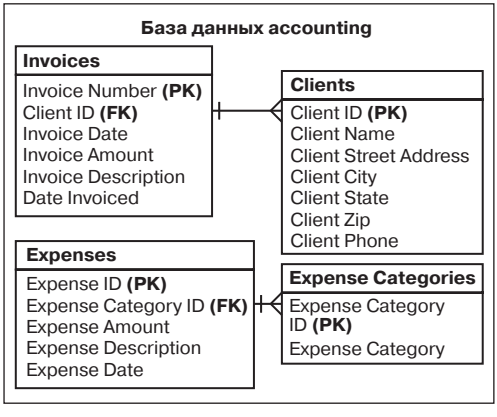

Рис. 3.7. Для новых первичных ключей я добавил соответствующие внешние ключи и обозначил связи (обе типа «один ко многим»)

Поскольку я создал таблицы с новыми первичными ключами, то должен снова проверить, удовлетворяют ли они требованиям второй нормальной формы. В нашем примере нарушение бросается в глаза: поле Expense Category может встречаться в разных записях о затратах. Поэтому я создам еще одну таблицу Expense Categories (Категории затрат) – рис. 3.6.

5. Создайте необходимые внешние ключи, обозначающие связи между таблицами (рис. 3.7).

Последний этап приведения ко второй нормальной форме – включение внешних ключей и связей, показывающих, как соотносятся данные в разных таблицах. Напомню, что первичный ключ в одной таблице, скорее всего, будет внешним в другой. Если обнаруживается, что первичному ключу в одной таблице не соответствует внешний ключ ни в одной другой, вы, может быть, что-то забыли (хотя и не обязательно).

С

Еще один способ проверить совместимость со второй нормальной формой – посмотреть на связи между таблицами. В идеале должны быть только связи «один ко многим». Таблицы, между которыми существует связь «многие ко многим», следует реорганизовать.

Третья нормальная форма

Говорят, что база данных находится в третьей нормальной форме (3НФ), если она уже приведена ко второй нормальной форме и любая неключевая колонка не зависит ни от каких других неключевых колонок. Иными словами, все поля таблицы, за исключением ключевых, должны быть взаимонезависимы.

Если вы правильно выполнили первые два шага нормализации, то, возможно, на следующем вообще не придется ничего делать. Однако, если в процессе работы были произведены какие-то изменения (как оно часто бывает), то последний этап следует рассматривать как завершающую проверку. Предположим, что я захотел вместе с каждым счетом хранить имя контактного лица и адрес электронной почты (рис. 3.8). Проблема в том, что эта информация связана не со счетом, а с клиентом, а значит, требование третьей нормальной формы не будет удовлетворено. Правильно было бы добавить поля в таблицу Clients (рис. 3.9).

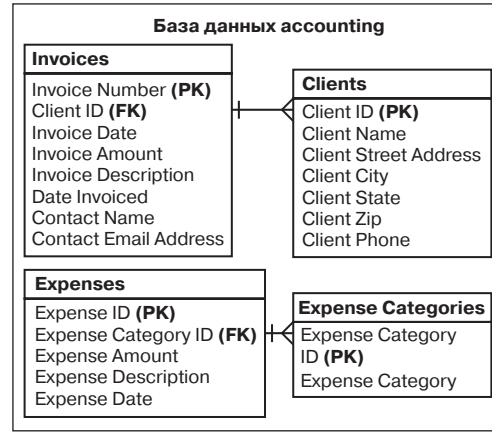

Рис. 3.8. Изменение требований к базе данных может вступить в конфликт с проектом. Так, после добавления полей Contact Name и Email Address база перестала быть нормализованной

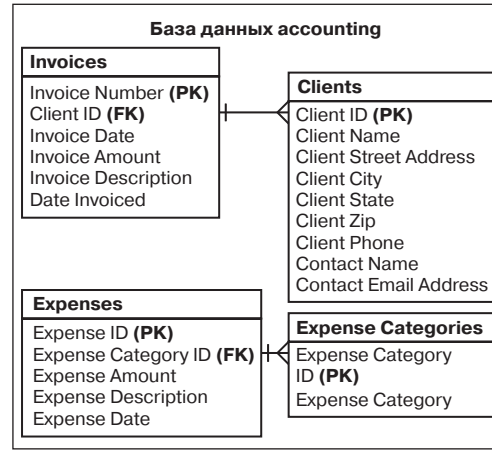

Рис. 3.9. Правильное решение – включить новую информацию (рис. 3.8) в таблицу Clients

С После того как вы нарисовали примерную структуру базы данных на бумаге, можно создать несколько электронных таблиц, описывающих проект, или воспользоваться специализированной программой. Такой файл послужил бы неплохим справочным руководством для программистов, работающих над Web-сайтом. Кроме того, его можно было бы представить заказчику по завершении работы над проектом.

П Когда MySQL начнет принимать во внимание внешние ключи (данный функционал разработчики планируют добавить в версии 4.1), нормализация базы данных приобретет еще большее значение, чем сейчас.

Ослабление требований нормализации

Хотя приведение базы данных к третьей нормальной форме повышает надежность и целостность данных, вовсе не обязательно нормализовать все без исключения таблицы, с которыми вы работаете. Но прежде чем отказываться от проверенных методик, следует осознать, что это может иметь плачевные последствия, пусть и не в самом скором будущем.

Две главных причины ослабить требования нормализации – это удобство и производительность. Чем меньше таблиц в базе, тем проще в них разобраться и манипулировать ими. Кроме того, операции обновления, выборки и модификации структуры в полностью нормализованной базе могут выполняться медленнее. Нормализация означает, что целостность данных и масштабируемость для вас важнее простоты и скорости. Есть, правда, альтернативные способы повысить производительность, но мало что может помочь при восстановлении данных, запорченных из-за неправильного проектирования.

Типы данных в MySQL

Идентифицировав все таблицы и колонки в базе данных, нужно определить тип данных, хранящихся в каждом поле. Существует три основных категории типов, общих практически для всех СУБД:

- текст;
- числа;
- дата и время.

В каждой категории есть несколько разных типов, причем некоторые из них специфичны для MySQL. Правильный выбор типа данных в колонке не только определяет, что и как в ней будет храниться, но и влияет на общую производительность. В табл. 3.2 перечислено большинство из имеющихся в MySQL типов; указано, сколько места занимает каждый, и приведено краткое описание.

У многих типов есть необязательный атрибут Length (Длина), ограничивающий размер данных. В квадратных скобках приведены необязательные параметры, которые можно указывать после имени типа в круглых скобках; если же параметр заключен в круглые скобки, то он обязателен. Кроме того, числовые типы могут иметь модификатор `UNSIGNED` (при этом поле может содержать только неотрицательные значения) или `ZEROFILL` (тогда неиспользованное место будет заполнено нулями). Если для типа указан модификатор `ZEROFILL`, то автоматически предполагается и модификатор `UNSIGNED`. Типы данных, описывающие время и дату, характеризуются собственным поведением, документированным в разделе www.mysql.com/doc/D/A/DATETIME.html руководства по MySQL. Как правило, вы будете использовать только поля типов `DATE` и `TIME`, так что беспокоиться о всяческих тонкостях не придется. У типа данных

TEXT есть два расширения со специфическим поведением: ENUM и SET. Они позволяют определить множество допустимых значений. Поле типа ENUM может принимать только одно из нескольких тысяч возможных значений, а поле типа SET – несколько значений не более чем из 64. Следует иметь в виду, что типы данных ENUM и SET не поддерживаются другими СУБД и их использование входит в противоречие с правилами нормализации.

Таблица 3.2. Основные типы данных в MySQL

Тип	Размер	Описание
CHAR[*длина*]	Число байтов равное *длина*	Поле фиксированной длины от 0 до 255 символов
VARCHAR[*длина*]	*длина* + 1 байт	Поле переменной длины от 0 до 255 символов
TINYTEXT	*длина* + 1 байт	Строка максимальной длиной 255 символов
TEXT	*длина* + 2 байта	Строка максимальной длиной 65 535 символов
MEDIUMTEXT	*длина* + 3 байта	Строка максимальной длиной 16 777 215 символов
LONGTEXT	*длина* + 4 байта	Строка максимальной длиной 4 294 967 295 символов
TINYINT[*длина*]	1 байт	Целое в диапазоне от –128 до 127 или от 0 до 255, если указан модификатор UNSIGNED
SMALLINT[*длина*]	2 байта	Целое в диапазоне от –32 768 до 32767 или от 0 до 65535, если указан модификатор UNSIGNED
MEDIUMINT[*длина*]	3 байта	Целое в диапазоне от –8 388 608 до 8 388 607 или от 0 до 16 777 215, если указан модификатор UNSIGNED
INT[*длина*]	4 байта	Целое в диапазоне от –2 147 483 648 до 2 147 483 647 или от 0 до 4 294 967 295, если указан модификатор UNSIGNED
BIGINT[*длина*]	8 байт	Целое в диапазоне от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 или от 0 до 18 446 744 073 709 551 615, если указан модификатор UNSIGNED
FLOAT	4 байта	Небольшое число с плавающей точкой
DOUBLE [*длина, десятичные знаки*]	8 байт	Большое число с плавающей точкой
DECIMAL [*длина, десятичные знаки*]	Число байтов равное *длина* + 1 или *длина* + 2	Число типа DOUBLE, хранящееся в формате с фиксированной десятичной точкой
DATE	3 байта	Дата в формате *ГГГГ-ММ-ДД* (год, месяц, день)
DATETIME	8 байт	Дата и время в формате *ГГГГ-ММ-ДД ЧЧ:ММ:СС* (год, месяц, день, часы, минуты, секунды)
TIMESTAMP	4 байта	Временной штамп в формате *ГГГГММДДЧЧММСС*; допустимый диапазон заканчивается в 2037 году
TIME	3 байта	Время в формате *ЧЧ:ММ:СС*
ENUM	1 или 2 байта	Перечисление, означающее, что в колонке может находиться одно из нескольких возможных значений
SET	1, 2, 3, 4 или 8 байт	Аналогично ENUM, но в колонке может храниться более одного значения из нескольких возможных

Выбор типа данных

1. Определите, какие колонки будут текстовыми, какие – числовыми, а какие должны содержать дату.

   Обычно эта задача не вызывает затруднений. Выясняется, что такие числовые данные, как почтовые индексы или денежные суммы, приходится хранить в текстовом виде, если включать соответствующие специальные символы (знак доллара, запятые и дефисы). Однако лучше хранить записи обычных чисел, а форматированием заняться там, где это необходимо.

2. Выберите подходящий тип для каждой колонки.

   Для повышения производительности учтите следующее:

   – работа с полями фиксированной длины (например, типа CHAR) в общем случае ведется быстрее, чем с полями переменной длины (например, типа VARCHAR), но при этом первые занимают больше места. Дополнительную информацию вы найдете во врезке «CHAR против VARCHAR»;

   – размер любого поля следует выбирать минимально возможным, исходя из того, какое максимальное значение может в нем храниться. Например, если ожидаемое число клиентов порядка нескольких сотен, то в качестве типа поля Client ID можно выбрать UNSIGNED SMALLINT с тремя цифрами (позволяет хранить значения до 999).

     Следует учитывать, что при попытке вставить строку из пяти символов в поле типа CHAR(2) последние три символа будут отброшены. Это верно в отношении любого типа с заданной длиной (CHAR, VARCHAR, INT и т.д.).

CHAR против VARCHAR

По поводу преимуществ использования того или иного из двух типов – CHAR или VARCHAR – ведутся дискуссии. Оба хранят строки и предполагают установку максимальной заданной длины. Единственное важное различие состоит в том, что все данные, хранящиеся как CHAR, всегда будут выводиться как строка, длина которой равна ширине колонки (приведение осуществляется за счет добавления пробелов в конце строки). Напротив, длина строк VARCHAR будет ровно такой, какая требуется для их записи.

Отсюда два следствия:

- колонки VARCHAR занимают меньше места на диске;
- пока вы не используете таблицы типа InnoDB (более подробно о них рассказывается в главе 11), доступ к колонкам CHAR осуществляется быстрее, чем к VARCHAR.

Впрочем, разница в быстродействии и «поглощении» дискового пространства может быть столь незначительной, что нет смысла об этом задумываться.

Наконец, есть и третье различие, не слишком существенное: MySQL удаляет лишние пробелы из колонок CHAR при выборке данных, а из колонок VARCHAR – при вставке данных.

3. Задайте максимальную длину текстовых и числовых колонок, а также другие атрибуты, к примеру UNSIGNED (табл. 3.3).

   Вместо того чтобы объяснять, как и почему я определил типы всех 21 колонок, я решил свести их в табл. 3.3. У каждого разработчика своя система предпочтений, но важнейший принцип состоит в том, чтобы принимать решения на основе имеющейся информации, а не пользоваться все время обобщенными (и неэффективными) типами TEXT и INT.

У названий многих типов данных есть синонимы: INT и INTEGER, DEC и DECIMAL и т.д.

Значение в поле типа TIMESTAMP записывается автоматически при выполнении операций INSERT и UPDATE, даже если явно оно не указано. Если в таблице есть несколько полей типа TIMESTAMP, то обновляется только первое из них.

Тип BLOB, похожий на TEXT, позволяет хранить в таблице двоичные данные. Его использование я продемонстрирую в главе 9.

Таблица 3.3. При проектировании баз данных выбору оптимального типа каждого поля часто уделяют недостаточно внимания

Имя колонки	Таблица	Тип колонки
Invoice Number (Номер счета)	Invoices (Счета)	SMALLINT(4) UNSIGNED
Client ID (Идентификатор клиента)	Invoices	SMALLINT(3) UNSIGNED
Invoice Date (Дата счета)	Invoices	DATE
Invoice Amount (Сумма счета)	Invoices	DECIMAL(10,2) UNSIGNED
Invoice Description (Описание счета)	Invoices	TINYTEXT
Client ID (Идентификатор клиента)	Clients (Клиенты)	SMALLINT(3) UNSIGNED
Client Name (Имя клиента)	Clients	VARCHAR(40)
Client Street Address (Почтовый адрес клиента)	Clients	VARCHAR(80)
Client City (Город проживания клиента)	Clients	VARCHAR(30)
Client State (Штат, где проживает клиент)	Clients	CHAR(2)
Client Zip (Почтовый индекс клиента)	Clients	MEDIUMINT(5) UNSIGNED
Client Phone (Телефон клиента)	Clients	VARCHAR(14)
Contact Name (Контактное лицо)	Clients	VARCHAR(40)
Contact Email Address (Адрес электронной почты)	Clients	VARCHAR(60)
Expense ID (Идентификатор затрат)	Expenses (Затраты)	SMALLINT(4) UNSIGNED
Expense Category ID (Идентификатор категории затрат)	Expenses	TINYINT(3) UNSIGNED
Expense Amount (Количество затрат)	Expenses	DECIMAL(10,2) UNSIGNED
Expense Description (Описание затраты)	Expenses	TINYTEXT
Expense Date (Дата, когда была произведена затрата)	Expenses	DATE
Expense Category ID (Идентификатор категории затрат)	Expense Categories (Категории затрат)	TINYINT(3) UNSIGNED
Expense Category (Категория затрат)	Expense Categories	VARCHAR(30)

- Первичные ключи не могут содержать NULL, согласно принципам правильного проектирования базы данных (да MySQL и не допустит этого).
- Если для колонки типа ENUM указан атрибут NOT NULL, то значением по умолчанию автоматически становится первое из допустимых значений.
- На всякий случай подчеркну, что NULL отличается от числа 0, от пустой строки ('') и от пробела (' ').

NULL и значения по умолчанию

Итак, вы уже познакомились с несколькими атрибутами полей, определяемыми при выборе типа данных, например UNSIGNED и ZEROFILL. Есть еще два атрибута, которые показывают, может ли поле содержать NULL и каково его значение по умолчанию.

В контексте баз данных и программирования фраза «значение поля равно NULL» эквивалентна фразе «поле не имеет значения». В идеале каждое поле базы данных должно иметь значение, но на практике так бывает редко. Если вы все же предпочитаете идеальный вариант, добавьте NOT NULL в описание типа колонки. Например, первичный ключ можно было бы описать так:

```
client_id SMALLINT(3) UNSIGNED NOT NULL
```

При создании таблицы для некоторых колонок можно также задать значения по умолчанию. В тех случаях, когда во многих записях некоторое поле будет иметь одно и то же значение, можно сделать его значением по умолчанию – тогда не нужно будет указывать его в операторе вставки новой строки, если, конечно, не вставляется какое-либо другое значение. Вот пример:

```
gender ENUM('M', 'F') DEFAULT 'F'
```

В табл. 3.4 отражены соответствующие изменения.

Таблица 3.4. Для улучшения проекта базы данных я добавил к описаниям некоторых колонок атрибуты NOT NULL и DEFAULT

Имя колонки	Таблица	Тип колонки
Invoice Number (Номер счета)	Invoices (Счета)	SMALLINT(4) UNSIGNED ⇨ NOT NULL DEFAULT 0
Client ID (Идентификатор клиента)	Invoices	SMALLINT(3) UNSIGNED
Invoice Date (Дата счета)	Invoices	DATE NOT NULL
Invoice Amount (Сумма счета)	Invoices	DECIMAL(10,2) UNSIGNED NOT NULL
Invoice Description (Описание счета)	Invoices	TINYTEXT
Client ID (Идентификатор клиента)	Clients (Клиенты)	SMALLINT(3) UNSIGNED ⇨ NOT NULL DEFAULT 0
Client Name (Имя клиента)	Clients	VARCHAR(40) NOT NULL
Client Street Address (Почтовый адрес клиента)	Clients	VARCHAR(80)
Client City (Город проживания клиента)	Clients	VARCHAR(30)
Client State (Штат, где проживает клиент)	Clients	CHAR(2)
Client Zip (Почтовый индекс клиента)	Clients	MEDIUMINT(5) UNSIGNED
Client Phone (Телефон клиента)	Clients	VARCHAR(14)
Contact Name (Контактное лицо)	Clients	VARCHAR(40)
Contact Email Address (Адрес электронной почты)	Clients	VARCHAR(60)
Expense ID (Идентификатор затрат)	Expenses (Затраты)	SMALLINT(4) UNSIGNED ⇨ NOT NULL DEFAULT 0
Expense Category ID (Идентификатор категории затрат)	Expenses	TINYINT(3) UNSIGNED
Expense Amount (Количество затрат)	Expenses	DECIMAL(10,2) UNSIGNED NOT NULL
Expense Description (Описание затраты)	Expenses	TINYTEXT
Expense Date (Дата, когда была произведена затрата)	Expenses	DATE
Expense Category ID (Идентификатор категории затрат)	Expense Categories (Категории затрат)	TINYINT(3) UNSIGNED
Expense Category (Категория затрат)	Expense Categories	VARCHAR(30)

Индексы

Индексы – это специальный механизм, который используется в СУБД для повышения производительности. Строя над таблицей индексы по некоторым полям, вы сообщаете MySQL, что на эти поля надо обратить особое внимание. На самом деле в целях эффективности программа хранит индексы в отдельных файлах.

MySQL позволяет построить над каждой таблицей до 32 индексов, а каждый индекс может включать до 16 полей. Хотя на первый взгляд непонятно, зачем нужны многоколонные индексы, их полезность становится очевидной, если часто приходится производить поиск по одному и тому же набору колонок (например, по имени и фамилии, городу и штату и т.д.).

С другой стороны, с индексированием не стоит перебарщивать. Конечно, индексы ускоряют выборку данных из базы, но зато замедляют обновление (поскольку изменять приходится не только сами данные, но и индексы). Лучше всего строить индексы по колонкам, которые:

- часто используются в условии `WHERE` запроса;
- часто используются во фразе `ORDER BY` запроса;
- имеют много различных значений (колонки, где встречается много повторяющихся значений, индексировать не стоит).

Отметим, что в MySQL над колонкой, содержащей первичный ключ, индекс строится автоматически.

В MySQL есть три типа индексов: INDEX, UNIQUE (применяется, когда все значения ключа должны быть уникальны) и PRIMARY KEY (разновидность индекса типа UNIQUE). В табл. 3.5 приведены индексы, которые я предлагаю использовать для базы данных accounting.

Последний атрибут колонки, который часто используется в сочетании с индексами, – это AUTO_INCREMENT. При определении поля с таким свойством, например:

```
client_id SMALLINT(3) UNSIGNED
⇨ NOT NULL AUTO_INCREMENT
```

вы сообщаете MySQL, что при вставке новой записи в это поле нужно записать «следующее» значение. Если речь идет о числовом поле, то «следующим» считается наименьшее число, превышающее максимальное значение из тех, что уже находятся в этой колонке.

П Атрибут AUTO_INCREMENT в MySQL эквивалентен последовательностям в Oracle.

П Использование индексов, построенных по полям переменной длины, менее эффективно, да и вообще MySQL с такими полями всегда работает медленнее, чем с полями фиксированной длины.

С Создаваемому индексу можно присвоить имя (см. главу 4), а если вы этого не сделаете, то по умолчанию оно будет совпадать с именем той колонки, по которой строится индекс.

Таблица 3.5. Для повышения производительности я добавил в базу accounting несколько (не очень много) индексов, ускоряющих доступ к информации

Колонка	Тип индекса
Invoice Number (Номер счета)	PRIMARY KEY
Client ID (Идентификатор клиента)	PRIMARY KEY
Expense ID (Идентификатор затрат)	PRIMARY KEY
Expense Category ID Идентификатор категории затрат)	PRIMARY KEY
Invoice Date (Дата счета)	INDEX
Client Name (Имя клиента)	INDEX или UNIQUE

Заключительные этапы проектирования

Последнее, о чем стоит упомянуть в связи с проектированием баз данных, – это необходимость придерживаться некоторых соглашений об именовании. Хотя MySQL позволяет присваивать базам данных, таблицам и колонкам достаточно произвольные имена, я все-таки рекомендую следовать перечисленным ниже правилам (некоторые из них обязательны):

- используйте только алфавитно-цифровые символы;
- не употребляйте имен длиннее 64 символов (это ограничение MySQL);
- пользуйтесь для разделения слов символом подчеркивания;
- употребляйте в словах только маленькие буквы (а вот это вопрос личных предпочтений, а не четкое правило);
- в именах таблиц употребляйте множественное число, а в названиях колонок – единственное;
- названия колонок, соответствующих первичным и внешним ключам, завершайте словом `id` (или `ID`);
- подбирайте смысловые названия колонок;
- старайтесь, чтобы имена всех колонок во всех таблицах различались, за исключением названий ключевых колонок.

По большей части это мои личные рекомендации, которым, конечно, не обязательно следовать. Некоторые разработчики предпочитают начинать каждое слово в имени прописными буквами, а не разделять подчеркиванием (например, ContactName вместо Contact_Name). Другие включают в название колонки ее тип. Главное, чтобы вы

всюду придерживались единожды выбранного стиля.

В табл. 3.6 приведена окончательная структура базы данных, которую мы создадим в следующей главе (переводы названий колонок и таблиц см. в табл. 3.3, 3.4).

П В системе UNIX регистр символов для имен баз данных и таблиц имеет значение, в отличие от системы Windows. В именах колонок прописные и строчные буквы не различаются на любой платформе.

С Строго придерживаясь принципов проектирования баз данных, вы сведете к минимуму число ошибок при программировании интерфейса базы (см. главы 6–8).

Таблица 3.6. В окончательном проекте базы данных учтены некоторые соглашения об именовании, которыми не стоит пренебрегать

Имя колонки	Таблица	Тип колонки
invoice_id	invoices	`SMALLINT(4) UNSIGNED NOT NULL DEFAULT 0`
client_id	invoices	`SMALLINT(3) UNSIGNED`
invoice_date	invoices	`DATE NOT NULL`
invoice_amount	invoices	`DECIMAL(10,2) UNSIGNED NOT NULL`
invoice_description	invoices	`TINYTEXT`
client_id	clients	`SMALLINT(3) UNSIGNED NOT NULL DEFAULT 0`
client_name	clients	`VARCHAR(40) NOT NULL`
client_street	clients	`VARCHAR(80)`
client_city	clients	`VARCHAR(30)`
client_state	clients	`CHAR(2)`
client_zip	clients	`MEDIUMINT(5) UNSIGNED`
client_phone	clients	`VARCHAR(14)`
contact_name	clients	`VARCHAR(40)`
contact_email	clients	`VARCHAR(60)`
expense_id	expenses	`SMALLINT(4) UNSIGNED NOT NULL DEFAULT 0`
expense_category_id	expenses	`TINYINT(3) UNSIGNED`
expense_amount	expenses	`DECIMAL(10,2) UNSIGNED NOT NULL`
expense_description	expenses	`TINYTEXT`
expense_date	expenses	`DATE`
expense_category_id	expense_categories	`TINYINT(3) UNSIGNED`
expense_category	expense_categories	`VARCHAR(30)`

4

Язык SQL

Язык SQL (Structured Query Language – структурированный язык запросов) предназначен специально для взаимодействия с базами данных. Он используется во всех сколько-нибудь значимых СУБД, и MySQL – не исключение.

Язык SQL был создан вскоре после того, как Э. Ф. Кодд выдвинул свою теорию реляционных баз данных. Спустя десять лет, в 1989 году, Национальный институт стандартизации США (ANSI) выпустил первый стандарт SQL, который получил название SQL 89. В 1992 году вышел стандарт SQL2, который и поныне считается актуальным (его еще называют SQL92 или просто SQL).

На примере базы данных accounting, спроектированной в предыдущей главе, я продемонстрирую все важнейшие возможности SQL. Поскольку для взаимодействия с MySQL этот язык необходим, изложенная в данной главе информация пригодится вам и при чтении оставшейся части книги.

Создание баз данных и таблиц

Первый наш опыт применения SQL будет заключаться в создании базы данных. В главе 2 я без пространных пояснений уже создал две базы, чтобы показать на примере, как добавляются пользователи. Тогда же вы познакомились с командой GRANT языка SQL. Напомню синтаксис предложения для создания базы данных:

```
CREATE DATABASE имя_базы_данных;
```

Слово CREATE используется также для создания таблиц:

```
CREATE TABLE имя_таблицы (имя_колонки1
⇨ описание, имя_колонки2 описание, . . . );
```

Как вы видите, после указания имени таблицы внутри скобок определяются ее колонки – в порядке следования. Описания колонок отделены друг от друга запятыми. В конце предложения, используемого для создания таблицы, можно указать имена индексов по ней (вы вправе сделать это позже в отдельной команде).

Для демонстрации предложения CREATE я создам базу данных accounting и четыре входящих в нее таблицы, учитывая правила нормализации, изложенные в главе 3.

Создание баз данных и таблиц

1. Далее в этой главе все предложения SQL будут вводиться с помощью монитора mysql. Откройте mysql, применяя синтаксис (имя пользователя, пароль и т.д.), специфичный для вашей операционной системы и конфигурации, – см. главу 2.
2. Создайте новую базу данных и сделайте ее рабочей (рис. 4.2):

```
CREATE DATABASE accounting;
USE accounting;
```

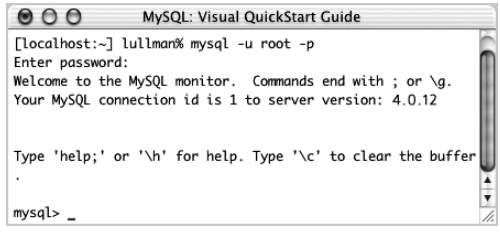

Рис. 4.1. Все примеры из этой главы вводятся с помощью клиента mysql

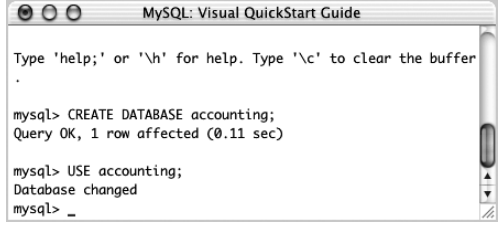

Рис. 4.2. Первым делом я создаю и назначаю рабочей базу данных accounting

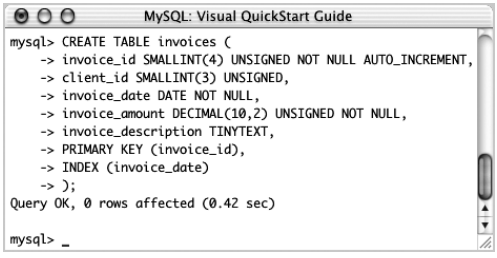

Рис. 4.3. Монитор mysql позволяет записывать команды в нескольких строках: так они выглядят понятнее

В первой строке создается база данных (предполагается, что вы вошли в mysql от имени пользователя, обладающего правами на создание новых баз данных). Во второй строке вы сообщаете MySQL, что далее будете работать с только что созданной базой. Напомню, что в мониторе mysql все предложения должны завершаться точкой с запятой.

Хотя в языке SQL прописные и строчные буквы не различаются, лично я привык записывать зарезервированные слова SQL прописными буквами, что позволяет легко отличить их от имен баз данных, таблиц и колонок. Если вам больше нравится противоположный вариант, пишите зарезервированные слова строчными буквами.

3. Создайте таблицу invoices (рис. 4.3):

```
CREATE TABLE invoices (
invoice_id SMALLINT(4) UNSIGNED NOT
⇨ NULL AUTO_INCREMENT,
client_id SMALLINT(3) UNSIGNED,
invoice_date DATE NOT NULL,
invoice_amount DECIMAL(10,2)
⇨ UNSIGNED NOT NULL,
invoice_description TINYTEXT,
PRIMARY KEY(invoice_id),
INDEX(invoice_date)
);
```

Структура таблицы invoices была разработана в предыдущей главе. Последовательность перечисления колонок определяет порядок хранения данных в таблице. Индексы должны быть указаны в самом конце, когда имя и описание соответствующей колонки уже известны.

Поскольку mysql не начнет выполнять запрос, пока не встретит точку с запятой, одно предложение SQL можно записывать в нескольких строках, как показано на рис. 4.3.

При создании таблицы вы можете задать ее тип. Чаще всего используются типы MyISAM, BDB, InnoDB, temporary и HEAP. Если тип таблицы не указан, то по умолчанию MySQL создает таблицу типа MyISAM (это усовершенствованный тип ISAM, который присутствовал в предыдущих версиях). В главе 11 мы подробнее познакомимся с типом InnoDB.

4. Создайте остальные три таблицы (см. рис. 4.4):

```
CREATE TABLE clients (
   client_id SMALLINT(3) UNSIGNED
   ⇨ NOT NULL AUTO_INCREMENT,
   client_name VARCHAR(40) NOT NULL,
   client_street VARCHAR(80),
   client_city VARCHAR(30),
   client_state CHAR(2),
   client_zip MEDIUMINT(5) UNSIGNED,
   client_phone VARCHAR(14),
   contact_name VARCHAR(40),
   contact_email VARCHAR(60),
   PRIMARY KEY(client_id),
   INDEX(client_name)
);
CREATE TABLE expenses (
   expense_id SMALLINT(4) UNSIGNED
   ⇨ NOT NULL AUTO_INCREMENT,
   expense_category_id TINYINT(3)
   ⇨ UNSIGNED,
   expense_amount DECIMAL(10,2)
   ⇨ UNSIGNED,
   expense_date DATE,
   PRIMARY KEY(expense_id)
);
CREATE TABLE expense_categories (
   expense_category_id TINYINT(3)
   ⇨ UNSIGNED NOT NULL AUTO_INCREMENT,
   expense_category VARCHAR(30),
   PRIMARY KEY(expense_category_id)
);
```

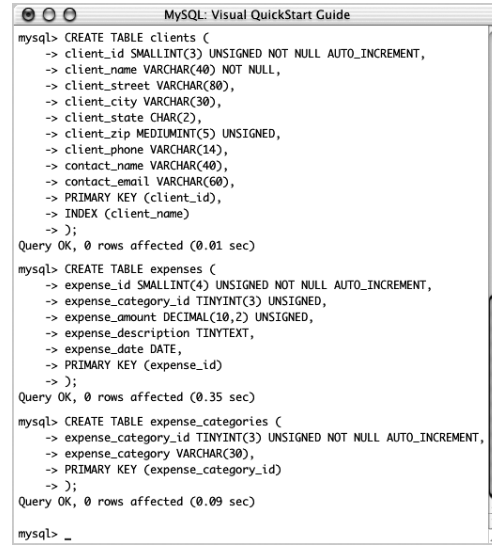

Рис. 4.4. Об успешном завершении mysql сообщает, печатая сообщения `Query OK`

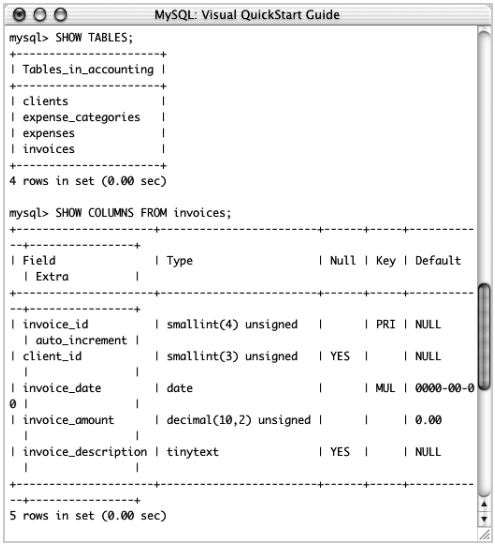

Рис. 4.5. Команда `SHOW` позволяет проверить наличие таблиц и их структуру

Синтаксис создания таблиц все тот же – изменяются только имена и описания полей.

5. Проверьте, что таблицы были успешно созданы (рис. 4.5):

```
SHOW TABLES;
SHOW COLUMNS FROM invoices;
```

Детально обсуждать команду `SHOW` не имеет смысла; достаточно сказать, что она выводит имена таблиц в базе данных или названия и типы колонок указанной таблицы.

С Напомню, что базы данных (но не таблицы!) можно также создавать с помощью утилиты mysqladmin:

```
mysqladmin -u root -p create
⇨ имя_базы_данных
```

П Далее в этой главе я буду исходить из предположения, что вы работаете с клиентом mysql и уже выбрали базу accounting в качестве рабочей.

С Указать рабочую базу данных можно прямо при запуске mysql – тогда не придется вводить команду `USE`:

```
mysql -u root -p имя_базы_данных
```

П Запрос `DESCRIBE имя_таблицы` аналогичен команде `SHOW COLUMNS FROM имя_таблицы`.

Вставка данных

После того как база данных и таблицы созданы, можно начать заполнять их с помощью команды INSERT. У нее два формата. В первом случае вы явно указываете имена колонок:

```
INSERT INTO имя_таблицы (имя_колонки1,
⇨ имя_колонки2, ...)
⇨ VALUES('значение1', 'значение2', ...);

INSERT INTO имя_таблицы (имя_колонки4,
⇨ имя_колонки8)
⇨ VALUES('значениеX', 'значениеY');
```

Таким способом вы можете добавлять строки, вводя значения лишь тех колонок, которые необходимо заполнить. Остальные колонки будут содержать NULL или значение по умолчанию, если оно было задано в описании таблицы. Если же колонка не может содержать NULL и значение по умолчанию не определено, то при попытке вставить строку без ввода значения этой колонки возникнет ошибка.

Во втором случае имена колонок не указываются вовсе, зато перечисляются значения каждой из них:

```
INSERT INTO имя_таблицы VALUES
⇨ ('значение1', NULL, 'значение3', ...);
```

Применяя второй способ, вы должны ввести значение для каждой колонки, включая NULL; к примеру, если в таблице шесть колонок, то должно быть перечислено шесть значений. При несовпадении типов данных или неправильном количестве значений команда выдает ошибку, поэтому первый способ вставки записей в общем предпочтителен.

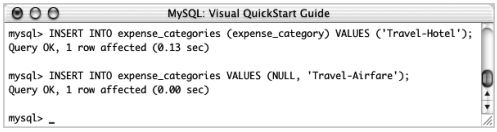

Рис. 4.6. Два способа вставки записей в таблицу

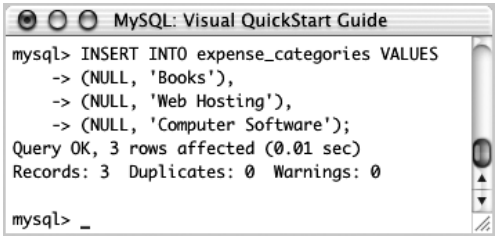

Рис. 4.7. MySQL позволяет вставлять в одно предложение `INSERT` сразу несколько записей

MySQL также позволяет одновременно вставлять несколько строк, разделяя записи запятыми:

```
INSERT INTO имя_таблицы (имя_колонки1,
⇨ имя_колонки4) VALUES('значениеA',
⇨ 'значениеB'),
⇨ ('значениеC', 'значениеD'),
⇨ ('значениеE', 'значениеF');
```

Однако стандарт ANSI SQL2 не допускает такой записи.

Вставка записей в таблицу

1. Вставьте новую строку в таблицу expense_categories (рис. 4.6).

   Можно воспользоваться любым из следующих предложений:

   ```
   INSERT INTO expense_categories
   ⇨ (expense_category)
   ⇨ VALUES('Travel-Hotel');

   INSERT INTO expense_categories
   ⇨ VALUES(NULL, 'Travel-Airfare');
   ```

   Поскольку в таблице есть всего две колонки, одна из которых автоинкрементная, использование обоих методов дает один и тот же результат. По рис. 4.6 видно, что второй запрос MySQL выполняет быстрее, чем первый, хотя и ненамного.

2. Вставьте несколько значений в таблицу expense_categories (рис. 4.7):

   ```
   INSERT INTO expense_categories VALUES
   (NULL, 'Books'),
   (NULL, 'Web Hosting'),
   (NULL, 'Computer Software');
   ```

   Мы воспользовались тем, что MySQL допускает присутствие в одном предложении `INSERT` сразу нескольких записей.

3. Повторяйте действия, описанные в пп. 1 и 2, пока таблицы expense_categories и clients не будут заполнены.

   Сейчас я не стану помещать записи в таблицы invoices и expenses, поскольку они зависят от информации в таблицах expense_categories и clients. Позже мы рассмотрим и эту процедуру.

С Если нужно вставить значение, содержащее одиночную кавычку, экранируйте ее обратной косой чертой:

```
INSERT INTO users(last_name,
⇨ first_name) VALUES('O\'Malley',
⇨ 'Juan');
```

П Пробелы в конце значения, помещенного в колонку типа VARCHAR, MySQL автоматически удаляет – это еще одно отклонение от стандарта ANSI SQL2.

П В текущей версии MySQL слово INTO в предложении INSERT не обязательно.

С Рекомендую заключать строковые значения в одиночные кавычки, а числовые указывать вообще без кавычек (последнее относится и к NULL).

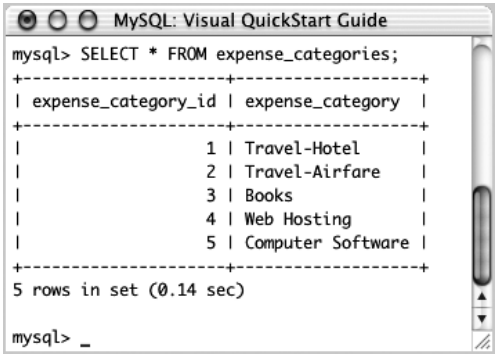

Рис. 4.8. Простой запрос SELECT возвращает все строки и колонки из таблицы

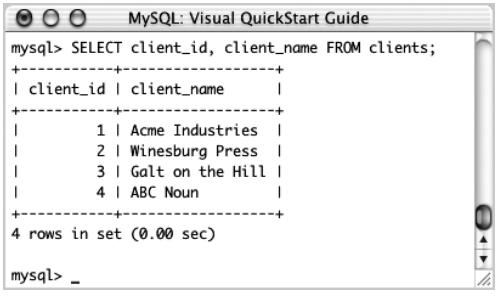

Рис. 4.9. Можно ограничить объем запрашиваемой информации, указав имена интересующих вас колонок

Выборка данных

Итак, теперь в таблицах базы данных есть несколько записей, и можно приступать к выборке данных с помощью одной из самых распространенных команд SQL – SELECT. Она возвращает набор строк, удовлетворяющих заданному критерию.

В простейшем виде запрос выглядит так:

```
SELECT * FROM имя_таблицы;
```

Звездочка означает, что вас интересуют значения во всех колонках. Альтернативный вариант – явное перечисление имен колонок, разделяемых запятыми:

```
SELECT user_id, first_name, last_name
⇨ FROM users;
```

У последнего способа несколько преимуществ. Первое – это производительность. Не имеет смысла отбирать данные из колонок, которые вам не нужны. Второе преимущество – порядок следования: вы можете вернуть колонки не в том порядке, в котором они фактически хранятся в таблице. Наконец, третье – возможность «на лету» преобразовать значения, о чем пойдет речь в главе 5.

Выборка данных из таблицы

1. Выберите все данные из таблицы expense_categories (рис. 4.8):

   ```
   SELECT * FROM expense_categories;
   ```

   По этой простейшей команде выбираются и выводятся на экран значения всех колонок из всех строк таблицы.

2. Выберите только поля client_id и client_name из таблицы clients (рис. 4.9).

   Вместо того чтобы запрашивать все поля из таблицы clients, вы можете ограничиться только теми, что вас интересуют.

3. Пользуясь информацией, выбранной на этапах 1 и 2, заполните таблицы expenses и invoices (рис. 4.10):

```
INSERT INTO expenses VALUES
(NULL, 3, 19.99, 'Larry Ullman\'s
⇨ "MySQL: Visual QuickStart
⇨ Guide" ', '2002-04-20'),
(NULL, 1, 104.50, 'Palmer House
⇨ Hotel, Chicago', '2002-1-26'),
(NULL, 2, 689.90, 'Flight to
⇨ Chicago', '2002-1-26'),
(NULL, 5, 99.99,
⇨ 'Mmmm...software', NULL);
```

Когда известны первичные ключи в таблицах expense_categories и clients (expense_category_id и client_id соответственно), можно добавлять данные в две другие таблицы. Поскольку база данных реляционная, важно, чтобы записи соотносились друг с другом, то есть первичный ключ в одной таблице совпадал с внешним ключом в другой. Поэтому, чтобы указать, что код затрат равен Book (Книга), мы вводим в поле expense_category_id значение 3. Поддержание таких связей – это основа основ нормализованных баз данных.

Заметьте, что в колонку типа DATE можно вводить дату в различных форматах, например 2002-04-07, 20020407 или 2002-4-7 (так обозначается седьмое апреля 2002 года).

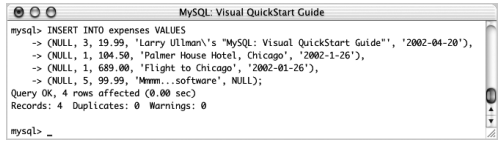

Рис. 4.10. После того как значения внешних ключей в поле expense_category_id стали известны, можно приступить к заполнению таблицы expenses

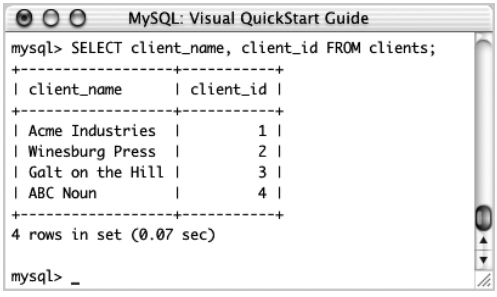

Рис. 4.11. При изменении порядка колонок в запросе SELECT изменяется и порядок следования возвращаемых данных

4. Повторяйте вышеописанные действия, пока в каждой таблице базы данных не окажется достаточно много записей.

   В оставшейся части этой главы я буду выполнять запросы, ориентируясь на информацию, которая приводилась в предыдущих примерах. Если вы ввели другие данные (иных клиентов, категории затрат и т.д.), учтите различия в формулировке запросов. Принципы их составления, впрочем, не зависят от конкретных данных, так как структура базы у нас с вами должна быть одинакова.

С Как ни странно, в предложении SELECT необязательно указывать имена таблиц и колонок. Соответствующий пример будет приведен в следующей главе.

П Порядок перечисления колонок в предложении SELECT (если, конечно, вы извлекаете не все колонки) определяет и последовательность значений в возвращенных строках. Сравните рис. 4.9 с рис. 4.11.

С С помощью предложения SELECT одну колонку можно выбирать несколько раз, выполняя различные ее преобразования.

П В следующих четырех главах будет рассказано о более простых способах вставки записей.

Использование условий

Приведенные до сих пор предложения SELECT выбирали все записи из таблицы. Если строк немного, это не проблема, но по мере их увеличения производительность начнет заметно падать. Чтобы увеличить скорость выборки, можно указывать условия в самых разных комбинациях. Условия формулируются во фразе WHERE предложения SELECT и записываются примерно так же, как в языках программирования:

```
SELECT * FROM имя_таблицы WHERE
⇨ имя_колонки = 'значение';
SELECT expense_amount FROM expenses
⇨ WHERE expense_amount >= 10.00;
SELECT client_id FROM clients WHERE
⇨ client_name = 'Acme Industries';
```

В табл. 4.1 перечислены операторы, чаще всего употребляемые в условиях запросов. Ниже в этой главе я продемонстрирую использование операторов LIKE и NOT LIKE (см. раздел «Использование операторов LIKE и NOT LIKE»), а в главе 11 – применение оператора REGEX и выполнение полнотекстового поиска. Операторы из табл. 4.1 можно употреблять совместно для постановки сложных условий:

```
SELECT expense_amount FROM expenses
⇨ WHERE (expense_amount >= 10.00)
⇨ AND (expense_amount <= 20.00);
SELECT * FROM expenses WHERE
⇨ (expense_category_id=1) OR
⇨ (expense_category_id=2);
```

Чтобы показать, как используются условия, я выберу из базы более конкретные данные. Представленные ниже фрагменты кода иллюстрируют лишь малую часть возможностей. В этой и последующих главах вы увидите еще много примеров условий в предложении SELECT.

Таблица 4.1. Часто употребляемые операторы MySQL (полный их перечень приведен в приложении 2)

Оператор	Назначение
=	Равно
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно
!=	Не равно
IS NOT NULL	Имеет значение (в том числе равное пустой строке или нулю)
IS NULL	Не имеет значения
BETWEEN	Внутри диапазона
NOT BETWEEN	Вне диапазона
OR *(также* ||*)*	Хотя бы один из операндов равен true
AND *(также* &&*)*	Оба операнда равны true
NOT *(также* !*)*	Условие ложно

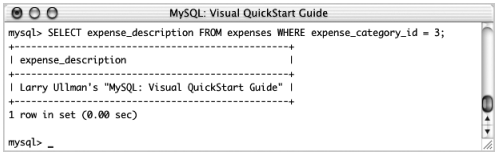

Рис. 4.12. В условии из этого запроса используется значение внешнего ключа expense_category_id в таблице expenses для отбора конкретных записей

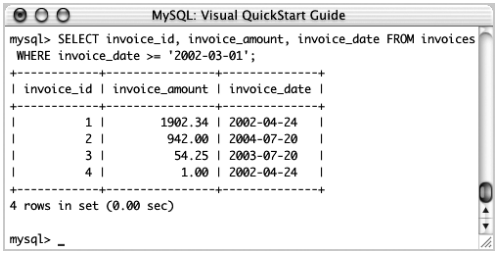

Рис. 4.13. С полями типа DATE можно обращаться весьма гибко, в частности сравнивать их

Выборка данных, удовлетворяющих условиям

1. Выберите поле expense_description для каждой записи с типом затрат Book (рис. 4.12).

```
SELECT expense_description
⇨ FROM expenses
⇨ WHERE expense_category=3;
```

   Поскольку я знаю, что числовое значение кода затрат Book в поле expense_category_id таблицы expense_categories равно 3, то могу записать показанный выше запрос. Он вернет поле expense_description из каждой записи, у которой в поле expense_category_id значится 3. Заметьте, что числа не следует заключать в кавычки ни в условии WHERE, ни в какой-либо другой части запроса. Если при заполнении таблиц вы не ввели тип затрат Book, модифицируйте запрос.

2. Выберите идентификатор, сумму и дату каждого счета, введенного начиная с 1 марта 2002 года (рис. 4.13):

```
SELECT invoice_id, invoice_amount,
⇨ invoice_date FROM invoices WHERE
⇨ invoice_date >= '2002-03-01';
```

   Для дат разрешено применять операторы «больше», «меньше», а также «больше либо равно» и «меньше либо равно».

3. Выберите из таблицы expenses все записи, где не указана дата (рис. 4.14):

```
SELECT * FROM expenses WHERE
⇨ expense_date IS NULL;
```

Условие IS NULL означает «без значения». Не забывайте, что пустая строка – это все равно значение, поэтому условию IS NULL она не удовлетворяет. В этом случае следовало бы записать

```
SELECT * FROM expenses WHERE
⇨ expense_date = '';
```

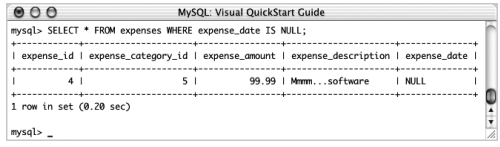

Рис. 4.14. Значение NULL имеет особый смысл в базах данных, поэтому для сравнения с ним используются специальные операторы IS NULL и IS NOT NULL

П Как ни странно, в списке отбираемых колонок необязательно указывать те, которые упомянуты в условии WHERE:

```
SELECT invoice_id FROM invoices
⇨ WHERE client_id = 4;
```

Причина в том, что список колонок показывает лишь, какие значения возвращать.

С Используя операторы IN и NOT IN, вы можете указать, что значение колонки принадлежит или не принадлежит заданному списку:

```
SELECT * FROM invoices WHERE
⇨ invoice_date IN ('2002-04-24',
⇨ '2002-04-26', '2002-04-28');
```

С Задавать условия можно не только во фразе WHERE, но и во фразе HAVING, хотя я рекомендую сначала освоиться с WHERE. Подробнее конструкция HAVING рассматривается в документации по MySQL.

П В запросах можно выполнять математические вычисления с помощью операторов сложения (+), вычитания (-), умножения (*) и деления (/).

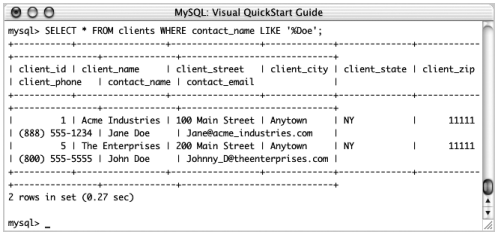

Рис. 4.15. Использование предиката `LIKE` в сочетании с метасимволами позволяет вести поиск по неточному совпадению

Операторы LIKE и NOT LIKE

С использованием в условиях чисел, дат и значений `NULL` все понятно, но вот со строками дело обстоит сложнее. Для сравнения строк на равенство можно написать такой запрос:

```
SELECT * FROM users WHERE user_name =
⇨ 'строка';
```

Однако для менее строгого сравнения требуются дополнительные операторы и специальные символы. Если, например, вы хотите найти человека по фамилии Smith, Smiths или Smithson, то понадобится более гибкий запрос. Вот тут-то и приходят на помощь операторы `LIKE` и `NOT LIKE`. Они применяются по отношению к строкам совместно с двумя метасимволами: знак подчеркивания (_) соответствует любому символу, а знак процента (%) – произвольному числу символов, включая нулевое. В вышеупомянутом примере запрос можно было бы сформулировать так:

```
SELECT * FROM users WHERE last_name
⇨ LIKE 'Smith%';
```

Этот запрос вернет все поля тех записей, в которых поле last_name (фамилия) начинается со строки Smith. Поскольку по умолчанию прописные и строчные буквы при поиске не различаются, то будут найдены также фамилии, начинающиеся на Smith.

Использование операторов LIKE и NOT LIKE

1. Выберите всю информацию о клиентах, для которых полное имя контактного лица заканчивается на Doe (рис. 4.15):

```
SELECT * FROM clients WHERE
⇨ contact_name LIKE '%Doe';
```

Поскольку эта таблица не до конца нормализована (иначе сведения о контактном лице были бы разбиты на два поля: фамилия и имя), то для поиска фамилии Doe приходится воспользоваться поиском по неточному совпадению.

2. Выберите имя клиента и сведения о контактном лице для всех записей, кроме тех, в которых имя контактного лица – John, Joe, Joey и т.д. (рис. 4.16).

```
SELECT client_name, contact_name
⇨ FROM clients WHERE contact_name
⇨ NOT LIKE 'Jo%';
```

При желании исключить некоторые записи я могу воспользоваться предикатом NOT LIKE в сочетании с метасимволами.

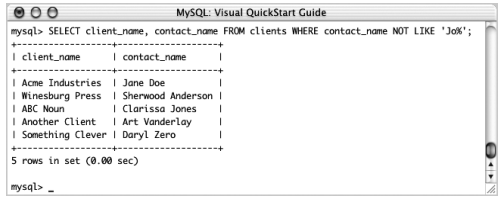

Рис. 4.16. Предикат NOT LIKE позволяет отобрать все записи кроме удовлетворяющих критерию неточного совпадения

С Запросы, включающие предикат LIKE, обычно выполняются очень медленно, поскольку в этом случае не всегда возможно использование имеющихся индексов. Поэтому не злоупотребляйте таким приемом.

П Метасимволы можно употреблять как в начале, так и в конце поисковой строки:

```
SELECT * FROM users WHERE
⇨ user_name LIKE '_Smith%';
```

П Хотя, как правило, предикаты LIKE и NOT LIKE применяются к строкам, при необходимости можно использовать их и для числовых колонок.

С Если вы хотите использовать знак подчеркивания или процента в самой поисковой строке, то его надо экранировать (поставить впереди символ обратной косой черты), чтобы система MySQL не спутала его с метасимволом.

П Символ подчеркивания можно использовать более одного раза. Например, по запросу LIKE '__' будут найдены строки, состоящие из любых двух букв.

Операция соединения

Поскольку реляционные базы зачастую устроены довольно сложно, для извлечения необходимой информации могут понадобиться более изощренные запросы, чем те, что приведены выше. В частности, нередко приходится выполнять операцию соединения (join) – подавать запрос к нескольким взаимосвязанным таблицам.

В SQL рассматривается несколько типов соединений, хотя MySQL реализует не все прописанные в стандарте возможности. Для пользователей с небольшим опытом работы двух основных видов соединений будет достаточно почти во всех случаях. Самый распространенный вид соединения называется *внутренним* (inner или cross join).

```
SELECT * FROM invoices, clients
⇨ WHERE invoices.client_id =
⇨ clients.client_id
```

Это предложение извлекает из таблиц invoices и clients все записи, в которых значения полей invoices.client_id и clients.client_id совпадают. Другими словами, при выполнении запроса внешний ключ client_id в таблице invoices будет полностью заменен информацией о соответствующем клиенте из таблицы clients. При таком внутреннем соединении возвращаются лишь те строки, для которых имеется соответствие (следовательно, записи о клиентах, которым не выставлен ни один счет, не будут выведены).

При выборке из нескольких таблиц необходимо квалифицировать именем таблицы (с помощью нотации *имя_таблицы.имя_колонки*) имена тех колонок, которые встречаются более чем в одной таблице. Это типичная ситуация, поскольку в реляционных базах первичному ключу в одной таблице часто назначают то же имя, что и внешнему ключу в связанной таблице.

Второй из рассматриваемых видов – внешнее левое соединение – отличается от внутреннего тем, что может возвращать также записи, не имеющие соответствия. Синтаксис левого соединения таков:

```
SELECT *
⇨ FROM invoices LEFT JOIN clients
⇨ ON invoices.client_id =
⇨ clients.client_id;
```

Обратите внимание, что там, где во внутреннем соединении стояла запятая, теперь находятся слова LEFT JOIN, а слово WHERE заменено на ON. Для левого соединения очень важно, какая таблица указана первой. В вышеприведенном примере будут возвращены все записи из таблицы clients вместе с информацией из таблицы invoices. Скоро я покажу этот механизм в действии, и тогда многое станет понятно.

Если таблицы слева и справа от оператора LEFT JOIN соединяются по колонкам с одинаковыми именами, то можно проще записать запрос:

```
SELECT *
⇨ FROM invoices LEFT JOIN clients
⇨ ON USING(client_id);
```

Использование соединений

1. Выберите сумму и дату выставления счета, а также имена клиентов для каждого счета (рис. 4.17):

```
   SELECT invoice_amount,
⇨ invoice_date, client_name
   ⇨ FROM invoices, clients
   ⇨ WHERE invoices.client_id =
⇨ clients.client_id;
```

   Этот запрос, по сути дела, заменяет значение поля client_id из таблицы invoices значением поля client_name из соответствующей записи таблицы clients.

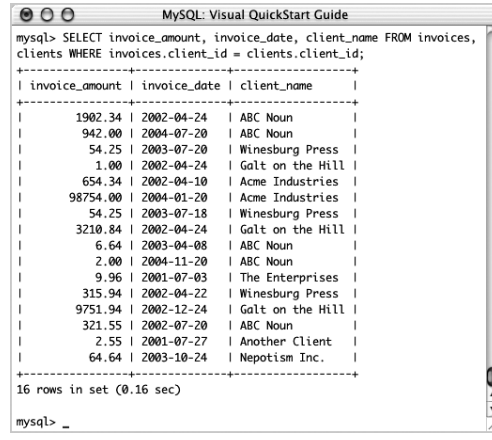

Рис. 4.17. Запрос с внутренним соединением, хотя и весьма многословен, извлекает более полезную информацию из базы данных

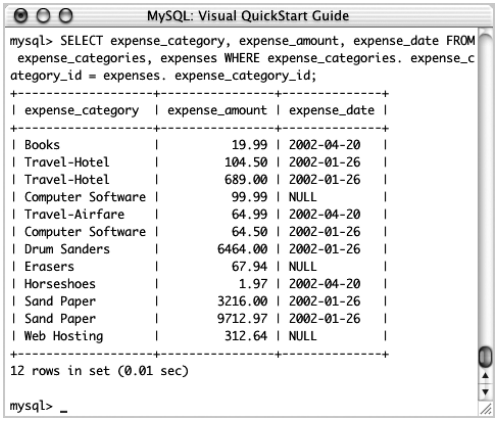

Рис. 4.18. Внутренние соединения полезны, когда есть связь «первичный ключ–внешний ключ», как в случае с таблицами expenses и expense_categories

2. Выберите категорию, дату и сумму каждой затраты (рис. 4.18):

```
SELECT expense_category,
⇨ expense_amount, expense_date
⇨ FROM expense_categories, expenses
⇨ WHERE expense_categories.
⇨ expense_category_id =
⇨ expenses.expense_category_id;
```

На примере данного запроса демонстрируется тот же принцип, что и на этапе 1. Поскольку речь идет о внутреннем соединении, порядок указания таблиц не играет роли (результат будет таким же, если изменить последовательность таблиц на противоположную).

3. Выберите имена всех клиентов и все счета для каждого клиента (рис. 4.19):

```
SELECT client_name, invoice_id,
⇨ invoice_amount, invoice_date,
⇨ invoice_description FROM clients
⇨ LEFT JOIN invoices USING(client_id);
```

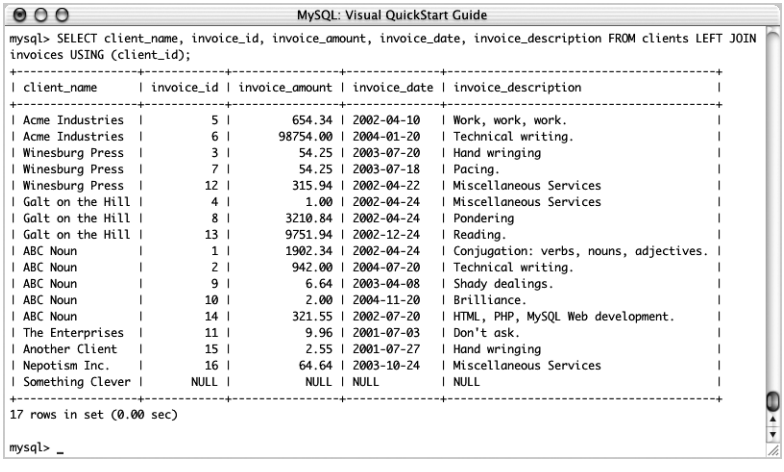

Рис. 4.19. Для левых соединений точная формулировка играет бóльшую роль, чем для внутренних. Наличие соответствия необязательно. Сравните этот результат с рис. 4.20

Этот запрос (с внешним соединением) извлекает имена всех клиентов и с каждым ассоциирует выставленные ему счета. Даже если для клиента нет счетов (как, например, для Something Clever внизу), он все равно выводится. Если бы в этом примере использовалось внутреннее соединение, клиент Something Clever не был бы отобран.

4. Выберите идентификаторы, суммы, даты и описания всех счетов, а также имена соответствующих клиентов (рис. 4.20):

```
SELECT client_name, invoice_id,
⇨ invoice_amount, invoice_date,
⇨ invoice_description FROM invoices
⇨ LEFT JOIN clients USING (client_id);
```

Этот запрос отличается от предыдущего только порядком соединения таблиц. Но результат содержит на одну запись меньше, так как клиенту Something Clever не выставлено ни одного счета.

Разрешается соединять более двух таблиц. Вы можете даже соединить таблицу саму с собой!

Таблицы соединяются по любым колонкам, а не только по тем, что выступают в роли первичных и внешних ключей.

Можно соединять таблицы из разных баз данных (пользуясь синтаксисом *имя_базы_данных.имя_таблицы.имя_колонки*) при условии, что обе находятся на одном и том же сервере (делать это по сети не разрешается).

Соединение, в котором отсутствует условие `WHERE` (например, `SELECT * FROM invoices, clients`) называется *полным*; при этом возвращаются все записи из обеих таблиц. Если таблицы велики, то возвращаемых строк может быть чересчур много.

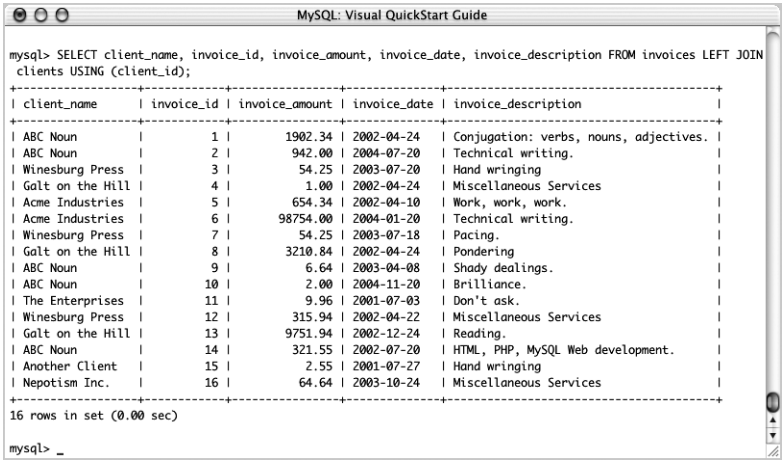

Рис. 4.20. Порядок упоминания таблиц в левом соединении влияет на результат. Вне зависимости от условия отбираются все записи из первой таблицы

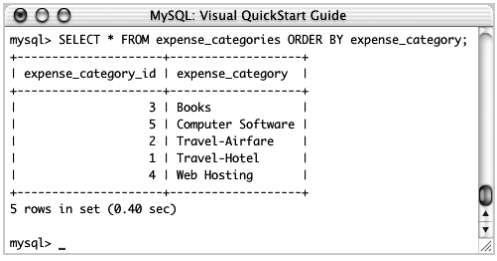

Рис. 4.21. Команда SELECT с фразой ORDER BY возвращает строки, отсортированные по полю expense_category

Сортировка результатов запроса

Если условие WHERE показывает, какие данные нужно отобрать, то фраза ORDER BY определяет способ упорядочения отобранных данных. Вот как используется эта конструкция:

```
SELECT * FROM имя_таблицы
⇨ ORDER BY имя_колонки;

SELECT invoice_amount, invoice_date
⇨ FROM invoices ORDER BY invoice_date;
```

По умолчанию ORDER BY предполагает сортировку в порядке возрастания, то есть числа располагаются от меньших к большим, а даты – от ранних к поздним. Порядок можно изменить на противоположный, добавив слово DESC:

```
SELECT expense_description, expense_date ⇨
FROM expenses
⇨ ORDER BY expense_date DESC;
```

Разрешается сортировать возвращаемые строки по нескольким колонкам, что будет продемонстрировано ниже. Важно понимать, что сортировка ведется лишь по колонкам, включенным в список отбираемых. Если способ сортировки явно не указан, то данные возвращаются в непредсказуемом порядке (обычно в порядке возрастания первичного ключа, хотя и не всегда).

Сортировка данных

1. Расположите все категории затрат в алфавитном порядке (рис. 4.21):

   ```
   SELECT * FROM expense_categories
   ⇨ ORDER BY expense_category;
   ```

Так как в таблице expenses две колонки, вывести данные можно четырьмя способами: сортируя по каждой колонке в порядке возрастания или убывания. Приведенный выше запрос возвращает описания затрат, расположенные по алфавиту (поле expense_category).

2. Упорядочьте счета по клиенту и дате выставления (рис. 4.22):

```
SELECT client_id, invoice_date,
⇨ invoice_amount FROM invoices
⇨ ORDER BY client_id ASC,
⇨ invoice_date DESC;
```

Результат этого запроса отсортирован по полю client_id, а затем найденные группы записей с одинаковым значением отсортированы по полю invoice_date.

П Поскольку MySQL может работать с различными языками, то строковые данные сортируются так, как принято в языке, выбранном при установке (по умолчанию это английский).

П Если в колонке, по которой производится сортировка, есть значения NULL, то соответствующие строки окажутся в начале списка результатов при любом способе упорядочения: как по возрастанию, так и по убыванию.

С Вы можете (и часто будете) использовать команду ORDER BY в сочетании с условием WHERE, соединениями и другими конструкциями. В таком случае фразу ORDER BY следует размещать после всех условий:

```
SELECT invoice_id,
⇨ invoice_amount, invoice_date
⇨ FROM invoices
⇨ WHERE invoice_date >=
⇨ '2002-03-01'
⇨ ORDER BY invoice_date DESC;
```

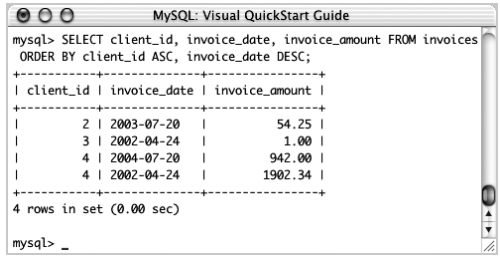

Рис. 4.22. При указании в ORDER BY двух колонок сначала производится сортировка по первой из них, а затем полученный результат еще раз сортируется по второй колонке

Ограничение размера результата

В запросе SELECT может также присутствовать фраза LIMIT. В отличие от условия WHERE, которое определяет, какие строки возвращать, и команды ORDER BY, определяющей способ упорядочения этих строк, ограничение LIMIT показывает, сколько строк вернуть. Используется оно так:

```
SELECT * FROM имя_таблицы LIMIT 10;
```

```
SELECT * FROM имя_таблицы LIMIT 10, 20;
```

В первом примере возвращается не более 10 первых отобранных записей, а во втором – не более 20 записей, начиная с десятой.

Слово LIMIT можно использовать совместно с WHERE и ORDER BY, разместив его в конце запроса:

```
SELECT * FROM invoices
⇨ WHERE invoice_amount > 100.00
⇨ ORDER BY invoice_amount ASC
⇨ LIMIT 10;
```

Ограничение LIMIT не уменьшает нагрузку на сервер базы данных[1] (поскольку он все равно должен отобрать все нужные записи, а только потом урезать результат), но помогает сократить накладные расходы при передаче результата клиенту. Разумеется, не имеет смысла возвращать те колонки или строки, в которых вы не заинтересованы.

[1] Это не совсем так. В документации по MySQL описаны случаи, когда употребление LIMIT все же позволяет ускорить выполнение запроса. – *Прим. переводчика.*

Ограничение числа возвращаемых строк

1. Выберите счет, выставленный раньше всех прочих (рис. 4.23):

```
SELECT * FROM invoices
⇨ ORDER BY invoice_date
⇨ LIMIT 1;
```

   Чтобы вернуть самую раннюю запись, отсортируйте данные по дате в порядке возрастания. А чтобы увидеть только один счет, ограничьте вывод результата с помощью конструкции `LIMIT 1`.

2. Выберите информацию о двух самых крупных затратах на наждачную бумагу (рис. 4.24):

```
SELECT expense_amount,
⇨ expense_description
⇨ WHERE expenses.expense_category_id
⇨ =expense_categories.expense_
⇨ category_id AND expense_category =
⇨ 'Sand paper'
⇨ ORDER BY expense_amount DESC
⇨ LIMIT 2;
```

Запрос кажется довольно громоздким, но неплохо иллюстрирует изученный материал. Сначала я определяю, какие колонки возвращать, затем указываю две таблицы, поскольку мне придется их соединять. Далее задаются условия, определяющие способ соединения таблиц (`expenses.expense_category_id = expense_categories.expense_category_id`) и интересующую меня категорию затрат (`expense_category = 'Sand paper'`). Наконец, я сортирую отобранные записи так, чтобы самые крупные расходы оказались в начале списка, и ограничиваю размер результата двумя строками. Если в вашей базе данных нет категории затрат Sand paper (Наждачная бумага), модифицируйте запрос.

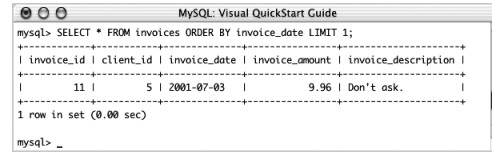

Рис. 4.23. Наличие конструкции `LIMIT 1` гарантирует, что будет возвращено не более одной записи, так что вам не придется просматривать ненужную информацию

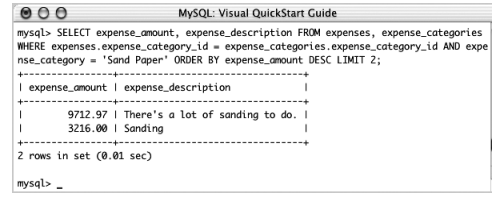

Рис. 4.24. Этот довольно сложный запрос позволяет извлечь из данных, хранящихся в базе accounting, весьма полезную информацию

В главе 9 вы узнаете о применении фразы `LIMIT` для разбиения результата на страницы – обычно такой прием используется в поисковых машинах.

Конструкцию `LIMIT x, y` чаще всего применяют для формирования страниц результата, когда сначала показываются, к примеру, первые 20 записей, затем следующие 20 и т.д.

В главе 5 мы изучим последнюю из применяемых в предложении `SELECT` фраз – `GROUP BY`.

Обновление данных

Коль скоро в таблицах уже есть какие-то данные, у вас появляется возможность тем или иным способом модифицировать их. Чаще всего необходимость в этом возникает при обнаружении неправильных данных или когда информация (скажем, фамилия или электронный адрес клиента) изменяется.

Синтаксис обновления колонок выглядит следующим образом:

```
UPDATE имя_таблицы
⇨ SET имя_колонки=значение;
```

В одном предложении можно изменять сразу несколько колонок:

```
UPDATE имя_таблицы
⇨ SET имя_колонки1='значение1',
⇨ имя_колонки2='значение2' . . .;
```

Обычно для указания подлежащих обновлению строк применяется условие WHERE; в противном случае будут изменены все строки таблицы:

```
UPDATE имя_таблицы
⇨ SET имя_колонки1='значение1'
⇨ WHERE имя_колонки2='значение2';
```

Операция обновления, равно как и удаления, – это одна из важнейших причин использования первичных ключей. Соответствующее поле, которое никогда не должно изменяться, может быть отправной точкой в условиях WHERE, когда все остальные поля могут менять значения.

Обновление записей

Добавьте адрес клиента с именем Galt on the Hill (рис. 4.25):

```
UPDATE clients SET client_street =
⇨ '1000 Tuttle Drive', client_city =
⇨ 'Brazilia', client_state = 'IL',
⇨ client_zip = '60000'
⇨ WHERE client_id = 3;
```

Во время первоначального ввода данных об этом клиенте я не включил туда адресную информацию. С помощью предложения UPDATE можно будет восполнить этот пробел позже.

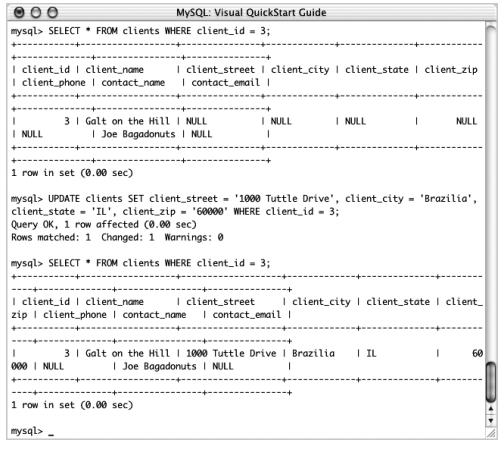

Рис. 4.25. Команда UPDATE позволяет изменять существующие данные в таблицах

С Ни в коем случае не забывайте включать условие WHERE в предложение UPDATE, если не хотите, чтобы изменения затронули каждую строку таблицы.

С Не следует применять команду UPDATE к первичному ключу, поскольку это поле никогда не должно изменяться. В результате изменения первичного ключа может нарушиться целостность связей с какой-то другой таблицей.

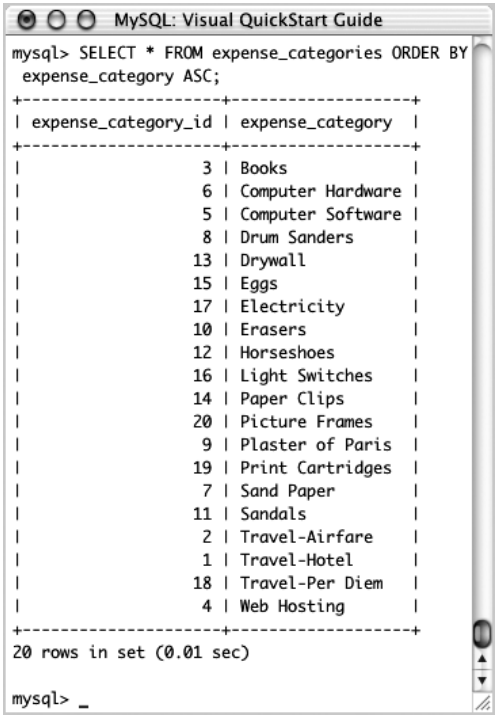

Рис. 4.26. Для удаления всех категорий затрат, связанных с транспортными расходами, сначала расположим имеющиеся категории в алфавитном порядке

Удаление данных

Вы можете удалить ненужные данные из базы – для этого предназначена команда DELETE:

DELETE FROM *имя_таблицы*
⇨ WHERE *имя_колонки=значение*;

Учтите, что удаленную запись уже нельзя восстановить, поэтому перед масштабным удалением имеет смысл сделать резервную копию базы. Кроме того, возьмите за правило включать в предложение удаления условие WHERE, иначе вы вырежете все строки из таблицы. Запрос

DELETE FROM *имя_таблицы*;

полностью очищает таблицу, тем не менее сохраняя ее структуру. Запрос же TRUNCATE FROM *имя_таблицы* сначала удаляет таблицу (как данные, так и структуру), а затем восстанавливает структуру. Конечный результат точно такой же, но последний метод (позаимствованный из СУБД Oracle) быстрее и безопаснее.

При удалении записей возникает также проблема сохранения целостности базы данных. В таблице invoices базы данных accounting, рассматриваемой выше, есть поле client_id. Если удалить запись о клиенте, то могут появиться «фантомные» записи, так как отметки о выставленных ему счетах будут ссылаться на несуществующего клиента (по полю client_id). До тех пор пока в MySQL не будет введена поддержка связей между ключами, эту проблему необходимо решать вручную. В общем, удаляя записи, не забывайте вносить изменения в связанные таблицы.

На следующем примере я объясню, как следовало бы действовать, если бы перед вами стояла задача свести все транспортные затраты в одну категорию.

Удаление данных

1. Просмотрите все существующие категории затрат (рис. 4.26):

```
SELECT * FROM expense_categories
⇨ ORDER BY expense_category ASC;
```

   Чтобы решить, какие записи объединять, в последний раз взгляните на всю таблицу. Следует также записать, какие значения поля expense_category_id соответствуют категориям транспортных затрат. В нашем случае это 1, 2 и 18.

2. Удалите три записи из таблицы (см. рис. 4.27):

```
DELETE FROM expense_categories
⇨ WHERE expense_category_id = 1 OR
⇨ expense_category_id = 2 OR
⇨ expense_category_id = 18;
```

   Для полной уверенности в том, что удаляются именно ненужные фрагменты, я указал значения первичных ключей, хотя мог бы записать запрос и так:

```
DELETE FROM expense_categories
⇨ WHERE expense_category
⇨ LIKE "Travel-%";
```

3. Создайте новую категорию затрат Travel (Поездки):

```
INSERT INTO expense_categories
⇨ VALUES(NULL, 'Travel');
```

4. Получите присвоенное категории Travel значение поля expense_category_id (см. рис. 4.28):

```
SELECT expense_category_id
⇨ FROM expense_categories
⇨ WHERE expense_category = 'Travel';
```

   Отметим, что удаление записей из таблицы оставляет «дыры» в последовательности значений первичного ключа. Хотя значения 1, 2 и 18 теперь не заняты, следующей записи все равно будет присвоен идентификатор 21.

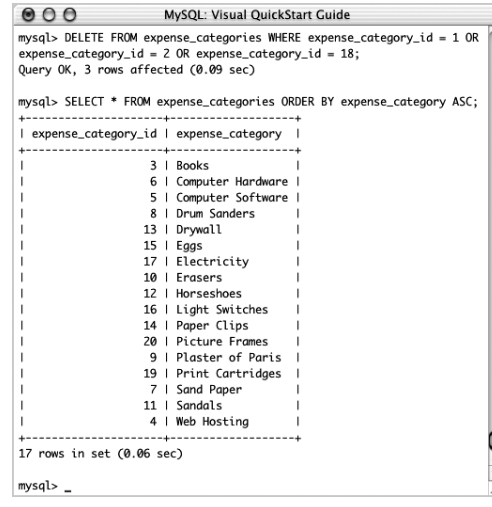

Рис. 4.27. После удаления записей просмотрите таблицу еще раз, чтобы убедиться в том, что изменения внесены правильно (ср. рис. 4.26)

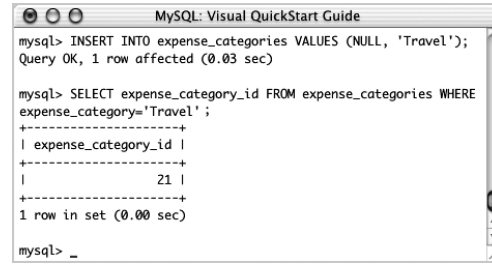

Рис. 4.28. Следующий шаг – вставить новую категорию и получить значение ключа для нее

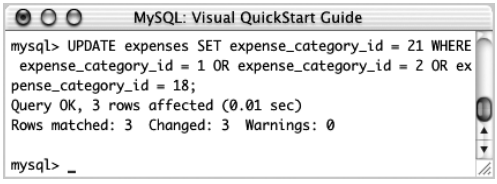

Рис. 4.29. Наконец, следует обновить таблицу expenses с учетом последних изменений

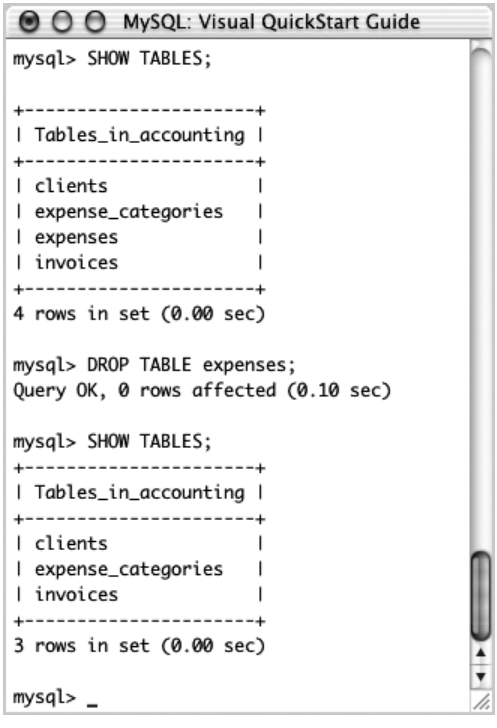

Рис. 4.30. Для удаления всей базы данных или только одной таблицы (как ее содержимого, так и структуры) пользуйтесь командой DROP

5. Обновите таблицу expenses с учетом внесенных изменений (рис. 4.29):

```
UPDATE expenses  SET
⇨ expense_category_id = 21
⇨ WHERE expense_category_id = 1 OR
⇨ expense_category_id = 2 OR
⇨ expense_category_id = 18;
```

Поскольку таблица expense_categories связана с таблицей expenses, нужно отразить в последней изменения, внесенные в первую.

С Для удаления как данных, так и самой таблицы пользуйтесь командой DROP (рис. 4.30):

```
DROP TABLE имя_таблицы;
```

С Для полного удаления базы данных со всеми таблицами и информацией пользуйтесь предложением

```
DROP DATABASE имя_базы_данных;
```

С Не забывайте, что при входе в монитор mysql с указанием параметра `--i-am-a-dummy` запрещено выполнять предложения UPDATE и DELETE без указания условия WHERE.

П Начиная с MySQL версии 4.0 разрешается применять команду DELETE сразу к нескольким таблицам.

Модификация структуры таблиц

Зарезервированное слово ALTER используется для модификации структуры таблиц в базе данных. Обычно под этим понимают добавление, удаление и изменение типа колонок. Но с помощью ALTER можно также переименовать таблицу, назначить другой первичный ключ либо изменить состав индексов. Хотя при добросовестном проектировании структура базы данных должна с самого начала быть правильной, на практике модификации производятся довольно часто. Вот как выглядит базовый синтаксис предложения ALTER:

```
ALTER TABLE имя_таблицы CLAUSE;
```

Поскольку фраз CLAUSE существует много, я решил привести наиболее распространенные в табл. 4.2. Полный перечень вы найдете в приложении 2.

Для демонстрации команды ALTER я модифицирую таблицу clients, разбив поле contact_name на два: contact_first_name и contact_last_name.

Таблица 4.2. Команда ALTER применяется для различных модификаций таблиц

Команда	Применение	Назначение
ADD COLUMN	ALTER TABLE *имя_таблицы* ⇨ ADD COLUMN *имя_колонки* VARCHAR(40)	Добавляет новую колонку в конец таблицы
CHANGE COLUMN	ALTER TABLE *имя_таблицы* ⇨ CHANGE COLUMN *имя_колонки* ⇨ *имя_новой_колонки* VARCHAR(60)	Позволяет изменить тип данных и другие свойства колонки
DROP COLUMN	ALTER TABLE *имя_таблицы* ⇨ DROP COLUMN *имя_колонки*	Удаляет из таблицы колонку со всем ее содержимым
ADD INDEX	ALTER TABLE *имя_таблицы* ⇨ ADD INDEX *имя_индекса*(*имя_колонки*)	Добавляет индекс по колонке с указанным именем
DROP INDEX	ALTER TABLE *имя_таблицы* ⇨ DROP INDEX *имя_индекса*	Удаляет существующий индекс
RENAME AS	ALTER TABLE *имя_таблицы* ⇨ RENAME AS *новое_имя_таблицы*	Изменяет имя таблицы

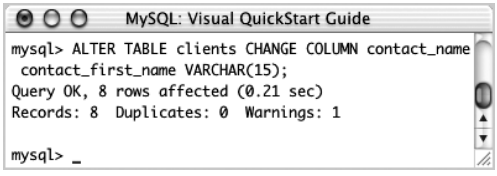

Рис. 4.31. Для переименования или изменения типа колонки применяется команда ALTER TABLE в сочетании с CHANGE COLUMN

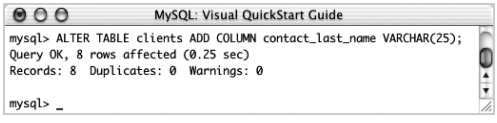

Рис. 4.32. Для добавления новой колонки в таблицу применяется команда ALTER TABLE в сочетании с ADD COLUMN

Изменение структуры таблицы

1. Переименуйте поле contact_name (см. рис. 4.31):

   ```
   ALTER TABLE clients
   ⇨ CHANGE COLUMN contact_name
   ⇨ contact_first_name VARCHAR(15);
   ```

   Эта команда просто изменяет имя и тип данных колонки contact_name. Теперь она называется contact_first_name, и ее длина равна не 40, а 15 символам. Все данные в колонке сохранятся, но будут урезаны до 15 символов.

2. Создайте колонку contact_last_name (рис. 4.32):

   ```
   ALTER TABLE clients
   ⇨ ADD COLUMN contact_last_name
   ⇨ VARCHAR(25);
   ```

   Теперь таблица содержит новую колонку, в которой еще нет значений.

3. Обновите контактную информацию (рис. 4.33):

```
SELECT client_id, contact_first_name
⇨ FROM clients WHERE
⇨ contact_first_name IS NOT NULL;

UPDATE clients SET contact_first_name
⇨ = 'Jane', contact_last_name =
⇨ 'Doe' WHERE client_id = 1;
```

Последний шаг – изменить таблицу с учетом проделанной работы. Для этого я сначала просмотрю информацию на экране, а потом выполню команду UPDATE для каждой строки. Этот процесс довольно утомителен – вот еще одна причина, по которой не стоит модифицировать структуру таблиц после того, как база данных введена в эксплуатацию.

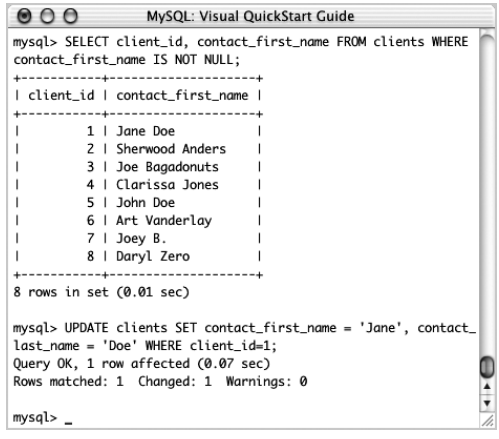

Рис. 4.33. После модификации структуры таблицы нужно обновить информацию, заполнив новую колонку

С Поскольку, используя команду ALTER, можно серьезно запортить базу данных, рекомендую перед применением ALTER всегда делать резервную копию базы.

П Употреблять слово COLUMN в большинстве вариантов команды ALTER необязательно.

С При добавлении новой колонки в таблицу можно включить модификатор AFTER *имя_колонки*, указывающий, после какой колонки должна находиться создаваемая:

```
ALTER TABLE clients ADD COLUMN
⇨ contact_last_name VARCHAR(25)
⇨ AFTER contact_first_name;
```

5

Функции MySQL

В MySQL есть несколько десятков встроенных функций, призванных облегчить решение типичных задач. В этой главе я рассмотрю самые полезные. Если вы уверенно программируете на языках C или C++, то можете даже написать собственную функцию (хотя обсуждение подобной практики выходит за рамки данной книги).

Рассматриваемые здесь функции по большей части применяются в запросах для форматирования и модификации возвращаемых данных. Как и в предыдущей главе, все упражнения предполагают использование монитора mysql. Общие приемы работы иллюстрируются на примере базы данных accounting.

Функции для работы с текстом

Для начала я рассмотрю группу функций, предназначенных для манипулирования текстовыми и символьными данными. Большинство их перечислено в табл. 5.1.

Для использования функции надо применить ее к некоторой колонке, указанной в запросе, например:

```
SELECT FUNCTION(имя_колонки)
⇨ FROM имя_таблицы;
```

Если требуется выбрать несколько колонок, можно записать запрос в таком виде:

```
SELECT *, FUNCTION(имя_колонки)
⇨ FROM имя_таблицы;
```

или

```
SELECT имя_колонки1,
⇨ FUNCTION(имя_колонки2), имя_колонки3 ⇨
FROM имя_таблицы;
```

Таблица 5.1. Это лишь часть функций, применяемых в MySQL для работы с текстовыми колонками

Функция	Применение	Назначение
LENGTH()	LENGTH(*имя_колонки*)	Возвращает длину строки, хранящейся в колонке
LEFT()	LEFT(*имя_колонки*, *x*)	Возвращает *x* левых символов из колонки
RIGHT()	RIGHT(*имя_колонки*, *x*)	Возвращает *x* правых символов из колонки
TRIM()	TRIM(*имя_колонки*)	Исключает лишние пробелы из начала и конца хранимой строки
UPPER()	UPPER(*имя_колонки*)	Переводит символы строки в верхний регистр
LOWER()	LOWER(*имя_колонки*)	Переводит символы строки в нижний регистр
SUBSTRING()	SUBSTRING(*имя_колонки*, *начало*, *число*)	Возвращает *число* символов из колонки от позиции *начало* (позиции нумеруются с 0)

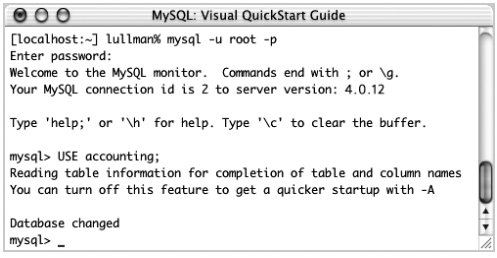

Рис. 5.1. Как и раньше, все примеры будут выполняться в мониторе mysql для базы данных accounting

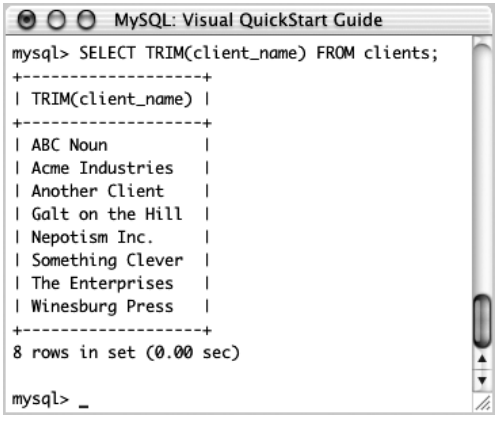

Рис. 5.2. Функция `TRIM()` убирает все лишние символы пропуска из начала и конца строки

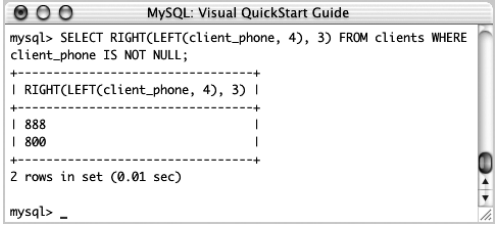

Рис. 5.3. Такое применение функций `RIGHT()` и `LEFT()` позволяет вернуть только наиболее важную часть значения; то же самое можно сделать и с помощью функции `SUBSTRING()`

Хотя регистр символов в именах функций не различается, я все же буду записывать их прописными буквами, чтобы отличать от имен таблиц и колонок. Между именем функции и открывающей скобкой не должно быть пробелов, хотя внутри скобок пробелы допустимы.

Форматирование текста

1. Откройте монитор mysql и укажите рабочую базу данных (рис. 5.1):

   ```
   mysql -u имя_пользователя -p
   USE accounting;
   ```

   Как и в предыдущей главе, в дальнейшем я буду предполагать, что база данных accounting уже выбрана.

2. Удалите лишние пробелы из имен клиентов (рис. 5.2):

   ```
   SELECT TRIM(client_name)
   ⇨ FROM clients;
   ```

   Функция `TRIM()` убирает все символы пропуска (пробелы, знаки табуляции и возврата каретки) из начала и конца строки.

3. Просмотрите коды городов в телефонных номерах клиентов (рис. 5.3):

   ```
   SELECT RIGHT(LEFT(client_phone, 4), 3)
   ⇨ FROM clients WHERE client_phone
   ⇨ IS NOT NULL;
   ```

Запрос выглядит пугающе, но на самом деле я всего лишь последовательно применил две функции. Сначала выбираются все непустые номера клиентов. Поскольку номер телефона хранится в виде *(123)456-7890*, можно быть уверенным, что код города – это символы со второго по четвертый. Чтобы извлечь эту информацию, можно было бы воспользоваться либо функцией `SUBSTRING()`, либо сочетанием функций `LEFT()` и `RIGHT()`. Я выбрал второй способ, чтобы продемонстрировать, как комбинируются функции. Таким образом, сначала будут извлечены первые четыре символа, а затем первый отрезан.

4. Найдите самое длинное название категории затрат (рис. 5.4):

```
SELECT LENGTH(expense_category),
⇨ expense_category
⇨ FROM expense_categories
⇨ ORDER BY LENGTH(expense_category)
⇨ DESC;
```

По этому запросу отбираются все категории затрат и их дли́ны, а затем производится сортировка данных в порядке убывания длины.

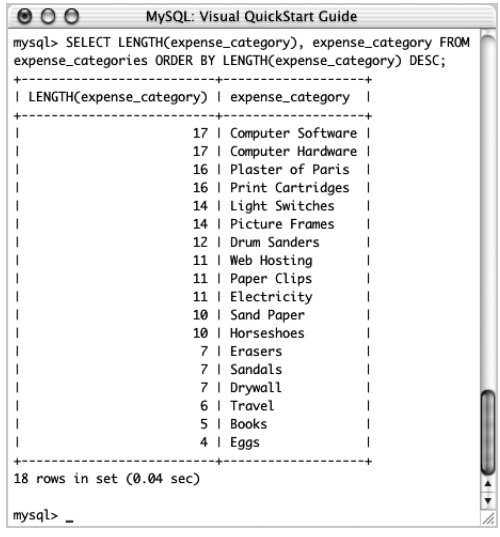

Рис. 5.4. Функцию можно использовать даже во фразе `ORDER BY`

П Запрос, подобный приведенному в п. 4 (см. также рис. 5.4), может быть полезен для уточнения необходимой длины колонки после того, как база данных заполнена.

С Бо́льшую часть функций MySQL можно использовать не только в запросах `SELECT`, но и, например, для форматирования вставляемых данных в предложениях `INSERT`.

С Функции можно применять не только к строкам, хранящимся в таблице. Например, совершенно корректна такая запись:

```
SELECT UPPER('makemebig');
```

Конкатенация и псевдонимы

Функция CONCAT() – одна из самых полезных для работы с текстом – заслуживает того, чтобы отвести ей целый раздел. Заодно мы обсудим часто используемую в SQL концепцию псевдонимов. Функция CONCAT() выполняет операцию конкатенации; так в программировании называется объединение строк. Внутри скобок вы указываете различные значения, подлежащие конкатенации, разделяя их запятыми:

```
CONCAT(имя_колонки1, имя_колонки2)
```

Хотя чаще всего функция CONCAT() применяется к колонкам, она позволяет объединять и строки, заключенные в одинарные кавычки. Вот как можно получить полное имя человека из фамилии и имени:

```
CONCAT(last_name, ', ', first_name)
```

Поскольку при конкатенации создается новое значение, нужен какой-то способ сослаться на него. Именно здесь и приходят на помощь псевдонимы. *Псевдоним* (alias) – это всего лишь дополнительное символическое имя, которому предшествует зарезервированное слово AS:

```
SELECT CONCAT(last_name, ', ',
⇨ first_name) AS name FROM users;
```

В результате из таблицы users будут отобраны имена и фамилии всех пользователей и представлены в виде полного имени, которое будет называться name. Именно этот псевдоним используется в программах для ссылки на возвращенные значения. Общий синтаксис

```
SELECT имя_колонки AS псевдоним
⇨ FROM имя_таблицы;
```

можно применять к любым отбираемым колонкам, даже если они не модифицировались с помощью функций.

Использование конкатенации и псевдонимов

1. Выведите всю информацию об адресе клиента в одном значении (рис. 5.5):

```
SELECT client_name, CONCAT
⇨ (client_street, ', ', client_city,
⇨ ', ', client_state, ' ', client_zip)
⇨ AS address FROM clients;
```

   Таким образом `CONCAT()` собирает всю адресную информацию в одно аккуратно отформатированное поле, названное `address`.

2. Выберите все затраты вместе с описанием и названием категории (рис. 5.6):

```
SELECT expense_amount, expense_date,
⇨ CONCAT(expense_category, ': ',
⇨ expense_description) FROM
⇨ expenses, expense_categories
⇨ WHERE expenses.expense_
⇨ category_id = expense_categories.
⇨  expense_category_id;
```

   В этом запросе я объединил данные двух таблиц, чтобы просматривать информацию как о затратах, так и об их категориях. Конкатенируются две колонки из разных таблиц.

3. Покажите десять счетов на наибольшие суммы, а также имена и идентификаторы соответствующих клиентов (рис. 5.7):

```
SELECT invoices.*, CONCAT
⇨ (client_name, ' - ',
⇨ clients.client_id)
⇨ AS client_info FROM invoices
⇨ LEFT JOIN clients USING(client_id)
⇨ ORDER BY invoice_amount DESC
⇨ LIMIT 10;
```

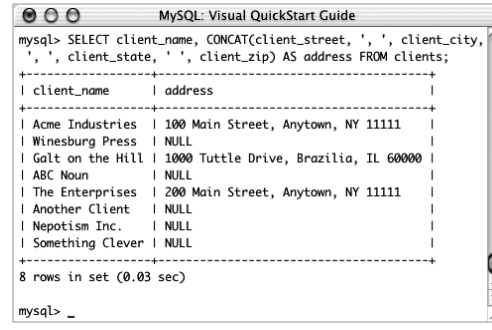

Рис. 5.5. Функция `CONCAT()` очень полезна для форматирования результатов запроса

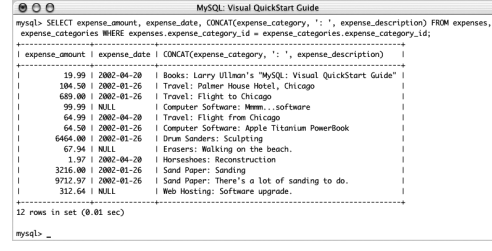

Рис. 5.6. Функции можно применять по-разному, в том числе и для полей из нескольких таблиц

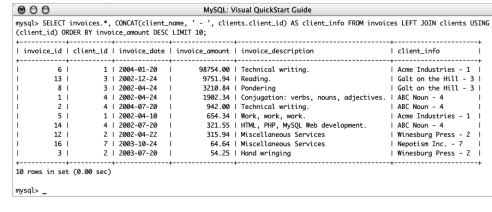

Рис. 5.7. Функцию `CONCAT()` и псевдонимы можно применять в любом запросе, в частности включающем соединения

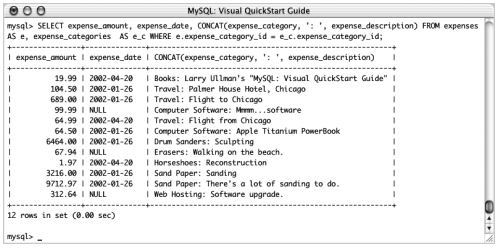

Рис. 5.8. Я упростил запрос, применив псевдонимы для имен таблиц. На конечном результате это не отразилось

В этом запросе я выполнил левое соединение, отсортировал результат по колонке `invoice_amount` и ограничил результат десятью записями. Функция `CONCAT()` применена к имени и идентификатору клиента.

4. Упростите запрос 2, воспользовавшись псевдонимами для имен таблиц (рис. 5.8):

```
SELECT expense_amount, expense_date,
⇨ CONCAT(expense_category, ': ',
⇨ expense_description) FROM
⇨ expenses AS e, expense_
⇨ categories AS e_c WHERE
⇨ e.expense_category_id =
⇨ e_c.expense_category_id;
```

Этот запрос эквивалентен предыдущему, но за счет псевдонимов имен таблиц он стал проще.

П Длина псевдонима не должна превышать 255 символов, прописные и строчные буквы в псевдонимах различаются.

С Слово `AS`, предшествующее псевдониму, можно опускать, то есть допустима такая запись:

```
SELECT имя_колонки псевдоним
⇨ FROM имя_таблицы;
```

П Кроме функции `CONCAT()` есть еще функция `CONCAT_WS()`, где `WS` означает «with separators» (с разделителями). Она вызывается так: `CONCAT_WS(`*разделитель*`, `*имя_колонки1*`, `*имя_колонки2*`, ...)`. В результате между значениями перечисленных колонок вставляется указанный разделитель.

Функции для работы с числами

Помимо стандартных математических действий – сложения, вычитания, умножения и деления – в MySQL есть еще десятка два функций, предназначенных для форматирования числовых данных и манипулирования ими (наиболее употребительные перечислены в табл. 5.2).

Особо хочу отметить функции `FORMAT()`, `ROUND()` и `RAND()`. Первая (строго говоря, она применима не только к числам) представляет любое число в традиционно принятом формате[1]. Например, если в базе указана цена автомобиля 20198.20, то результатом вызова `FORMAT(car_cost, 2)` будет 20,198.20.

Таблица 5.2. Наиболее употребительные функции для работы с числами, не считая тригонометрических и логарифмических

Функция	Применение	Назначение
`ABS()`	`ABS(`*имя_колонки*`)`	Возвращает абсолютную величину колонки
`CEILING()`	`CEILING(`*имя_колонки*`)`	Возвращает наименьшее целое число, большее либо равное значению в колонке
`FLOOR()`	`FLOOR(`*имя_колонки*`)`	Возвращает наибольшее целое число, меньшее либо равное значению в колонке
`FORMAT()`	`FORMAT(`*имя_колонки*`, `*y*`)`	Возвращает значение в колонке, представленное числом с *y* десятичных знаков, в котором группы из трех цифр разделены запятыми
`MOD()`	`MOD(`*x*`, `*y*`)`	Возвращает остаток от деления *x* на *y* (один или оба аргумента могут быть колонками)
`RAND()`	`RAND()`	Возвращает случайное число между 0 и 1,0
`ROUND()`	`ROUND(`*x*`, `*y*`)`	Возвращает число *x*, округленное до *y* десятичных знаков
`SIGN()`	`SIGN(`*имя_колонки*`)`	Возвращает индикатор знака аргумента: -1, если аргумент отрицателен, 0 – если равен нулю и 1 – если положителен
`SQRT()`	`SQRT(`*имя_колонки*`)`	Вычисляет квадратный корень из значения колонки

[1] Этот формат зависит от региональных установок компьютера, на котором выполняется сама СУБД. – *Прим. ред.*

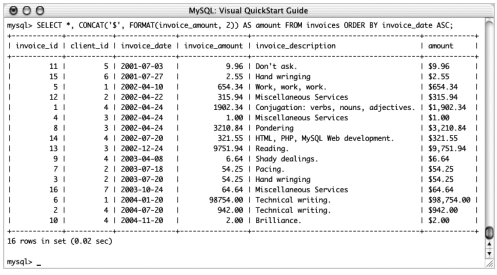

Рис. 5.9. Применив две функции и надлежащее форматирование, можно вывести денежные суммы в привычном виде

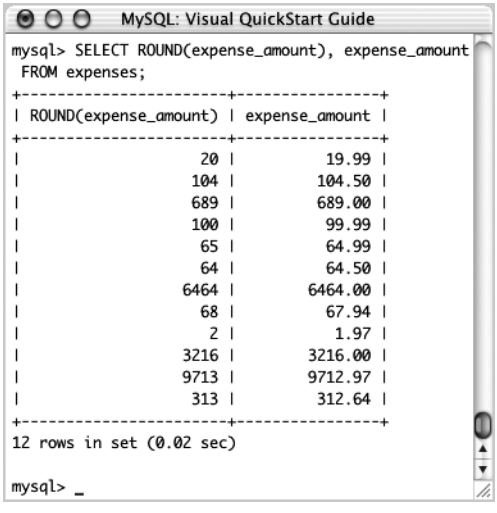

Рис. 5.10. Функцию `ROUND()` удобно использовать, если вас интересует только ограниченное число знаков после запятой

Функция `ROUND()` принимает один аргумент, чаще всего значение колонки, и округляет его до указанного числа десятичных знаков. Если второй аргумент (число знаков) не задан, то округление производится до ближайшего целого. Если указано больше десятичных знаков, чем имеется в самом числе, то оставшиеся позиции заполняются нулями справа от десятичной точки.

Функция `RAND()`, как следует из ее названия (от англ. random – «произвольный») возвращает случайное число из диапазона от 0 до 1.0:

```
SELECT RAND();
```

Эта функция может также использоваться для сортировки отобранных данных в произвольном порядке:

```
SELECT * FROM имя_таблицы
⇨ ORDER BY RAND();
```

Использование функций для работы с числами

1. Выведите счета, отсортированные по дате, форматируя денежные суммы в валюте США (рис. 5.9):

   ```
   SELECT *, CONCAT('$', FORMAT
   ⇨ (invoice_amount, 2)) AS amount
   ⇨ FROM invoices ORDER BY
   ⇨ invoice_date ASC;
   ```

   С помощью описанной выше функции `FORMAT()` в сочетании с `CONCAT()` можно представить любое число как денежную сумму (например, для вывода на Web-страницу).

2. Округлите суммы всех затрат до доллара (рис. 5.10):

   ```
   SELECT ROUND(expense_amount),
   ⇨ expense_amount FROM expenses;
   ```

Если функции ROUND() не передан второй аргумент, то она округляет до ближайшего целого.

3. Дважды отберите имена всех клиентов, отсортировав их в случайном порядке (рис. 5.11, 5.12):

```
SELECT client_id, client_name
⇨ FROM clients ORDER BY RAND();
```

Хотя применение конструкции ORDER BY RAND() в такой ситуации вряд ли пригодится на практике, но идею вы уловили. Функция RAND() формирует не идеальную случайную последовательность, но для практических целей ее в большинстве случаев достаточно. Обратите внимание, что функция RAND() не имеет аргументов.

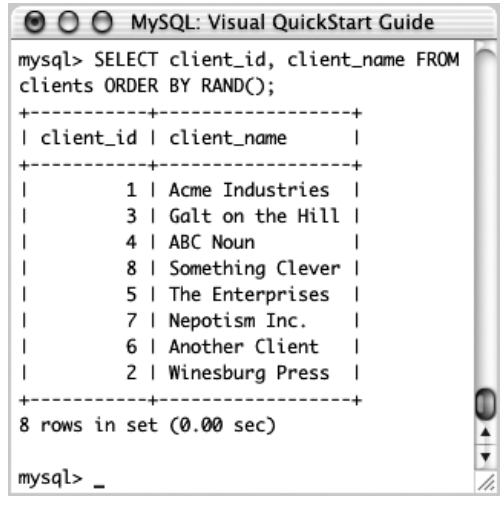

Рис. 5.11. Результат первого выполнения запроса со случайной сортировкой

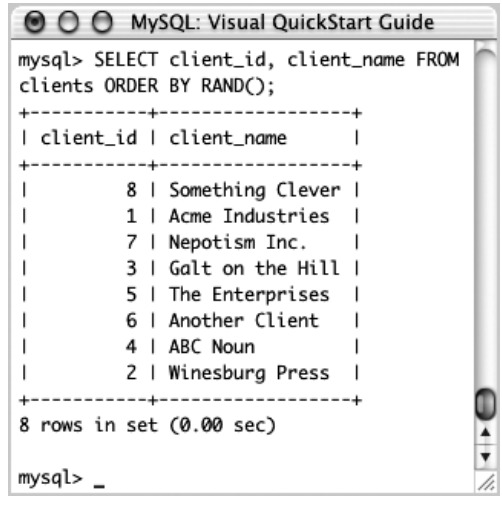

Рис. 5.12. При втором выполнении того же запроса результаты упорядочены по-другому

Помимо перечисленных выше в MySQL есть еще тригонометрические, логарифмические и другие функции.

Функция MOD() дает тот же результат, что оператор '%':

```
SELECT MOD(9,2);
SELECT 9 % 2;
```

Удачный пример использования конструкции ORDER BY RAND() – произвольный вывод одного баннера из группы на Web-страницу.

Еще раз напомню, что функции можно применять не только к колонкам. Так, абсолютно корректны следующие запросы:

```
SELECT ROUND(34.089, 1);
SELECT SQRT(81);
SELECT ABS(-8);
```

Функции для работы с датой и временем

В MySQL реализованы очень гибкие средства для работы с типами данных DATE и TIME. Правда, большинство пользователей из-за недостатка знаний использует их не в полной мере. Например, MySQL позволяет выполнять вычисления с датами или возвратить только название месяца. Многие из имеющихся функций приведены в табл. 5.3.

Таблица 5.3. В MySQL много функций для работы с датой и временем

Функция	Применение	Назначение
HOUR()	HOUR(*имя_колонки*)	Возвращает только час
MINUTE()	MINUTE(*имя_колонки*)	Возвращает только минуту
SECOND()	SECONS(*имя_колонки*)	Возвращает только секунду
DAYNAME()	DAYNAME(*имя_колонки*)	Возвращает название дня
DAYOFMONTH()	DAYOFMONTH (*имя_колонки*)	Возвращает номер дня в месяце
MONTHNAME()	MONTHNAME (*имя_колонки*)	Возвращает название месяца
MONTH()	MONTH(*имя_колонки*)	Возвращает номер месяца
YEAR()	YEAR(*имя_колонки*)	Возвращает год
ADDDATE()	ADDDATE(*имя_колонки*, ⇨ INTERVAL *x тип*)	Возвращает дату, полученную прибавлением *x* единиц к значению колонки (см. врезку «Функции ADDDATE() и SUBDATE()»)
SUBDATE()	SUBDATE(*имя_колонки*, ⇨ INTERVAL *x тип*)	Возвращает дату, полученную вычитанием *x* единиц из значения колонки (см. врезку «Функции ADDDATE() и SUBDATE()»)
CURDATE()	CURDATE()	Возвращает текущую дату
CURTIME()	CURTIME()	Возвращает текущее время
NOW()	NOW()	Возвращает текущие дату и время
UNIX_TIMESTAMP()	UNIX_TIMESTAMP (*дата*)	Возвращает число секунд, прошедших с 1 января 1970 года или от указанной даты

Итак, возможностей работы с датой и временем довольно много, и их лучше всего проиллюстрировать на примерах.

Использование функций для работы с датой и временем

1. Выведите все счета, выставленные в апреле (рис. 5.13):

   ```
   SELECT * FROM invoices
   ⇨ WHERE MONTH(invoice_date) = 4;
   ```

   По данному запросу будут выведены все счета, выставленные в апреле – четвертом месяце года. То же самое можно было бы записать иначе:

   ```
   WHERE MONTHNAME = 'April'
   ```

   (хотя всегда лучше употреблять в качестве критерия поиска чи́сла, а не строки).

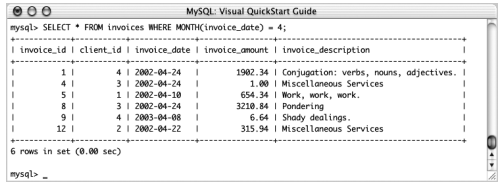

Рис. 5.13. С помощью функции `MONTH()` можно выполнить поиск по колонке, содержащей дату

Функции ADDDATE() и SUBDATE()

Функции `ADDDATE()` и `SUBDATE()`, равно как и их синонимы `DATE_ADD()` и `DATE_SUB()`, выполняют вычисления с датами. Порядок их вызова таков:

`ADDDATE(`*дата*`, INTERVAL` *x тип*`)`

Здесь *дата* может быть как литералом, так и значением колонки. Значение *x* интерпретируется с учетом типа. Возможны следующие типы: `SECOND`, `MINUTE`, `HOUR`, `DAY`, `MONTH` и `YEAR`, а также их комбинации: `MINUTE_SECOND`, `HOUR_MINUTE`, `DAY_HOUR` и `YEAR_MONTH`.

Чтобы прибавить к дате два часа, нужно написать:

```
ADDDATE(дата, INTERVAL 2 HOUR)
```

Чтобы прибавить две недели к дате 2 декабря 2002 года, пишем

```
ADDDATE('2002-12-31', INTERVAL 14 DAY)
```

Чтобы вычесть 15 месяцев из даты, пишем

```
SUBDATE(date, INTERVAL '1-3' YEAR_MONTH)
```

В последнем запросе мы просим MySQL вычесть один год и три месяца из значения, хранящегося в колонке date.

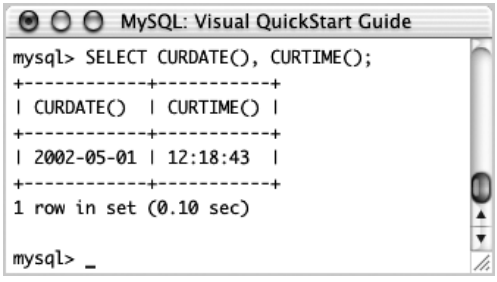

Рис. 5.14. Функции `CURDATE()` и `CURTIME()` возвращают соответственно текущие дату и время. Функция `NOW()` возвращает одно значение, содержащее как дату, так и время

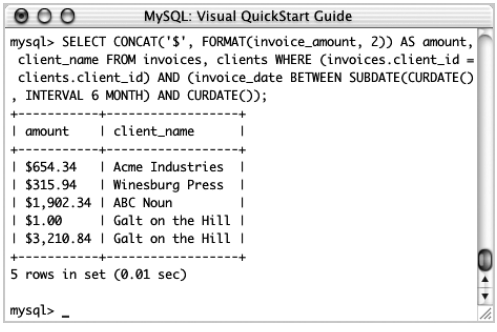

Рис. 5.15. Здесь функция `CURDATE()` используется для указания диапазона дат

2. Покажите текущие дату и время (рис. 5.14):

```
SELECT CURDATE(), CURTIME();
```

Чтобы получить информацию о текущих дате и времени от сервера MySQL, следует воспользоваться функциями `CURDATE()` и `CURTIME()`. Перед вами пример запроса, в котором не нужно указывать имя таблицы.

3. Выведите суммы счетов за последние шесть месяцев и имена клиентов, которым выставлены эти счета (рис. 5.15):

```
SELECT CONCAT('$', FORMAT
⇨ (invoice_amount, 2)) AS amount,
⇨ client_name FROM invoices, clients
⇨ WHERE (invoices.client_id =
⇨ clients.client_id) AND
⇨ (invoice_date BETWEEN
⇨ SUBDATE(CURDATE(), INTERVAL
⇨ 6 MONTH) AND CURDATE());
```

В этом запросе используется несколько уже встречавшихся приемов. Во-первых, соединены две таблицы, чтобы можно было вместе с суммой счета вывести имя клиента. Во-вторых, сумма счета преобразована в более привычный формат. В-третьих, в условии `WHERE` есть две части: одна для указания способа соединения таблиц, а другая – для выборки счетов, выставленных за последние шесть месяцев. Слово `BETWEEN` (между) служит для указания диапазона начиная с даты, отстоящей от текущей на шесть месяцев – `SUBDATE(CURDATE(), INTERVAL 6 MONTH)`, и кончая текущей датой – `CURDATE()`. Тем самым мы отсекаем авансовые счета (с датой выставления в будущем).

Формат даты и времени

Есть еще две функции, относящиеся к дате и времени; возможно, ими-то вы и будете чаще всего пользоваться. Это DATE_FORMAT() и TIME_FORMAT(). Они в какой-то мере перекрываются: DATE_FORMAT() может форматировать одновременно дату и время, а вот TIME_FORMAT() – только время. Порядок вызова таков:

```
SELECT DATE_FORMAT(имя_колонки,
⇨ 'форматирование') AS псевдоним
⇨ FROM имя_таблицы;
```

Параметр *форматирование* может включать специальные символы, предваряемые знаком процента, вместо которых подставляются те или иные компоненты даты. В табл. 5.4 перечислены чаще всего употребляемые символы форматирования. Их можно использовать в любом сочетании вместе с произвольными текстовыми вкраплениями, например знаками препинания.

Предположим, в колонке the_date хранится значение 2002-04-30 23:07:45 (30-е апреля 2002 года, 23 часа 7 минут 45 секунд). Его можно отформатировать следующими способами:

- как время – 11:07:45 PM (post meridiem – пополудни):

  ```
  DATE_FORMAT(the_date, '%r')
  ```

- как время без секунд – 11:07 PM:

  ```
  DATE_FORMAT(the_date, '%l:%i:%p')
  ```

- как дату – April 30th, 2002:

  ```
  DATE_FORMAT(the_date, '%M %D, %Y')
  ```

Таблица 5.4. Символы форматирования, применяемые в функциях DATE_FORMAT() и TIME_FORMAT()

Символ	Значение	Пример
%e	Число дня месяца	1-31
%d	Число дня месяца с двумя цифрами	01-31
%D	Число дня с суффиксом	1st-31st
%W	День недели	Sunday-Saturday (воскресенье-суббота)
%a	Сокращенное название дня недели	Sun-Sat
%c	Число месяца	1-12
%m	Число месяца с двумя цифрами	01-12
%M	Название месяца	January-December (январь-декабрь)
%b	Сокращенное название месяца	Jan-Dec
%Y	Год	2002
%y	Последние две цифры года	02
%l	Час	1-12
%h	Час с двумя цифрами	01-12
%k	Час в 24-часовом измерении	0-23
%H	Час в 24-часовом измерении, с двумя цифрами	00-23
%i	Минуты	00-59
%S	Секунды	00-59
%r	Время	8:17:02 PM
%T	Время в 24-часовом измерении	20:17:02
%p	Сокращение AM (ante meridiem, до полудня) или PM (post meridiem, пополудни)	AM или PM

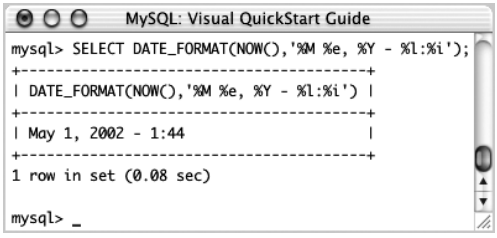

Рис. 5.16. Использование нескольких символов форматирования в функции `DATE_FORMAT()`

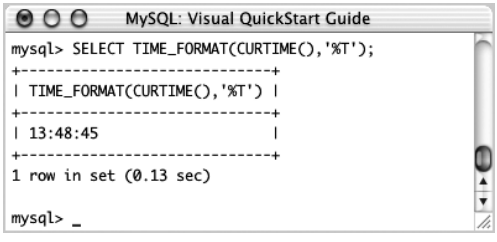

Рис. 5.17. При форматировании времени без даты надо пользоваться функцией `TIME_FORMAT()`

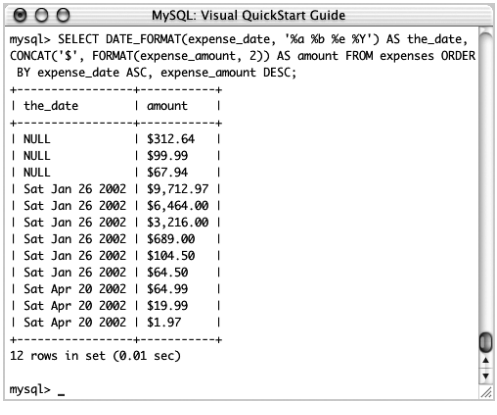

Рис. 5.18. Функция `DATE_FORMAT()` не оказывает влияния на `NULL`

Форматирование даты и времени

1. Выведите текущие дату и время в формате *Месяц ДД, ГГГГ – ЧЧ:ММ* (рис. 5.16):

   ```
   SELECT DATE_FORMAT(NOW(),
   ⇨ '%M %e, %Y - %l:%i');
   ```

   С помощью функции `NOW()`, которая возвращает текущие дату и время, я могу поэкспериментировать с различными способами форматирования.

2. Выведите текущее время в 24-часовом измерении (рис. 5.17):

   ```
   SELECT TIME_FORMAT(CURTIME(), '%T');
   ```

   Функцию `DATE_FORMAT()` можно применять для одновременного форматирования даты и времени (или только даты), но если нужно отформатировать лишь время, то следует обратиться к функции `TIME_FORMAT()`. Ее можно применять как к значению, содержащему только время (например, `CURTIME()`), так и к значению, содержащему дату и время (например, `NOW()`).

3. Выберите все затраты, отсортировав их по дате и сумме и представив дату в формате *День недели (сокращенное название) День месяца (сокращенное название) Год* (рис. 5.18):

   ```
   SELECT DATE_FORMAT(expense_date,
   ⇨ '%a %b %e %Y') AS the_date,
   ⇨ CONCAT('$', FORMAT(expense_
   ⇨ amount, 2)) AS amount FROM expenses
   ⇨ ORDER BY expense_date ASC,
   ⇨ expense_amount DESC;
   ```

Это лишь один пример того, как с помощью функций форматирования можно изменить внешний вид результатов выполнения запроса.

Функции шифрования

В MySQL есть несколько встроенных функций шифрования и одна функция дешифрования. Вы уже встречались с функцией PASSWORD(), применяемой для шифрования паролей пользователя:

```
INSERT INTO users(user_id,
⇨ username, user_pass())
⇨ VALUES(NULL, 'строка',
⇨ PASSWORD('пароль'));
```

Функция ENCRYPT() аналогична PASSWORD() – она тоже возвращает зашифрованную строку, но принимает дополнительный аргумент для параметризации алгоритма шифрования:

```
INSERT INTO users(user_id, username,
⇨ user_pass())
⇨ VALUES(NULL, 'строка',
⇨ ENCRYPT('пароль', 'аргумент'));
```

Функция ENCRYPT() пользуется функцией crypt() из стандартной библиотеки UNIX, поэтому на других платформах может отсутствовать. В MySQL есть также функция DES_ENCRYPT(), применяемая на SSL-соединениях.

Функции PASSWORD() и ENCRYPT() возвращают зашифрованную строку, не поддающуюся расшифровке. Это существенно повышает безопасность, поскольку зашифрованные с помощью PASSWORD() и ENCRYPT() пароли не могут быть восстановлены в исходном виде. Если необходимо шифровать информацию так, чтобы позже ее можно было дешифровать, пользуйтесь функциями ENCODE() и DECODE(). Они также могут принимать дополнительный аргумент для параметризации шифрования:

```
INSERT INTO users(user_id, username,
⇨ user_ssn())
```

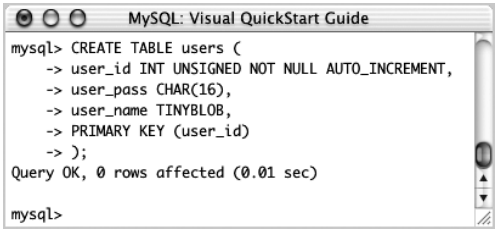

Рис. 5.19. Таблица users, необязательно являющаяся частью базы данных accounting, используется для демонстрации шифрования и дешифрования

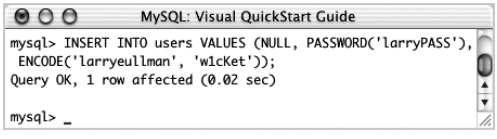

Рис. 5.20. При сохранении данные шифруются

```
VALUES(NULL, 'строка',
⇨ ENCODE('123-45-6789', 'аргумент'));
SELECT имя_пользователя,
⇨ DECODE(user_ssn, 'аргумент')
FROM users WHERE user_id=8;
```

Шифрование и дешифрование данных

1. Создайте новую таблицу users (рис. 5.19):

   ```
   CREATE TABLE users (
   user_id INT UNSIGNED NOT NULL
   AUTO_INCREMENT,
   user_pass CHAR(16),
   user_name TINYBLOB,
   PRIMARY KEY (user_id)
   );
   ```

   Поскольку ни в одной из существующих таблиц ничего шифровать не нужно, я решил создать новую таблицу users. Она содержит три поля: user_id, user_pass и user_name. Во втором из них будет храниться пароль пользователя, зашифрованный функцией PASSWORD(). Поскольку эта функция всегда возвращает строку длиной 16 символов, тип поля – CHAR(16). Поле user_name будет зашифровано с помощью функции ENCODE(), так что в дальнейшем его можно расшифровать. Функция ENCODE() возвращает двоичное значение, которое необходимо сохранять в колонке типа BLOB.

2. Вставьте новую запись о пользователе (рис. 5.20):

   ```
   INSERT INTO users
   VALUES(NULL, PASSWORD('larryPASS'),
   ENCODE('larryeullman', 'w1cKet'));
   ```

Я добавил в таблицу новую запись, воспользовавшись функцией PASSWORD() для шифрования пароля (larryPASS) и функцией ENCODE() с дополнительным параметром w1cKet для шифрования имени пользователя (larryeullman).

3. Выберите информацию о пользователе (рис. 5.21):

```
SELECT user_id FROM users
⇨ WHERE (user_pass =
⇨ PASSWORD('larryPASS'))
⇨ AND (DECODE(user_name, 'w1cKet') =
⇨ 'larryeullman');
```

Если некоторое значение было сохранено с помощью функции PASSWORD(), то для сравнения его с другим значением понадобится применить PASSWORD() еще раз. Так, во время регистрации посетителя вы шифруете введенный им пароль и при следующей попытке входа в систему сравниваете результат шифрования только что введенного пароля со значением, хранимым в базе. Если значения совпали, пользователь ввел правильный пароль, иначе предложение SELECT возвращает пустой результат (рис. 5.22).

Значение, сохраненное функцией ENCODE(), можно дешифровать функцией DECODE() при условии, что в обоих случаях указан один и тот же аргумент для параметризации шифрования.

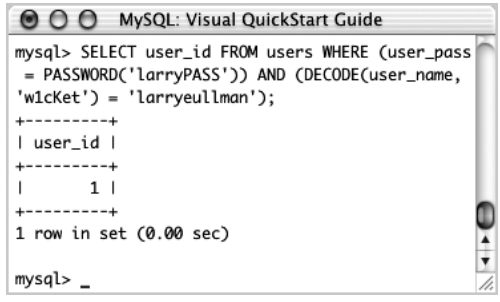

Рис. 5.21. При выборке данные дешифруются

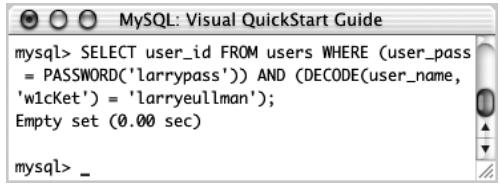

Рис. 5.22. Функция PASSWORD() предполагает различение символов в разном регистре, так что при вводе пароля строчными буквами вместо прописных результат выборки будет пустым

С При использовании функций ENCRYPT(), ENCODE() и DECODE() в программах храните параметризующий шифрование аргумент в безопасном месте.

П Раздел справочного руководства, посвященный функциям шифрования, находится на Web-странице www.mysql.com/doc/M/i/Miscellaneous_functions.html.

С Можно рекомендовать применение функции PASSWORD() для шифрования информации, которую никогда не придется просматривать (например, паролей и, возможно, имен пользователей). Функцию ENCODE() стоит использовать для хранения информации, требующей защиты и вместе с тем просмотра в исходном виде: номеров кредитных карточек, номеров социального страхования, возможно адресов и т.п.

Агрегатные функции

Для группировки результатов запроса применяется фраза GROUP BY. Ее назначение – сгруппировать возвращенные строки в блоки. Например, чтобы сгруппировать все счета по идентификатору клиента, нужно отправить серверу такой запрос:

```
SELECT * FROM invoices
⇨ GROUP BY client_id;
```

Возвращенные данные представляют собой агрегированную информацию, а не просто отдельные записи. Поэтому если раньше вы получали семь накладных для одного клиента, то применение GROUP BY вернет их как одну запись. Я не обсуждал эту концепцию раньше, потому что обычно GROUP BY используется в сочетании с одной из нескольких агрегатных функций, перечисленных в табл. 5.5.

Таблица 5.5. Агрегатные функции обычно используются в сочетании с фразой GROUP BY

Функция	Применение	Назначение
MIN()	MIN(*имя_колонки*)	Возвращает наименьшее значение в колонке
MAX()	MAX(*имя_колонки*)	Возвращает наибольшее значение в колонке
SUM()	SUM(*имя_колонки*)	Возвращает сумму всех значений в колонке
COUNT()	COUNT(*имя_колонки*)	Подсчитывает число строк

Вместе с предложением GROUP BY команды SELECT можно применять фразы WHERE, ORDER BY и LIMIT. Общая структура запроса выглядит так:

```
SELECT имена_колонок FROM имя_таблицы
⇨ WHERE условие GROUP BY имя_колонки
⇨ ORDER BY имя_колонки LIMIT x;
```

Группировка данных

1. Найдите счет с наибольшей суммой (рис. 5.23):

```
SELECT MAX(invoice_amount)
⇨ FROM invoices;
```

Поскольку функция MAX() возвращает наибольшее значение в колонке, это простейший способ достичь нужного результата. А можно было бы поступить и так:

```
SELECT invoice_amount
⇨ FROM invoices
⇨ ORDER BY invoice_amount DESC
⇨ LIMIT 1.
```

2. Определите суммарные затраты по каждой категории (рис. 5.24):

```
SELECT SUM(expense_amount),
⇨ expense_category
⇨ FROM expenses LEFT
⇨ JOIN expense_categories
⇨ USING (expense_category_id)
⇨ GROUP BY expense_categories.
⇨ expense_category_id;
```

Для решения задачи сначала соедините две таблицы, чтобы включить в результат название категории затрат. Затем все затраты группируются по идентификатору категории, и величины в каждой группе суммируются. MySQL возвращает таблицу из двух колонок, где записаны общая сумма и название категории.

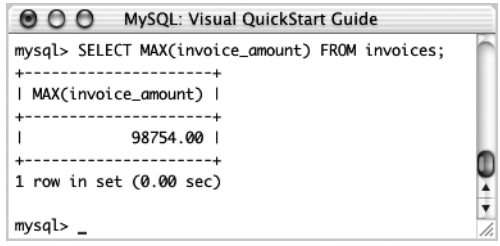

Рис. 5.23. Функция MAX() выполняет группировку, даже если фраза GROUP BY явно не указана

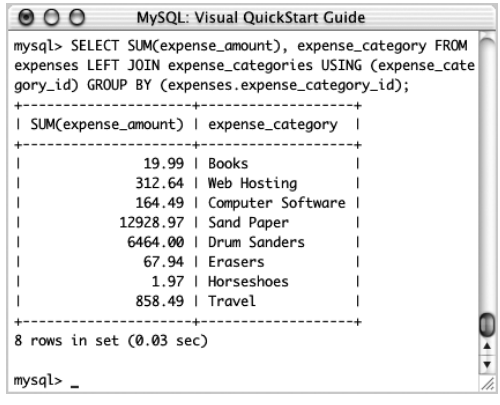

Рис. 5.24. Функцию SUM() в сочетании с фразой GROUP BY можно использовать для суммирования полей записей в группах с одним и тем же значением поля expense_category_id

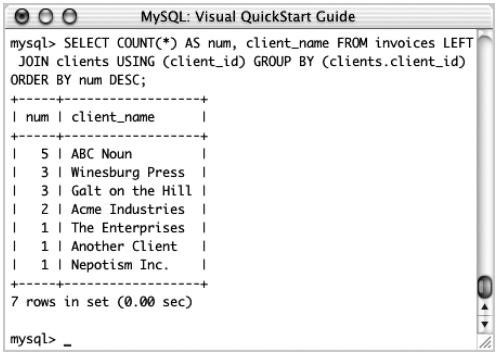

Рис. 5.25. Функция `COUNT()` возвращает количество записей в каждой группе, сформированной по команде `GROUP BY` (в данном случае группировка производится по полю client_id)

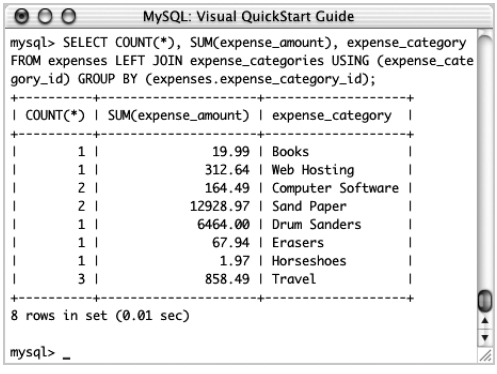

Рис. 5.26. Включив функцию `COUNT()` в запрос, показанный на рис. 5.24, я могу помимо всего прочего узнать, сколько записей оказалось в каждой категории

С Для того чтобы освоиться с фразой `GROUP BY` и агрегатными функциями, потребуется некоторое время. MySQL выдаст сообщение об ошибке, если вы введете синтаксически некорректный запрос. Поэкспериментируйте с монитором mysql, пока не разберетесь, как правильно формулировать запросы.

3. Выясните, сколько счетов выставлено каждому клиенту (рис. 5.25):

```
SELECT COUNT(*) AS num, client_name
⇨ FROM invoices LEFT JOIN clients
⇨ USING (client_id)
⇨ GROUP BY clients.client_id
⇨ ORDER BY num DESC;
```

Если нужно узнать, сколько записей в той или иной категории, на помощь придет сочетание функции `COUNT()` с фразой `GROUP BY`. Я применил `COUNT()` к каждой колонке (`*`), но мог бы с тем же успехом указать только колонку invoice_id. Сгруппированные результаты можно отсортировать, как и любые другие.

4. Измените запрос 2 так, чтобы в результатах было показано, сколько записей о затратах составило вычисленную сумму (рис. 5.26):

```
SELECT COUNT(*), SUM(expense_amount),
⇨ expense_category FROM expenses LEFT
⇨ JOIN expense_categories
⇨ USING(expense_category_id)
⇨ GROUP BY expense_categories.
⇨ expense_category_id;
```

Здесь я снова воспользовался функцией `COUNT()`, чтобы посчитать число записей в каждой группе. В одном запросе может присутствовать несколько агрегатных функций.

П Вы уже видели, что у значения `NULL` есть ряд особенностей. Вам будет интересно узнать, что команда `GROUP BY` сводит в одну группу все строки, содержащие в указанном поле `NULL`.

П Функция `COUNT()` подсчитывает только строки, не содержащие `NULL` в указанном поле.

Прочие функции

Прежде чем закончить эту главу, я упомяну еще две функции, которые нельзя отнести ни к одной из рассмотренных выше категорий. Первая из них – `LAST_INSERT_ID()` – очень важна для работы с реляционными СУБД; вторая – `DISTINCT()` – позволяет уточнить состав возвращаемых данных.

Функция `LAST_INSERT_ID()` возвращает значение автоинкрементного поля, установленное в результате выполнения последнего предложения `INSERT`. Например, в таблице expense_categories из базы данных accounting поле expense_category_id определено как целое без знака, не содержащее `NULL`. Кроме того, оно назначено первичным ключом и объявлено автоинкрементным. Это означает, что при попытке вставить `NULL` в данное поле MySQL автоматически запишет туда следующее по порядку значение, а `LAST_INSERT_ID()` его вернет. Затем полученный результат можно использовать в качестве значения внешнего ключа при вставке записи в связанную таблицу:

```
SELECT LAST_INSERT_ID();
```

Функция `DISTINCT()` используется для устранения дубликатов и обычно применяется к колонке:

```
SELECT DISTINCT(имя_колонки)
⇨ FROM имя_таблицы;
```

Часто `DISTINCT()` употребляют в сочетании с фразой `GROUP BY`, чтобы включить в группу только уникальные записи и, возможно, посчитать их.

Путаница с функцией LAST_INSERT_ID()

Часто считают, что функция `LAST_INSERT_ID()` возвращает значение последнего вставленного в таблицу поля независимо от того, как это было сделано. Такое суждение неверно. Функция возвращает последнее значение, вставленное в том же сеансе связи с сервером, в котором она была вызвана. К примеру, если с базой данных одновременно соединились десять сценариев и три монитора mysql, вставили в таблицу новую строку, а затем запросили значение автоинкрементного поля, то каждый из них получит значение, соответствующее своему сеансу.

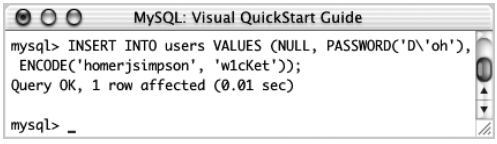

Рис. 5.27. Для проверки функции `LAST_INSERT_ID()` я добавлю пользователя в таблицу users

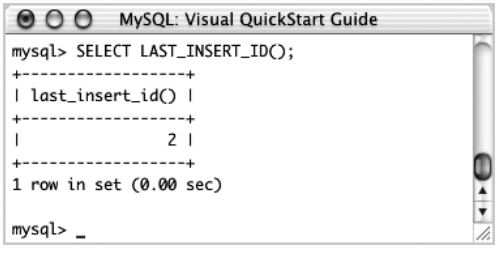

Рис. 5.28. Функция `LAST_INSERT_ID()` возвращает одно число, соответствующее последнему предложению `INSERT`

Использование функции LAST_INSERT_ID()

1. Вставьте нового пользователя в таблицу users (рис. 5.27):

   ```
   INSERT INTO users
   ⇨ VALUES(NULL, PASSWORD('D\'oh'),
   ⇨ ENCODE('homerjsimpson', 'w1cKet'));
   ```

   Работая с функциями шифрования, позаботьтесь о том, чтобы при вставке каждой строки использовались одна и та же функция и один и тот же параметризующий аргумент, иначе вам будет трудно восстановить данные в исходном виде.

2. Выберите идентификатор только что вставленного пользователя (рис. 5.28):

   ```
   SELECT LAST_INSERT_ID();
   ```

   Эта команда возвращает всего одно число, равное значению первичного ключа, вставленного вместо `NULL` на этапе 1.

Использование функции DISTINCT()

1. Перечислите разных клиентов, которым были выставлены счета (рис. 5.29):

```
SELECT DISTINCT(invoices.client_id),
⇨ client_name FROM invoices, clients
⇨ WHERE invoices.client_id =
⇨ clients.client_id;
```

   По этому запросу каждый клиент выводится только один раз – независимо от того, сколько ему было выставлено счетов.

2. Подсчитайте, скольким различным клиентам были выставлены счета (рис. 5.30):

```
SELECT COUNT(DISTINCT(client_id))
⇨ FROM invoices;
```

   Если `COUNT()` сочетается с `DISTINCT()`, то при подсчете будут учтены только уникальные значения.

Функция `LAST_INSERT_ID()` возвращает то же самое значение, что и функция PHP `mysql_ insert_id()`.

При вставке сразу нескольких записей функция `LAST_INSERT_ID()` возвращает значение, присвоенное автоинкрементному полю в первой вставленной записи.

В MySQL есть и другие функции, например `DATABASE()`. Так, запрос `SELECT DATABASE();` вернет имя базы данных, с которой вы в настоящий момент работаете.

Функция `USER()`, употребляемая в контексте `SELECT USER();`, возвращает имя пользователя, от имени которого вы работаете.

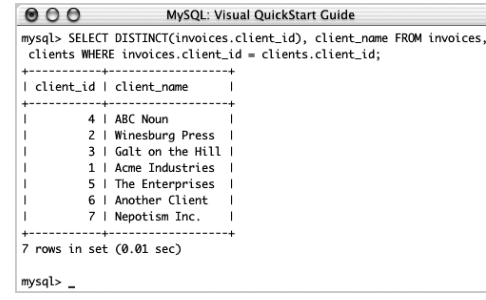

Рис. 5.29. Функция `DISTINCT()` в определенном отношении напоминает агрегатные функции: она тоже группирует записи

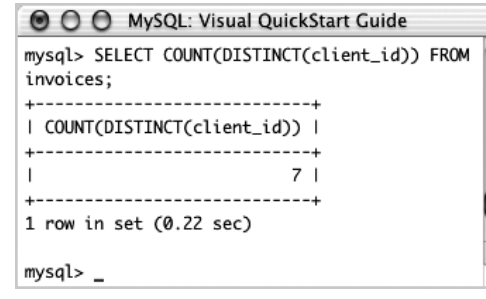

Рис. 5.30. Использование `DISTINCT()` в сочетании с `COUNT()` позволяет вывести число уникальных значений в колонке

6

MySQL и PHP

Для работы с MySQL используются различные языки, в том числе Perl, Java, C, C++ и Python, но чаще других – язык PHP. Сообщество его пользователей постоянно расширяется (одна из наиболее веских причин в том, что исходный текст PHP открыт), а тесная интеграция с MySQL еще больше увеличивает популярность этого продукта.

Поддержка MySQL встроена в PHP по умолчанию начиная с версии 4. Последняя версия (на момент написания этой книги – 4.2) предоставляет целый ряд возможностей, ориентированных именно на MySQL, и включает поддержку MySQL вплоть до версии 4.0. В этой главе предполагается, что вы используете версию PHP 4.0 или старше.

Данная глава рассчитана на несколько категорий пользователей: знакомых с PHP, но еще не работавших с MySQL; знакомых и с PHP, и с MySQL, но желающих освежить свои базовые знания; наконец, незнакомых с PHP и желающих узнать, как ведется взаимодействие с MySQL.

Мы продолжим работу с базой данных accounting и создадим на PHP интерфейс для управления затратами. Полученные знания позволят вам без труда создать аналогичный интерфейс для управления счетами.

Соединение с MySQL и выбор базы данных

Прежде всего, следует установить соединение с сервером MySQL. В PHP для этого предназначена функция `mysql_connect()`:

```
$db_connection = mysql_connect(имя_хоста,
⇨ имя_пользователя, пароль);
```

Параметры *имя_пользователя* и *пароль* – это имя и пароль одного из пользователей, зарегистрированных в базе данных mysql (см. главу 2). В качестве имени хоста обычно указывается `localhost`, хотя это и не обязательно.

В переменной `$db_connection` хранится ссылка на созданное соединение. Большинство функций PHP для работы с MySQL принимают ее в качестве необязательного аргумента, а если он не задан, автоматически используют открытое соединение.

После того как связь с MySQL установлена, нужно выбрать рабочую базу данных. В мониторе mysql для этой цели использовалось предложение `USE` *имя_базы_данных*, а в PHP тот же эффект достигается вызовом функции `mysql_select_db()`:

```
mysql_select_db(имя_базы_данных);
```

Чтобы показать, как устанавливается соединение с MySQL, я создам специальный файл, который можно будет включать во все PHP-сценарии, работающие с MySQL.

Установление соединения и выбор базы данных

1. Создайте в своем любимом текстовом редакторе новый PHP-документ (см. листинг 6.1). Начните со строки

   ```
   <?php
   ```

Листинг 6.1. Файл mysql_connect.inc будет использоваться во всех остальных сценариях для установления соединения с базой данных

Листинг

```
<?php

// ***** mysql_connect.inc *****
// ***** Листинг 6.1 *****
// Автор: Larry E. Ullman
// MySQL: Visual QuickStart Guide
// Контактный адрес:
mysql@DMCinsights.com
// Дата создания: 7 мая 2002 года
// Дата последней модификации:
// 7 мая 2002 года
// Файл содержит информацию о доступе
// к базе данных accounting.
// Здесь же устанавливается соединение
// с MySQL и выбирается база данных.

// Информация, относящаяся к конкретной
// базе данных:
DEFINE (DB_USER, "имя_пользователя");
DEFINE (DB_PASSWORD, "пароль");
DEFINE (DB_HOST, "localhost");
DEFINE (DB_NAME, "accounting");

// Подключиться к MySQL:
$db_connection = mysql_connect
⇨ (DB_HOST, DB_USER, DB_PASSWORD);

// Выбрать базу данных:
mysql_select_db (DB_NAME);
?>
```

2. Добавьте комментарий:

```
// ***** mysql_connect.inc *****
// ***** Листинг 6.1 *****
// Автор: Larry E. Ullman
// MySQL: Visual QuickStart Guide
// Контактный адрес:
// mysql@DMCinsights.com
// Дата создания: 7 мая 2002 года
// Дата последней модификации:
// 7 мая 2002 года
// Файл содержит информацию
// о доступе к базе данных accounting.
// Здесь же устанавливается
// соединение с MySQL и выбирается
// база данных.
```

По большей части я постараюсь воздерживаться от комментирования отдельных фрагментов, но сейчас хотелось бы особо подчеркнуть эти строки, чтобы вы поняли, как можно документировать конфигурационный файл. Кроме того, для удобства я буду включать имя файла и номер листинга в комментарии к каждому сценарию (чтобы проще было давать ссылки).

3. Определите имя хоста, имя и пароль пользователя, а также имя базы данных в виде констант:

```
DEFINE(DB_USER, "имя_пользователя");
DEFINE(DB_PASSWORD, "пароль");
DEFINE(DB_HOST, "localhost");
DEFINE(DB_NAME, "accounting");
```

Я предпочитаю представлять такого рода информацию в виде констант из соображения безопасности (тогда их невозможно будет случайно изменить), но, вообще говоря, это необязательно. Так или иначе, вынесение указанных параметров в отдельный файл позволяет отделить их от кода функций. Это удобно, хотя опять же вы вправе поступить по-другому.

4. Установите соединение с MySQL:

```
$db_connection = mysql_connect
⇨ (DB_HOST, DB_USER, DB_PASSWORD);
```

Если функции `mysql_connect()` удалось подключиться к MySQL, то она возвратит ссылку на открытое соединение. Эта ссылка запоминается в переменной `$db_connection`, хотя нигде в сценариях я не буду ее явно использовать.

5. Выберите рабочую базу данных:

```
mysql_select_db (DB_NAME);
```

Здесь вы сообщаете MySQL, к какой базе данных будут обращены запросы. Если не указать ее, то в других сценариях возникнут проблемы, хотя, когда приложение работает сразу с несколькими базами, заранее выбирать одну из них в глобальном плане не имеет особого смысла.

6. Закройте сценарий и сохраните файл, присвоив ему имя mysql_connect.inc:

```
?>
```

Я предпочитаю расширение .inc, и это нормально, если только файл сохраняется вне каталога, доступного из Сети (см. п. 7). При желании можете сохранить документ и с расширением .php.

7. Загрузите файл на сервер в корневой каталог Web-приложений (рис. 6.1).

Поскольку данный документ содержит конфиденциальную информацию о доступе к MySQL, следует обеспечить безопасное хранение файла. Лучше всего поместить его в каталог, являющийся прямым родителем каталога Web-приложения, или в такое место, которое недоступно из Сети. Если это невозможно, присвойте файлу расширение .php и поместите в каталог, защищенный паролем.

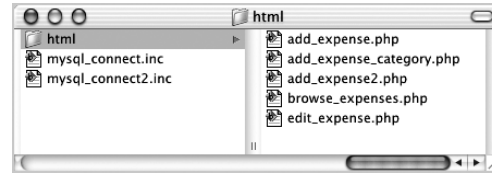

Рис. 6.1. Если в качестве корневого каталога для всех Web-документов выступает html (то есть именно туда ведет адрес www.dmcinsights.com), конфигурационные файлы следует хранить вне его

П Если при вызове функции `mysql_connect()` вы получаете сообщение о том, что функция не определена, это означает, что при компиляции PHP не была включена поддержка MySQL.

П В PHP есть также функция `mysql_pconnect()`, устанавливающая постоянное соединение с MySQL. Но ее использование заставляет пересмотреть архитектуру всего приложения, так как способ работы с подобными соединениями довольно-таки специфичен.

С Если в сценарии mysql_connect.inc модифицировать строки `DEFINE()`, то его можно будет использовать и в других проектах.

Простые запросы

Успешно соединившись с сервером и выбрав рабочую базу данных, вы можете приступать к выполнению запросов. Это могут быть как простые команды вставки, обновления и удаления, так и сложные выборки из нескольких таблиц, возвращающие более одной строки. В любом случае используется функция PHP `mysql_query()`:

```
$query_result = mysql_query($query);
```

Переменная `$query_result` содержит ссылку на результат выполнения запроса, то есть указатель на данные, возвращенные MySQL.

Последние два этапа сценария предполагают освобождение ресурсов, захваченных в результате выполнения запроса, и закрытие соединения с сервером:

```
mysql_free_result($query_result);
mysql_close();
```

В первой строке освобождается память, выделенная под хранение результатов запроса. Во второй строке закрывается соединение, открытое функцией `mysql_connect()`. Ни один из этих вызовов не является обязательным, так как PHP и MySQL автоматически освободят ресурсы и закроют соединение по завершении работы сценария, но включение данных строк считается «хорошим тоном» в программировании.

Чтобы продемонстрировать всю процедуру, я создам PHP-страницу, которая выводит форму и обрабатывает введенную в нее информацию. Назначение такой страницы – реализовать добавление новых категорий затрат в базу данных.

Выполнение простых запросов

1. Создайте в редакторе новый PHP-документ (листинг 6.2).
2. Начнем со стандартного пролога HTML-кода:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
⇨ XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/2000/REC-
⇨ xhtml1-20000126/DTD/xhtml1-
⇨ transitional.dtd">

<html xmlns="http://www.w3.org/
⇨ 1999/xhtml">

<head>

<title>Add An Expense Category</title>

</head>

<body>
```

В этой книге я буду следовать рекомендациям XHTML, поэтому код может несколько отличаться от того, к чему вы привыкли. Впрочем, для изложения материала это не имеет существенного значения. В заголовке страницы значится Add An Expense Category (Добавление категории затрат).

3. Начните раздел PHP-кода:

```
<?php
```

4. Проверьте, была ли форма отправлена:

```
if (isset($HTTP_POST_VARS['submit']))
{
```

Листинг 6.2. Эта PHP-страница позволяет добавлять записи в базу данных с помощью браузера

Листинг

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Add An Expense Category</title>
</head>
<body>
<?php

// ***** add_expense_category.php *****
// ***** Листинг 6.2 *****
// Этот сценарий выводит форму для вставки записей в таблицу
// expense_categories и обрабатывает введенные данные

if (isset($HTTP_POST_VARS['submit'])) { // Если форма отправлена, обработать ее.

  // Проверить, все ли обязательные поля заполнены.
  if (strlen($HTTP_POST_VARS['expense_category']) > 0) {

     // Включить файл с данными о соединении с MySQL:
     require_once ("../mysql_connect.inc");
```

Листинг 6.2. Эта PHP-страница позволяет добавлять записи в базу данных с помощью браузера (окончание)

Листинг

```
    // Создать запрос:
    $query = "INSERT INTO expense_categories VALUES (NULL,
                          '{$HTTP_POST_VARS['expense_category']}')";

    // Выполнить запрос:
    $query_result = mysql_query ($query);

    // Вывести сообщение об удачном или ошибочном завершении:
    if ($query_result) {
       echo '<b><font color="green">The category has been added!</font></b>';
    } else {
       echo '<b><font color="red">The category could not be added!</font></b>';
    }

    // Освободить ресурсы и закрыть соединение с базой данных:
    mysql_close();

  } else { // Напечатать сообщение, если категория не введена:
    echo '<b><font color="red">You forgot to enter the category!</font></b>';
  }

} else { // Если форма не отправлена, вывести ее.

// Для упрощения ввода выйти из режима интерпретатора PHP:
?>
Add a new category to the expense_categories table:<br />
<form action="add_expense_category.php" method="post">
<input type="text" name="expense_category" size="30" maxlength="30" />
<p />
<input type="submit" name="submit" value="Submit!" />
</form>
<?php
} // Конец главного условного оператора.
?>
</body>
</html>
```

Данная страница обеспечивает выполнение сразу двух операций: вывода HTML-формы и обработки введенных в нее данных. Поэтому я написал один большой условный оператор, который решает, что именно нужно делать, основываясь на наличии или отсутствии значения у переменной окружения `submit`. С этой техникой мы будем сталкиваться постоянно.

5. Проверьте, все ли обязательные поля заполнены:

```
if (strlen($HTTP_POST_VARS['expense_
⇨ category']) > 0) {
```

Я не хочу, чтобы сценарий вставлял категории затрат с пустым названием, поэтому сначала проверяю, введено ли что-нибудь в поле expense_category. Как правило, все поля базы данных, которые не могут содержать `NULL` (за исключением первичных ключей), следует проверять на наличие значения. Контроль введенных в форму данных – важнейшее условие безопасности вашей базы!

6. Включите файл, содержащий информацию о соединении с сервером:

```
require_once ("../mysql_connect.inc");
```

В этой строке в сценарий включается содержимое файла mysql_connect.inc, а следовательно, устанавливается соединение с сервером и выбирается база данных. Возможно, вам придется изменить путь к включаемому файлу – это зависит от того, где вы его сохранили.

7. Сформулируйте и выполните запрос:

```
$query = "INSERT INTO
⇨ expense_categories VALUES
(NULL, '{$HTTP_POST_VARS['expense_
⇨ category']}')";

$query_result = mysql_query ($query);
```

Сам запрос аналогичен рассмотренным в главе 4. Записав его текст в переменную, вы передаете ее функции `mysql_run()`, которая отправляет запрос серверу MySQL.

8. Напечатайте то или иное сообщение:

```
if ($query_result) {
echo '<b><font color="green">The
⇨ category has been added!</font></b>';

} else {

echo '<b><font color="red">
⇨ The category could not be added!
⇨ </font></b>';

}
```

Переменной `$query_result` присваивается ссылка на данные, возвращенные MySQL. Ее можно использовать как индикатор успешности выполнения операции (`The category has been added` – «Категория была добавлена», `The category could not be added` – «Категорию не удалось добавить»). В данном случае можно было бы сэкономить одну строку кода, написав

```
if (mysql_query($query)) {
```

9. Закройте соединение с базой данных:

```
mysql_close();
```

Хотя закрывать соединение с базой необязательно, это всегда стоит делать (если, конечно, подключение больше не понадобится). Для вышеприведенного запроса вызывать функцию `mysql_free_result()` не нужно, так как он не возвращает никаких строк.

10. Закончите первую ветку условного оператора и откройте вторую:

```
} else {
echo '<b><font color="red">You
⇨ forgot to enter the category!
⇨ </font></b>';
```

```
}
} else { // Если форма не отправлена,
  ⇨ вывести ее.
```

Здесь завершается часть сценария, посвященная обработке введенных данных. Далее приступим к выводу формы.

11. Создайте HTML-форму:

```
?>

Add a new category to the
⇨ expense_categories table: <br />

<form action="add_expense_
⇨ category.php" method="post">

<input type="text" name="expense_
⇨ category" size="30"
⇨ maxlength="30" />
<p />

<input type="submit" name="submit"
⇨ value="Submit!" />

</form>
```

Я решил сделать форму максимально простой. Заметьте, что имя поля в HTML-форме в точности совпадает с именем колонки в таблице базы данных. Это не строгое предписание, но вероятность ошибок благодаря этому уменьшается. Кроме того, поскольку вывод и обработку формы осуществляет одна и та же страница, в атрибуте `action` находится имя данного сценария. Вы могли бы внести небольшое усовершенствование, включив в форму несколько полей для ввода наименовании категории затрат. Тогда можно было бы за один раз выполнить несколько операций вставки.

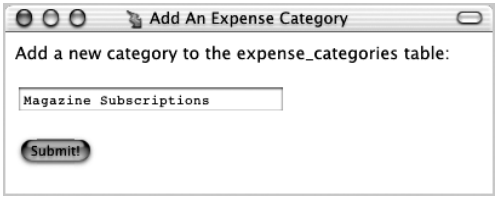

Рис. 6.2. Простая форма, выведенная сценарием add_expense_category.php, позволяет добавлять записи в таблицу expense_categories

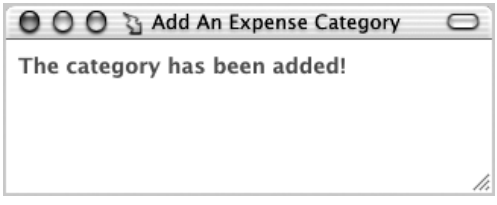

Рис. 6.3. Сценарий выводит сообщение об успешном добавлении новой категории

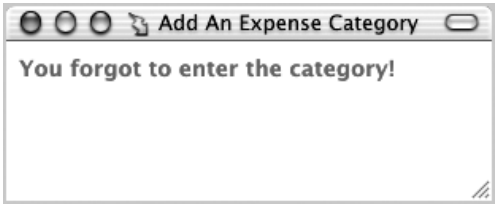

Рис. 6.4. Сценарий выводит сообщение о неправильно заполненной форме

12. Завершите сценарий:

```
<?php

}

?>

</body>

</html>
```

13. Сохраните файл под именем add_ expense_category.php.

Я предпочитаю длинные, содержательные имена файлов. Вы можете выбрать любое другое название, но не забудьте соответственно изменить атрибут `action` в тэге `<form>`.

14. Протестируйте сценарий, обратившись к нему из браузера (рис. 6.2–6.4).

После того как работа сценария завершилась, вы можете проверить занесенные в таблицу данные с помощью монитора mysql.

П В PHP не нужно заканчивать текст запроса точкой с запятой, как в мониторе mysql. Это типичная, хотя и безвредная ошибка.

С Если в поле формы введен вопросительный знак, его следует экранировать символом обратной косой черты перед вставкой записи в базу. В противном случае символ `?` будет принят за разделитель значений колонок (подробнее об этом рассказывается ниже, во врезке «Волшебные кавычки» из раздела «Безопасность»).

С Создавать переменную `$query` необязательно: можно сразу передать текст запроса как аргумент функции `mysql_query()`. Однако по мере усложнения запросов станет трудно обходиться без отдельной переменной.

Массив $HTTP_POST_VARS

Из соображений безопасности я воспользовался массивом `$HTTP_POST_VARS`, предоставляемым PHP, вместо того чтобы напрямую ссылаться на поля формы. Разница состоит в том, что вместо `$expense_category` я пишу `$HTTP_POST_VARS['expense_category']`, а это значит, что при формулировании запроса следует употреблять `{$HTTP_POST_VARS['expense_category']}`, чтобы PHP правильно подставил значение. Использовать массив `$HTTP_POST_VARS` можно в случае, когда форма отправлена методом `POST` (методу `GET` соответствует массив `$HTTP_GET_VARS`).

Поскольку PHP развивается в направлении ограниченного использования глобальных переменных, то программисты должны отдавать предпочтение именно этой новой технике. При работе с PHP версии 4.1 и старше можно пойти еще дальше и воспользоваться суперглобальными массивами `$_POST` и `$_GET`, которые аналогичны массивам `$HTTP_POST_VARS` и `$HTTP_GET_VARS`, но имеют глобальную область действия.

Таблица 6.1. Передача одной из этих констант в качестве параметра функции `mysql_fetch_array()` определяет способ возврата прочитанной из базы информации

Константа	Пример
`MYSQL_ASSOC`	`$row['column']`
`MYSQL_NUM`	`$row[0]`
`MYSQL_BOTH`	`$row[0]` **или** `$row['column']`

Выборка данных

В предыдущем разделе я продемонстрировал простой запрос к базе данных MySQL. К простым относятся запросы, начинающиеся со слов `INSERT`, `UPDATE`, `DELETE` и `ALTER`. Их общая черта в том, что они возвращают не данные, а только признак успешного или ошибочного завершения. В противоположность этому, запрос на выборку (`SELECT`) возвращает набор строк, для обработки которого в PHP предусмотрены дополнительные функции.

Главная из них – функция

`mysql_fetch_array()`

которая возвращает по одной строке результата выборки в виде массива. Обычно она вызывается в цикле до тех пор, пока еще есть данные. Типичная конструкция выглядит так:

```
while ($row = mysql_fetch_
⇨ array($query_result)) {

// Сделать что-то со строкой $row.
```

Функция принимает дополнительный параметр, указывающий тип возвращаемого массива: ассоциативный, индексированный или смешанный. *Ассоциативный массив* позволяет ссылаться на значения колонок по имени, а *индексированный* – по числовому индексу (индекс первой колонки равен 0). Значениями параметра могут быть константы, перечисленные в табл. 6.1.

Чтобы продемонстрировать, как обрабатываются результаты выполнения запроса, я создал сценарий, добавляющий новую запись о затратах в базу. Выпадающий список, содержащий названия категорий затрат, будет создан на основе содержимого таблицы expense_categories. Общая структура сценария очень похожа на add_expense_category.php.

Извлечение данных

1. Создайте в редакторе новый PHP-документ (листинг 6.3).
2. Начните со стандартного пролога HTML-кода. В заголовке страницы значится Enter An Expense (Ввести затрату):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
⇨ XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/2000/REC-
⇨ xhtml1-20000126/DTD/xhtml1-
⇨ transitional.dtd">

<html xmlns="http://www.w3.org/
⇨ 1999/xhtml">

<head>

<title>Enter An Expense</title>

</head>

<body>
```

Листинг 6.3. Сценарий add_expense_category.php извлекает данные из таблицы expense_categories для создания выпадающего списка

Листинг

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Enter An Expense</title>
</head>
<body>
<?php

// ***** add_expense.php *****
// ***** Листинг 6.3 *****
// Эта страница выводит форму для вставки записей в таблицу expenses
// и обрабатывает введенные в нее данные.

// Включить файл с данными о соединении с MySQL:
require_once ("../mysql_connect.inc");

if (isset($HTTP_POST_VARS['submit'])) { // Если форма была отправлена, обработать ее.

  // Проверить, заполнены ли обязательные поля:
  if (isset($HTTP_POST_VARS['expense_category_id']) AND
            (strlen($HTTP_POST_VARS['expense_amount']) > 0) AND
            (strlen($HTTP_POST_VARS['expense_description']) > 0) ) {

     // Создать запрос:
     $query = "INSERT INTO expenses VALUES (NULL,
     ⇨ {$HTTP_POST_VARS['expense_category_id']}, '" .
     ⇨ addslashes($HTTP_POST_VARS['expense_amount']) . "', '" .
     ⇨ addslashes($HTTP_POST_VARS['expense_description']) . "', NOW())";
```

Листинг 6.3. Сценарий add_expense_category.php извлекает данные из таблицы expense_categories для создания выпадающего списка (окончание)

Листинг

```
    // Вывести сообщение об удачном или ошибочном завершении:
    if (mysql_query ($query)) {
                    echo '<b><font color="green">The expense has been added!</font></b>';
    } else {
                    echo '<b><font color="red">The expense was not entered into the table!
                  ⇨ </font></b>';
    }
  } else { // Напечатать сообщение, если поле, обязательное для заполнения,
           // осталось пустым.
            echo '<b><font color="red">You missed a required field!</font></b>';
  }

} else { // Если не было отправки, вывести форму.

  echo 'Enter an expense:<br />
  <form action="add_expense.php" method="post">
  Expense Category: <select name="expense_category_id">';

  // Вывести категории затрат:
  $query_result = mysql_query ('SELECT * FROM expense_categories ORDER BY expense_category');
  while ($row = mysql_fetch_array ($query_result, MYSQL_NUM)) {
     echo "<option value=\"$row[0]\">$row[1]</option>\n";
  }

  // Очистка (необязательно):
  mysql_free_result($query_result);
  mysql_close();

  // Завершить создание формы:
  echo '</select><p />
  Expense Amount: <input type="text" name="expense_amount" size="10" maxlength="10" /><p />
  Expense Description: <textarea name="expense_description" rows="5" cols="40">
  ⇨ </textarea> <p />
  <input type="submit" name="submit" value="Submit!" />
  </form>';

} // Конец главного условного оператора.

?>
</body>
</html>
```

3. Начните раздел PHP-кода и включите туда файл, обеспечивающий соединение с MySQL:

```
<?php

require_once ("../mysql_connect.inc");
```

Поскольку доступ к базе необходим в обеих частях сценария (при выводе формы и обработке введенных данных), этот файл стоит включить сразу, а не после проверки условия, как в предыдущем примере.

4. Проверьте основное условие:

```
if (isset($HTTP_POST_VARS['submit']))
{
```

Как и раньше, этот оператор решает, что будет делать сценарий: выводить форму или обрабатывать ее.

5. Проверьте, заданы ли обязательные поля:

```
if (isset($HTTP_POST_VARS['expense_
⇨ category_id']) AND (strlen($HTTP_
⇨ POST_VARS['expense_amount']) > 0)
⇨ AND (strlen($HTTP_POST_VARS
⇨ ['expense_description']) > 0) ) {
```

Здесь проверяются три поля: expense_category_id, expense_amount и expense_description. Последние два представлены полями ввода, для которых функция `isset` может вернуть `true`, даже если значение не задано; поэтому следует посмотреть, положительна ли длина значения.

6. Сформулируйте запрос:

```
$query = "INSERT INTO expenses
⇨ VALUES (NULL, {$HTTP_POST_VARS
⇨ ['expense_category_id']}, '" .
⇨ addslashes($HTTP_POST_VARS
⇨ ['expense_amount']) . "', '" .
⇨ addslashes($HTTP_POST_VARS
⇨ ['expense_description']) . "',
⇨ NOW())";
```

Он отличается от того, что вы видели в листинге 6.2: здесь я воспользовался функцией `addslashes()`, автоматически экранирующей все символы, из-за которых могут возникнуть проблемы. Мы еще затронем эту тему в дальнейшем, но уже сейчас я хочу познакомить вас с данной техникой.

7. Выполните запрос:

```
if (mysql_query ($query)) {

echo '<b><font color="green">The
expense has been added!</font>
</b>';

} else {

echo '<b><font color="red">The
expense was not entered
into the table!</font></b>';
}
```

`The expense has been added` – «Запись о затрате была добавлена».

`The expense was not entered into the table` – «Запись о затрате не была введена в таблицу».

Выше я говорил, что выполнить запрос можно прямо в условии, что позволяет сэкономить строчку кода.

8. Завершите первую ветвь главного условного оператора:

```
} else {
echo '<b><font color="red">You
missed a required field!</font>
</b>';

}

} else {
```

`You missed a required field` – «Вы пропустили поле, обязательное для заполнения».

Если хотите сделать сценарий более профессиональным, можете добавить код,

который информирует пользователя о том, какие именно поля не заполнены.

9. Приступите к выводу HTML-формы:

```
echo 'Enter an expense:<br />

<form action="add_expense.php"
⇨ method="post">

Expense Category: <select
⇨ name="expense_category_id">';
```

Выпадающий список expense_category_id создается из значений, хранящихся в базе данных. Но, прежде чем обращаться к базе, нужно вывести начало тэга SELECT.

10. Сгенерируйте выпадающий список на основе результатов выборки из таблицы expense_categories:

```
$query_result = mysql_query
⇨ ('SELECT * FROM expense_categories
⇨ ORDER BY expense_category');

while ($row = mysql_fetch_array
⇨ ($query_result, MYSQL_NUM)) {

echo "<option value=\"$row[0]\">
⇨ $row[1]</option>\n";
}
```

Процедура преобразования таблицы в выпадающий список делится на три этапа:

- создается и выполняется запрос;
- выбираются возвращенные данные;
- для каждой возвращенной строки печатается тэг <option>.

В рассматриваемом случае я решил выбрать из таблицы все записи, отсортировав их по колонке expense_category. Фраза ORDER BY в запросе определяет порядок следования пунктов выпадающего списка. Функции mysql_fetch_array() передается параметр MYSQL_NUM, позволяющий обращаться к строкам по индексу.

11. Освободите ресурс и закройте соединение с базой данных:

```
mysql_free_result($query_result);

mysql_close();
```

Поскольку запрос возвращает данные, перед закрытием соединения я освободил память, обратившись к функции `mysql_free_result()`.

12. Завершите HTML-форму:

```
echo '</select><p />

Expense Amount: <input type="text"
⇨ name="expense_amount" size="10"
⇨ maxlength="10" /><p />

Expense Description: <textarea
⇨ name="expense_description" rows="5"
⇨ cols="40"></textarea>     <p />

<input type="submit" name="submit"
⇨ value="Submit!" />

</form>';
```

Таким образом выпадающий список закрывается, и в форму добавляются еще два поля ввода. Убедитесь, что имя кнопки `submit` написано так же (и с использованием того же регистра), как в условии главного ветвления.

13. Завершите сценарий:

```
}

?>

</body>

</html>
```

14. Сохраните сценарий в файле с именем add_expense.php, загрузите его на Web-сервер и протестируйте в браузере (рис. 6.5, 6.6).

Чтобы увидеть результат работы функции `mysql_fetch_array()` в цикле `while`, просмотрите в браузере исходный текст сгенерированной страницы (рис. 6.7).

На сайте www.dmcinsights.com/mysql есть дополнительные примеры, не вошедшие в книгу. Один из них содержит текст определяемой пользователем функции, которая упрощает создание выпадающих списков на основе табличных данных.

Функция `mysql_fetch_row()`, с которой вам, возможно, приходилось встречаться, эквивалентна `mysql_fetch_array($query_link, MYSQL_NUM)`.

Функция `mysql_fetch_assoc()` эквивалентна `mysql_fetch_array($query_link, MYSQL_ ASSOC)`.

В PHP есть также функция `mysql_fetch_object()` для извлечения результатов запроса. Она возвращает строку как объект.

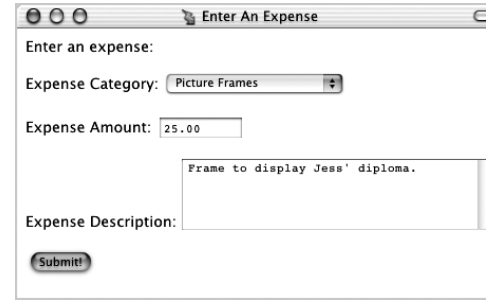

Рис. 6.5. Сценарий представляет данные из таблицы expense_categories в виде выпадающего списка, откуда пользователь выбирает категорию затрат

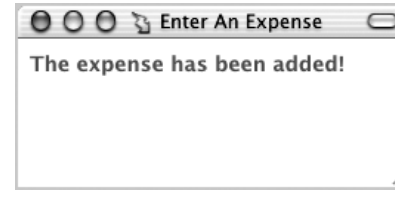

Рис. 6.6. Если не возникло ошибок, то после добавления записи о затрате выводится подтверждающее сообщение

Рис. 6.7. Просмотр HTML-кода страницы с рис. 6.5 показывает, как PHP сгенерировал выпадающий список

Применение функции mysql_insert_id()

Когда дело доходит до первичных и внешних ключей, могут возникнуть некоторые сложности. Ради поддержания целостности связи первичный ключ для одной таблицы (например, expense_categories) следует создать до вставки соответствующей записи в другую таблицу (expenses). В главе 5 вы уже видели, что в мониторе mysql для получения только что созданного первичного ключа (автоинкрементного поля) используется функция `LAST_INSERT_ID()`. В PHP ее аналогом служит функция `mysql_insert_id()`:

```
$id = mysql_insert_id();
```

При разработке приложения для работы с реляционной базой данных часто приходится писать сценарии типа add_expense.php, в которых первичный ключ из одной таблицы (expense_categories) связывается с внешним ключом из другой (expenses). Однако, если при добавлении записи о затратах указана еще не существующая категория затрат, новый первичный ключ (значение поля expense_category_id) необходимо создать до использования его в качестве внешнего (в новой записи о затратах). Ниже будет показано, как действовать в таких ситуациях.

Чтобы продемонстрировать применение функции `mysql_insert_id()`, я модифицирую страницу, которая выводит и обрабатывает форму для ввода информации о затратах. Как и раньше, я создам выпадающий список, содержащий названия всех имеющиеся категорий затрат, но также предоставлю пользователю возможность создать новую категорию, не закрывая форму. Вновь созданная категория будет добавлена в базу, а ее первичный ключ будет выступать в роли внешнего в таблице expenses.

Применение функции mysql_insert_id()

1. Откройте в редакторе файл add_expense.php (см. листинг 6.3).
2. В первой ветви условного оператора (листинг 6.3, строка 23) измените запрос (см. листинг 6.4) на

```
$query = "INSERT INTO expenses
⇨ VALUES (NULL, '";
```

Окончательный запрос аналогичен тому, что вы видели в листинге 6.3, но написан с учетом возможности ввода новой категории затрат. В этой строке начинается построение запроса.

Листинг 6.4. В более развитой версии сценария add_expense2.php пользователь может либо выбрать категорию затрат из списка, либо самостоятельно ввести ее название

Листинг

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Enter An Expense</title>
</head>
<body>
<?php

// ***** add_expense2.php *****
// ***** Листинг 6.4 *****
// Эта страница выводит форму для вставки записей в таблицу expenses
// и обрабатывает введенные в нее данные.

// Включить файл с данными о соединении с MySQL:
require_once ("../mysql_connect2.inc");

if (isset($HTTP_POST_VARS['submit'])) { // Если форма была отправлена, обработать ее.

  // Проверить, заполнены ли обязательные поля:
  if (isset($HTTP_POST_VARS['expense_category_id']) AND
              (strlen($HTTP_POST_VARS['expense_amount']) > 0) AND
              (strlen($HTTP_POST_VARS['expense_description']) > 0) ) {

             // Начало запроса:
             $query = "INSERT INTO expenses VALUES (NULL, '";

             // Была ли введена новая категория?
              if (strlen($HTTP_POST_VARS['expense_category']) > 0) {
```

Листинг 6.4. В более развитой версии сценария add_expense2.php пользователь может либо выбрать категорию затрат из списка, либо самостоятельно ввести ее название (продолжение)

Листинг

```
            // Создать второй запрос:
    $query2 = "INSERT INTO expense_categories VALUES (NULL, '" .
                    addslashes($HTTP_POST_VARS['expense_category']) . "')";

    // Выполнить второй запрос и проверить результат:
    if (mysql_query ($query2)) {
       echo '<b><font color="green">The expense category has been added!</font></b>
       ⇨ <br />';
       $query .= mysql_insert_id() . ", ";
    } else {
       echo '<b><font color="red">The expense category was not entered into the table!
       ⇨ </font></b><br />';
       $problem = TRUE;
    }

          } else { // Конец проверки ввода новой категории.
             $query .= "{$HTTP_POST_VARS['expense_category_id']}, ";
          }

          // Закончить построение запроса:
           $query .= "'" . addslashes($HTTP_POST_VARS['expense_amount']) .
               "', '" . addslashes($HTTP_POST_VARS['expense_description']) .
              "', NOW())";

          // Были обнаружены ошибки?
          if (!$problem) {

    // Напечатать сообщение об успешном выполнении операции или ошибке:
    if (mysql_query ($query)) {
       echo '<b><font color="green">The expense has been added!</font></b>';
    } else {
       echo '<b><font color="red">The expense was not entered into the table!
         ⇨ </font></b>';
    }
    } else { // Если была ошибка:
    echo '<b><font color="red">The expense was not entered into
    ⇨ the table because the expense_category could not be added!</font></b>';
          }

} else { // Напечатать сообщение о том, что категория не задана:
          echo '<b><font color="red">You missed a required field!</font></b>';
}
```

Листинг 6.4. В более развитой версии сценария add_expense2.php пользователь может либо выбрать категорию затрат из списка, либо самостоятельно ввести ее название (окончание)

Листинг

```
} else { // Если форма не была отправлена, вывести ее.

  echo 'Enter an expense:<br />
  <form action="add_expense2.php" method="post">
  <ul>
  <li>Expense Category: <select name="expense_category_id">';

  // Вывести категории затрат:
  $query_result = mysql_query ('SELECT * FROM expense_categories
ORDER BY expense_category');
  while ($row = mysql_fetch_array ($query_result, MYSQL_NUM)) {
     echo "<option value=\"$row[0]\">$row[1]</option>\n";
  }

  mysql_free_result($query_result);
  mysql_close();

  // Закончить вывод формы:
  echo '</select></li>
  or<br />
  <li>Enter a new expense category: <input type="text"
name="expense_category" size="30" maxlength="30" /></li>
  </ul>
  Expense Amount: <input type="text" name="expense_amount" size="10"
  ⇨ maxlength="10"/><p />
  Expense Description: <textarea name="expense_description" rows="5" cols="40">
  ⇨ </textarea> <p />
  <input type="submit" name="submit" value="Submit!" />
  </form>';

} // Конец главного условного оператора.

?>
</body>
</html>
```

3. Добавьте новую категорию затрат, если она введена:

```
if (strlen($HTTP_POST_VARS
⇨ ['expense_category']) > 0) {

$query2 = "INSERT INTO
⇨ expense_categories VALUES
⇨ (NULL, '" . addslashes ($HTTP_
⇨ POST_VARS['expense_category']) .
⇨ "')";

if (mysql_query ($query2)) {
echo '<b><font color="green">The
⇨ expense category has been added!
⇨ </font></b><br />';
$query .= mysql_insert_id() . ", ";

} else {

echo '<b><font color="red">The
⇨ expense category was not entered
⇨ into the table!</font></b><br />';

$problem = TRUE;

}

} else {

$query .= "{$HTTP_POST_VARS
['expense_category_id']}, ";

}
```

Здесь применяется та же техника, что в сценарии add_expense_category.php (листинг 6.2). Сначала проверяется, была ли введена категория затрат. Если да, то в таблицу expense_categories вставляется новая запись, и с помощью функции `mysql_insert_id()` вы получаете значение ее первичного ключа с целью его последующего использования в главном запросе. Также выводится сообщение об успешном или неудачном выполнении запроса: `The expense has been added` – «Запись о затрате была добавлена»; `The expense was not entered into the table` – «Запись о затрате не была введена в таблицу».

Если новая категория затрат не вводилась, то в главный запрос будет включено значение поля формы expense_category_id, соответствующее выбранной из списка категории.

4. Завершите главный запрос:

```
$query .= "'" . addslashes($HTTP_
POST_VARS['expense_amount']) .
"', '" . addslashes($HTTP_POST_
VARS['expense_description']) .
"', NOW())";
```

5. Проверьте, были ли ранее ошибки, и, если нет, выполните запрос:

```
if (!$problem) {

if (mysql_query ($query)) {

echo '<b><font color="green">The
⇨ expense has been added!</font>
⇨ </b>';

} else {

echo '<b><font color="red">The
⇨ expense was not entered into the
⇨ table!</font></b>';

}

} else {

echo '<b><font color="red">The
⇨ expense was not entered into the
⇨ table because the expense_category
⇨ could not be added!</font></b>';

}
```

`The expense has been added` – «Запись о затрате была добавлена».

`The expense was not entered into the table` – «Запись о затрате не была введена в таблицу».

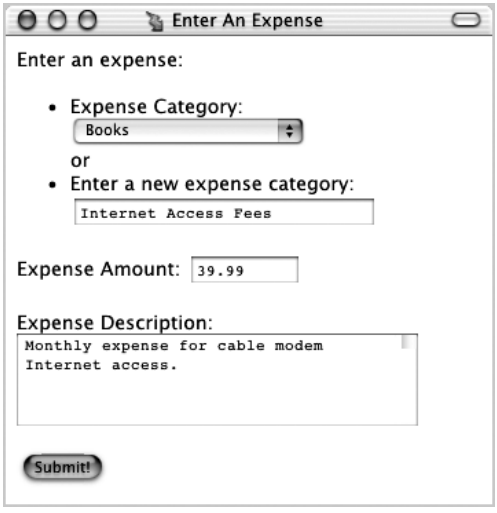

Рис. 6.8. Форма стала более «дружественной», так как пользователю не нужно сначала создавать категорию затрат, а потом уже вводить запись о затрате

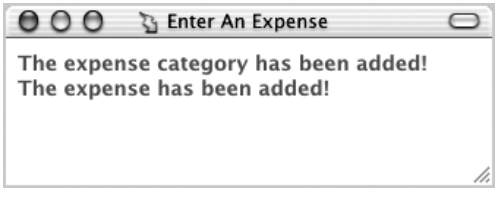

Рис. 6.9. Два сообщения показывают, что сценарий успешно выполнил вставку обеих записей в базу

```
The expense was not entered into the table
because the expense_category could not be
added
```

– «Запись о затрате не была введена в таблицу, поскольку невозможно добавить категорию затрат».

Переменная `$problem`, установленная на этапе 3, равна `TRUE`, если сценарий не смог добавить новую категорию затрат (в таком случае не следует вставлять и запись о затрате). В остальных отношениях эта часть сценария ничем не отличается от того, что вы делали раньше.

6. Добавьте в форму поле ввода expense_category:

```
<li>Enter a new expense category:

<input type="text" name="expense_
➪ category" size="30"
➪ maxlength="30" /></li>
```

Помимо добавления нового поля в форму я внес в HTML-код еще одно изменение – включил маркированный список, чтобы пользователь четко видел две возможности: выбрать существующую категорию затрат или ввести новую.

7. Сохраните файл, загрузите его на Web-сервер и протестируйте в браузере (рис. 6.8–6.11).

Я решил назвать этот файл add_expense2.php, чтобы отличать его от предыдущего. Поэтому соответствующим образом изменено значение атрибута `action` в тэге `<form>`.

С Функцию `mysql_insert_id()` следует вызывать сразу же после выполнения команды `INSERT`, прежде чем вы начнете любой другой запрос.

П Функция `mysql_insert_id()` возвращает 0, если в таблице, куда вставлялась запись, нет автоинкрементных колонок.

С Напомню, что числа в запросах SQL не следует заключать в кавычки. Поэтому значение поля expense_category_id включено в запрос без кавычек. Однако значение поля expense_amount все же закавычено, так как его источник – поле формы, куда пользователь по ошибке может ввести нецифровые символы.

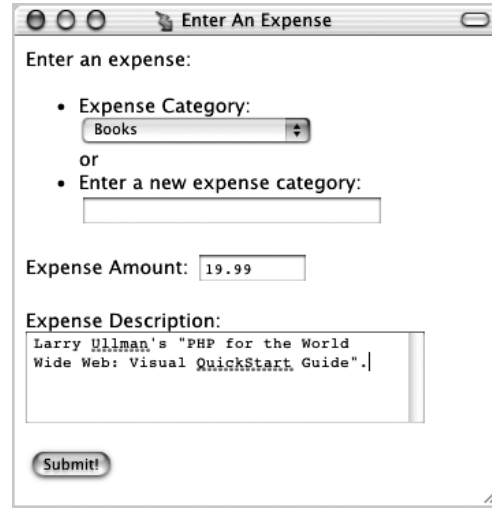

Рис. 6.10. Новая категория затрат не введена...

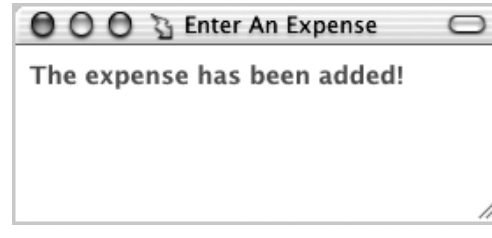

Рис. 6.11. ...поэтому сценарий работает так же, как предыдущий

Обработка ошибок

Обработка ошибок необходима в любом сценарии, но особую важность она приобретает при работе с базами данных. Вот наиболее распространенные ошибки:

- невозможность установить соединение с базой данных;
- невозможность выбрать рабочую базу данных;
- сбой при выполнении запроса;
- запрос не вернул результатов.

По мере накопления опыта вы поймете, когда возникают такого рода ошибки, но их немедленное обнаружение в сценарии поможет вам существенно сократить время отладки. Чтобы сценарий выдавал полезную информацию об ошибках, применяйте функции `mysql_error()` и `mysql_errno()`. Первая из них возвращает словесное описание ошибки, а вторая – ее числовой код.

Наряду с этими функциями в PHP есть еще два средства, имеющих отношение к обработке ошибок: символ `@` и функция `die()`. Если перед именем любой функции находится символ `@`, то будут подавлены все сообщения об ошибках и предупреждения, выдаваемые ею. Функция же `die()` немедленно завершает выполнение сценария и посылает браузеру строку, переданную ей в качестве аргумента. Вот как эти механизмы обычно применяются:

```
$db_connection = mysql_connect
⇨ (DB_HOST, DB_USER, DB_PASSWORD)
⇨ or die(mysql_error());

$query_result = @mysql_query($query);
```

Можно порекомендовать применение `die()` в тех случаях, когда успешное выполнение той или иной операции (например, соединения с сервером и выбора базы данных)

является необходимым условием дальнейшей работы сценария. Символ @ стоит употреблять тогда, когда функция может завершиться с ошибкой, но из-за этого не обязательно прекращать работу сценария.

Порядок обработки ошибок

1. Откройте в редакторе файл mysql_ connect.inc (листинг 6.1).
2. Измените код для установления соединения (строка 20), включив в него функции `die()` и `mysql_error()` – см. листинг 6.5:

   ```
   $db_connection = mysql_connect
   ⇨ (DB_HOST, DB_USER, DB_PASSWORD)
   ⇨ or die ('Could not connect to
   ⇨ MySQL: ' . mysql_error());
   ```

   `Could not connect to MySQL` – «Не удалось соединиться с MySQL».

   Поскольку функции `die()` может быть передана в качестве аргумента любая строка, вы можете формулировать сообщение об ошибке по своему усмотрению. Я просто воспользовался функцией `mysql_error()`, но мог бы, например, добавить HTML-разметку.

3. Измените код для выбора рабочей базы данных, добавив функцию обработки ошибок:

   ```
   mysql_select_db (DB_NAME) or
   ⇨ die ('Could not select the
   ⇨ database: ' . mysql_error());
   ```

   `Could not select the database` – «Не удалось выбрать базу данных».
4. Сохраните файл и загрузите его на Web-сервер.

   Чтобы было проще ссылаться на новую версию файла, я назвал его mysql_ connect2.inc.

Листинг 6.5. В новую версию файла mysql_connect.inc включены различные механизмы обработки ошибок

Листинг

```
<?php

// ***** mysql_connect.inc *****
// ***** Листинг 6.5 *****
// Автор: Larry E. Ullman
// MySQL: Visual QuickStart Guide
// Контактный адрес:
// mysql@DMCinsights.com
// Дата создания: 7 мая 2002 года
// Дата последней модификации:
// 7 мая 2002 года
// Файл содержит информацию о доступе
// к базе данных accounting.
// Здесь же устанавливается соединение
// с MySQL и выбирается база данных.

// Информация, относящаяся к конкретной
// базе данных:
DEFINE (DB_USER, "имя_пользователя");
DEFINE (DB_PASSWORD, "пароль");
DEFINE (DB_HOST, "localhost");
DEFINE (DB_NAME, "accounting");

// Установить соединение с MySQL:
$db_connection = mysql_connect (DB_HOST,
⇨ DB_USER, DB_PASSWORD) or
      die ('Could not connect to MySQL:
      ⇨ ' . mysql_error());

// Выбрать базу данных:
mysql_select_db (DB_NAME) or
   die ('Could not select the database:
   ⇨ ' . mysql_error());
?>
```

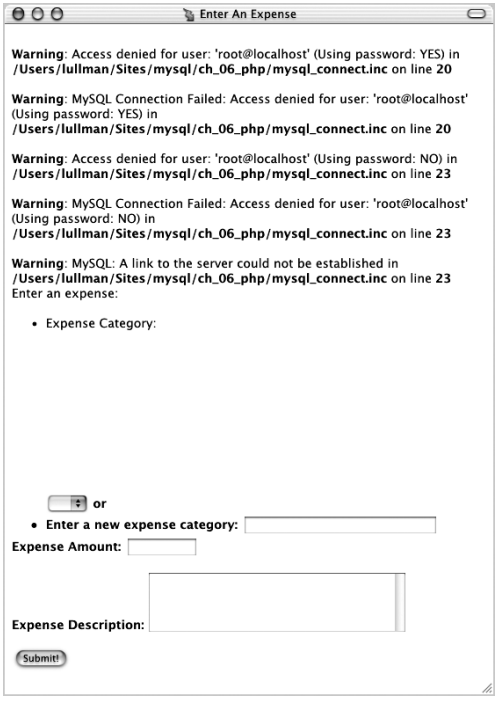

Рис. 6.12. Если бы ошибки не обрабатывались, даже самая простая из них привела бы к выдаче многочисленных сообщений и возврату искаженной страницы

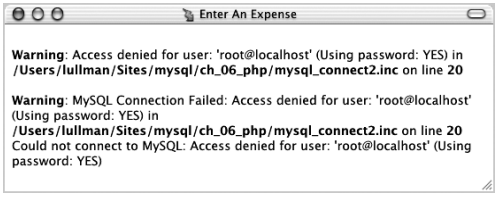

Рис. 6.13. При использовании конструкции `or die()` выводится не столь пугающая и более информативная страница

5. Проверьте, к чему привела модификация, изменив значения констант DB_USER, DB_PASSWORD, DB_HOST или DB_NAME на заведомо некорректные и попытавшись обратиться к какой-либо странице (рис. 6.12–6.14).

П В приложении 1 распространенные ошибки MySQL рассматриваются подробнее. Там же приводятся типичные причины их возникновения.

С Быть может, лучший метод отладки PHP-сценариев, обращающихся к серверу MySQL, заключается в том, чтобы выполнять запросы в мониторе mysql с целью контроля результатов.

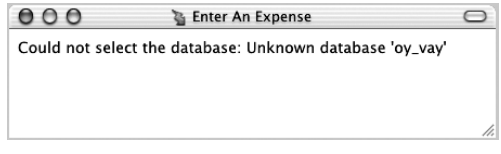

Рис. 6.14. Если выбор рабочей базы данных не состоялся, пользователь увидит простое сообщение об ошибке, и выполнение сценария прервется

Безопасность

В контексте PHP вопрос безопасности баз данных сводится к следующему:

- защита информации о доступе к базе;
- контроль информации, записываемой в базу.

Для решения первой проблемы следует обезопасить сценарий, содержащий параметры соединения. Выше в этой главе я уже упоминал некоторые способы безопасного хранения файла mysql_connect.inc. Наилучший, хотя и не всегда реализуемый на практике способ, – поместить файл вне корневого каталога Web-документов (рис. 6.1). Тогда браузер в принципе не сможет получить к нему доступ.

Вторую проблему можно решать по-разному. Во-первых, я уже объяснял, что следует использовать массив `$HTTP_POST_VARS` (или `$HTTP_GET_VARS`, `$_POST`, `$_GET`), а не глобальные переменные. Во-вторых, допускается употребление регулярных выражений (этот механизм в книге не рассматривается) для проверки правильности данных, полученных от пользователей. В-третьих, выше шла речь о применении к полученным данным функции `addslashes()`, которая экранирует недопустимые в запросе символы (того же эффекта можно добиться за счет механизма, описанного во врезке «Волшебные кавычки»). И наконец, начиная с версии PHP 4.0.3 предусмотрено использование функции `mysql_ escape_string()`:

```
$data = mysql_escape_string($data);
```

Она работает аналогично `addslashes()` – и должна применяться ко всем текстовым полям формы, – но в большей степени ориентирована на базы данных.

Волшебные кавычки

Механизм «волшебных кавычек» (возможность автоматически экранировать некоторые символы) многократно изменялся в ходе эволюции PHP. Это удобное средство для обработки символов одиночных и двойных кавычек в данных, полученных из HTML-формы. Но для поощрения «сознательного» программирования в последних версиях PHP этот механизм по умолчанию отключен, так что вы должны самостоятельно вызывать одну из функций:

```
addslashes()
```

или

```
mysql_escape_ string()
```

Выяснить, какую тактику применять, вы можете, запустив любой из вышеописанных сценариев и введя в текстовое поле строку, содержащую символ одиночной кавычки. Если при выполнении запроса возникает ошибка, то механизм «волшебных кавычек» отключен, и экранировать эти знаки придется своими силами. Если же в данных появляются повторяющиеся символы обратной косой черты, это значит, что и добавление «волшебных кавычек» обеспечено, и функция `addslashes()` / `mysql_escape_string()` вызвана. Следовательно, от чего-то одного нужно отказаться.

Чтобы продемонстрировать действие этой функции, равно как и еще одного приема, я написал сценарий, позволяющий пользователю редактировать запись о затратах.

Применение функции mysql_escape_string()

1. Создайте в текстовом редакторе новый PHP-документ, начав его со стандартного HTML-пролога (листинг 6.6). В заголовке страницы значится Edit An Expense (Редактировать данные о затрате):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
⇨ XHTML 1.0 Transitional//EN"

          "http://www.w3.org/TR/2000/
          ⇨ REC-xhtml1-20000126/DTD/
          ⇨ xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/
⇨ 1999/xhtml">

<head>

   <title>Edit An Expense</title>

</head>

<body>
```

Листинг 6.6. Последний в этой главе сценарий реализует улучшенный контроль безопасности за счет применения функции `mysql_escape_string()`. Кроме того, здесь демонстрируется выполнение команды `UPDATE`

Листинг

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Edit An Expense</title>
</head>
<body>
<?php
// ***** edit_expense.php *****
```

Листинг 6.6. Последний в этой главе сценарий реализует улучшенный контроль безопасности за счет применения функции `mysql_escape_string()`. Кроме того, здесь демонстрируется выполнение команды `UPDATE` (продолжение)

Листинг

```
// ***** Листинг 6.6 *****
// Эта страница выводит и обрабатывает форму для редактирования записей
// в таблице expenses. Входным параметром для нее служит идентификатор
// затраты eid (expense_id).

// Включить файл с данными о соединении с MySQL:
require_once ("../mysql_connect.inc");

if (isset($HTTP_POST_VARS['submit'])) { // Если форма была отправлена,
                                        // обработать ее.

    // Проверить, заполнены ли обязательные поля:
    if ( (isset($HTTP_POST_VARS['expense_category_id']) OR
          (strlen($HTTP_POST_VARS['expense_category']) > 0)) AND
          (strlen($HTTP_POST_VARS['expense_amount']) > 0) AND
          (strlen($HTTP_POST_VARS['expense_description']) > 0) ) {

        // Начало запроса:
        $query = "UPDATE expenses SET ";

        // Была ли введена новая категория?
        if (strlen($HTTP_POST_VARS['expense_category']) > 0) {

            // Создать второй запрос:
            $query2 = "INSERT INTO expense_categories VALUES (NULL,
 '{$HTTP_POST_VARS['expense_category']}')";

          // Выполнить второй запрос и проверить результат:
          if (mysql_query ($query2)) {
              echo '<b><font color="green">The expense category has
              ⇨ been added!</font></b><br />';
              $query .= "expense_category_id=" . mysql_insert_id() . ", ";
         } else {
         echo '<b><font color="red">The expense category was not
 entered into the table!</font></b><br />';
             $problem = TRUE;
    }

       } else { // Конец проверки ввода новой категории.
           $query .= "expense_category_id={$HTTP_POST_VARS ['expense_category_id']}, ";
      }
       // Закончить построение запроса:
```

2. Начните раздел PHP-кода и включите файл, который обеспечивает соединение с MySQL:

```
<?php

require_once ("../mysql_connect.inc");
```

Если вы переименовали файл mysql_connect.inc из предыдущего раздела, укажите его текущее имя.

3. Создайте условный оператор, определяющий, надо ли выводить или обрабатывать форму:

```
if (isset($HTTP_POST_
⇨ VARS['submit'])) {
```

Листинг 6.6. Последний в этой главе сценарий реализует улучшенный контроль безопасности за счет применения функции `mysql_escape_string()`. Кроме того, здесь демонстрируется выполнение команды `UPDATE` (продолжение)

Листинг

```
            $query .= "expense_amount='" .
             ⇨ mysql_escape_string($HTTP_POST_VARS['expense_amount']) .
            ⇨ "', expense_description='" .
             ⇨ mysql_escape_string($HTTP_POST_VARS['expense_description']) .
            ⇨ "', expense_date='" .
             ⇨ mysql_escape_string($HTTP_POST_VARS['expense_date']) .
             ⇨ "' WHERE expense_id={$HTTP_POST_VARS['expense_id']}'";

        // Были обнаружены ошибки?
        if (!$problem) {

      // Выполнить запрос:
            $query_result = mysql_query ($query);

            // Напечатать сообщение об успешном завершении операции
           // или ошибке:
            if (mysql_affected_rows() == 1) {
     echo '<b><font color="green">The expense has been edited!</font></b>';
     } else {
     echo '<b><font color="red">The expense was not edited!</font></b>';
     }
       } else { // Если была ошибка:
     echo '<b><font color="red">The expense was not edited because
     ⇨ the expense category could not be added!</font></b>';
       }
    } else { // Напечатать сообщение о том, что категория не задана:
```

Листинг 6.6. Последний в этой главе сценарий реализует улучшенный контроль безопасности за счет применения функции `mysql_escape_string()`. Кроме того, здесь демонстрируется выполнение команды `UPDATE` (продолжение)

Листинг

```
        echo '<b><font color="red">You missed a required field!</font></b>';
    }

} else { // Если форма не была отправлена, вывести ее.

   // Создать запрос:
   $query = "SELECT * FROM expenses WHERE expense_id = {$HTTP_GET_VARS['eid']} LIMIT 1";

   // Выполнить запрос:
   $query_result = @mysql_query ($query);

   // Извлечь и напечатать результаты:
   $row = @mysql_fetch_array ($query_result, MYSQL_ASSOC);
   @mysql_free_result($query_result);

   echo 'Edit this expense:<br />
   <form action="edit_expense.php" method="post">
   <ul>
   <li>Expense Category: <select name="expense_category_id">';

   // Вывести категории затрат:
   $query_result2 = mysql_query ('SELECT * FROM expense_categories ORDER BY
   ⇨ expense_category');
   while ($row2 = mysql_fetch_array ($query_result2, MYSQL_NUM)) {
      if ($row2[0] == $row['expense_category_id']) {
         echo "<option value=\"$row2[0]\" selected=\"selected\">$row2[1]</option>\n";
     } else {
         echo "<option value=\"$row2[0]\">$row2[1]</option>\n";
     }
   }

   @mysql_free_result($query_result2);
   mysql_close();

   // Закончить вывод формы:
   echo '</select></li>
   or<br />
   <li>Enter a new expense category: <input type="text"
   name="expense_category" size="30" maxlength="30" /></li>
   </ul>
   Expense Amount: <input type="text" name="expense_amount" value="' .
   ⇨ $row['expense_amount'] . '" size="10" maxlength="10" /><p />
   Expense Date: <input type="text" name="expense_date" value="' .
   ⇨ $row['expense_date'] . '" size="10" maxlength="10" /><p />
```

4. Проверьте, заполнены ли обязательные поля:

```
if ( (isset($HTTP_POST_VARS
⇨ ['expense_category_id']) OR
⇨ (strlen($HTTP_POST_VARS['expense_
⇨ category']) > 0)) AND
⇨ (strlen($HTTP_POST_VARS['expense_
⇨ amount']) > 0) AND
⇨ (strlen($HTTP_POST_ VARS['expense_
⇨ description']) > 0) ) {
```

Этот код аналогичен написанному в сценарии add_expense.php за одним исключением: здесь требуется выполнение хотя бы одного из условий `isset ($HTTP_POST_VARS['expense_category_id'])` или `strlen($HTTP_POST_VARS ['expense_category']) > 0`). Если категория затрат выбрана из списка, то истинно первое условие, а если введена новая – то второе.

5. Начните формирование запроса:

```
$query = "UPDATE expenses SET ";
```

Листинг 6.6. Последний в этой главе сценарий реализует улучшенный контроль безопасности за счет применения функции `mysql_escape_string()`. Кроме того, здесь демонстрируется выполнение команды `UPDATE` (окончание)

Листинг

```
   Expense Description: <textarea name="expense_description" rows="5" cols="40">' .
   ⇨ stripslashes($row['expense_description']) . '</textarea><p />
   <input type="submit" name="submit" value="Submit!" />
   <input type="hidden" name="expense_id" value="' . $HTTP_GET_VARS['eid'] . '" />
   </form>';

} // Конец главного условного оператора.

?>
</body>
</html>
```

Поскольку форма предназначена для редактирования существующей записи, то запрос начинается с команды UPDATE, а не INSERT. Остаток этой части сценария (и вся следующая) не содержат ничего нового.

6. Проверьте, введено ли название новой категории затрат, и действуйте по ситуации:

```
if (strlen($HTTP_POST_VARS
⇨ ['expense_category']) > 0) {

$query2 = "INSERT INTO expense_
⇨ categories VALUES (NULL,
⇨ '{$HTTP_POST_VARS['expense_
⇨ category']}')";

if (mysql_query ($query2)) {

echo '<b><font color="green">The
⇨ expense category has been
⇨ added!</font></b><br />';

$query .= "expense_category_id=" .
⇨ mysql_insert_id() . "; ";

} else {
echo '<b><font color="red">The
⇨ expense category was not
⇨ entered into the table!</font>
⇨ </b><br />';

$problem = TRUE;

}

} else { // Конец проверки ввода
⇨ новой категории.

$query .= "expense_category_id=
⇨ {$HTTP_POST_VARS
⇨ ['expense_category_id']}, ";
}
```

`The expense category has been added!` — «Категория затрат была добавлена».

`The expense category was not entered into the table!` — «Категория затрат не была введена в таблицу».

Одно из отличий от предыдущего сценария заключается в том, что запись предложения вставки – INSERT INTO *имя_таблицы* VALUES('*значение1*', '*значение2*' . . .), а предложения обновления – UPDATE *имя_таблицы* SET *имя_колонки1*='*значение1*', *имя_колонки2*='*значение2*', . . . В этом фрагменте изменяется значение поля expense_category_id в таблице expenses.

7. Завершите формирование запроса:

```
$query .= "expense_amount='" .
⇨ mysql_escape_string($HTTP_POST_
⇨ VARS['expense_amount']) .
⇨ "', expense_description='" .
⇨ mysql_escape_string($HTTP_POST_VARS
⇨ ['expense_description']) .
⇨ "', expense_date='" .
⇨ mysql_escape_string($HTTP_POST_VARS
⇨ ['expense_date']) . "' WHERE
⇨ expense_id={$HTTP_POST_VARS
⇨ ['expense_id']}";
```

8. Выполните запрос:

```
if (!$problem) {

$query_result = mysql_query
⇨ ($query);

if (mysql_affected_rows() == 1) {

echo '<b><font color="green">The
⇨ expense has been edited!
⇨ </font></b>';

} else {

echo '<b><font color="red">The
⇨ expense was not edited!</font>
⇨ </b>';

}
```

The expense has been edited! – «Запись о затрате была отредактирована». The expense was not edited! – «Запись о затрате не была отредактирована».

В этом примере я решил воспользоваться функцией mysql_affected_rows(), которая возвращает число строк, обновленных предыдущим запросом. Эта функция полезна при выполнении запросов типа UPDATE, ALTER, DELETE и INSERT.

9. Завершите условные операторы:

```
} else {
echo '<b><font color="red">The
⇨ expense was not edited because
⇨ the expense category could not
⇨ be added!</font></b>';
}
} else {
echo '<b><font color="red">You
⇨ missed a required field!</font>
⇨ </b>';
}
```

The expense was not entered into the table because the expense_category could not be added – «Запись о затрате не была введена в таблицу, поскольку невозможно добавить категорию затрат».

You missed a required field – «Вы пропустили поле, обязательное для заполнения».

На этом заканчивается часть сценария, посвященная обработке формы и обновлению базы данных. В оставшейся части выводится форма для редактирования записи о затратах.

10. Выберите запись из базы:

```
$query = "SELECT * FROM expenses
⇨ WHERE expense_id =
⇨ {$HTTP_GET_VARS['eid']} LIMIT 1";
$query_result = @mysql_query
⇨ ($query);
$row = @mysql_fetch_array
⇨ ($query_result, MYSQL_ASSOC);
@mysql_free_result($query_result);
```

Для того чтобы отредактировать запись, надо сначала выбрать ее из базы данных. Лучше всего обратиться к ней по первичному ключу (expense_id). В данном сценарии предполагается, что значение ключа передается в поле формы eid. Поскольку я извлекаю только одну запись, нет необходимости в цикле `while`. По завершении запроса ассоциативный массив `$row` будет содержать всю информацию из записи о затратах.

11. Выведите HTML-форму:

```
echo 'Edit this expense: <br />

<form action="edit_expense.php"
⇨ method="post">

<ul>

<li>Expense Category: <select name=
⇨ "expense_category_id">';
```

`Edit this expense` — «Редактировать запись о затрате».

12. Создайте выпадающий список:

```
$query_result2 = mysql_query
⇨ ('SELECT * FROM expense_categories
⇨ ORDER BY expense_category');

while ($row2 = mysql_fetch_array
⇨ ($query_result2, MYSQL_NUM)) {

if ($row2[0] == $row['expense_
⇨ category_id']) {

echo "<option value=\"$row2[0]\"
⇨ selected=\"selected\">$row2[1]</
⇨ option>\n";

} else {

echo "<option value=\"$row2[0]\"
⇨ >$row2[1]</option>\n";

}

}

@mysql_free_result($query_result2);

mysql_close();
```

Этот фрагмент кода несколько сложнее, чем в сценарии add_expense.php, так как требуется выделить в списке категорию, указанную в записи о затратах. Для этого подойдет обычный условный оператор, который добавляет атрибут `selected` в тэг `<option>`.

13. Завершите HTML-форму:

```
echo '</select></li>

or<br />

<li>Enter a new expense category:
⇨ <input type="text" name=
⇨ "expense_category" size="30"
⇨ maxlength="30" /></li>

</ul>

Expense Amount: <input type="text"
⇨ name="expense_amount" value="' .
⇨ $row['expense_amount'] . '"
⇨ size="10" maxlength="10" /><p />

Expense Date: <input type="text"
⇨ name="expense_date" value="' .
⇨ $row['expense_date'] . '" size=
⇨ "10" maxlength="10" /><p />

Expense Description: <textarea
⇨ name="expense_description"
⇨ rows="5" cols="40">' .
⇨ stripslashes($row['expense_
⇨ description']) . '</textarea><p />

<input type="submit" name="submit"
⇨ value="Submit!" />

<input type="hidden" name=
⇨ "expense_id" value="' .

$HTTP_GET_VARS['eid'] . '" />

</form>';

}
```

`Enter a new expense category` – «Добавить новую категорию затрат».

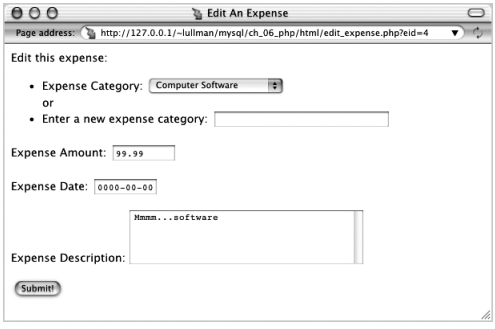

Рис. 6.15. Первоначально в записи не была задана дата. Теперь это упущение можно исправить с помощью формы редактирования

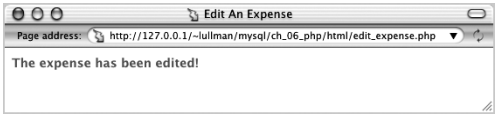

Рис. 6.16. Как и во всех сценариях, представленных в этой главе, об успешном завершении работы информирует стандартное сообщение

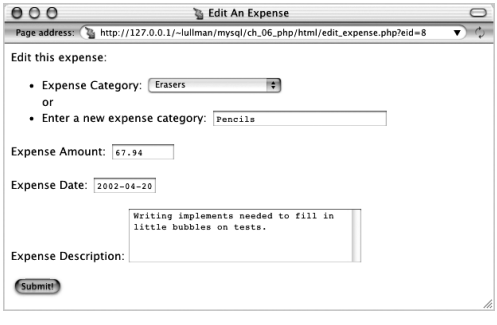

Рис. 6.17. Сценарий edit_expense.php, как и add_expense.php, позволяет сразу ввести новую категорию затрат

Для полей ввода из HTML-формы можно указать начальное значение в атрибуте `value` тэга `<input>`. Начальное значение областей ввода указывается между тэгами `<textarea>` и `</textarea>`. Первичный ключ записи следует сохранить в скрытом поле, чтобы сценарий «знал», какую именно запись обновлять.

14. Завершите сценарий и html-страницу:

```
?>
</body>
</html>
```

15. Сохраните сценарий в файле edit_expense.php, загрузите его на Web-сервер и протестируйте в браузере (рис. 6.15–6.17).

Для тестирования сценария необходимо дописать в конец URL строку вида `?eid=`*x*, где *x* – значение поля expense_id в записи таблицы expenses, подлежащей редактированию.

П Функция `mysql_real_escape_string()` производит экранирование с учетом языка – это еще одно преимущество по сравнению с `addslashes()`.

С То, что вы узнали в последних двух разделах, можно применить и к сценариям, написанным ранее, дабы улучшить контроль данных и обработку ошибок.

С С сайта книги вы можете загрузить сценарий browse_expenses.php, позволяющий просмотреть все записи о затратах и вызвать сценарий edit_expense.php для редактирования любой из них.

7

MySQL и Perl

Задолго до того как язык PHP стал играть ведущую роль в разработке Web-приложений, программисты уже использовали язык Perl (Practical Extraction and Reporting Language – язык построения и печати отчетов) как для решения административных задач, так и для Web-программирования. Perl разработал Ларри Уолл (Larry Wall) примерно 20 лет назад, и с тех пор этот язык приобрел большую популярность как средство написания CGI-сценариев (Common Gateway Interface – единый шлюзовой интерфейс) для динамических HTML-страниц.

Материал этой главы рассчитан на различные категории читателей. Первая из них – пользователи, владеющие языком Perl, но до сих пор не имевшие дела с MySQL. Вторая – пользователи Perl, уже работавшие с MySQL и желающие углубить свои знания. Наконец, данную главу могут изучить разработчики, не владеющие Perl, но стремящиеся узнать, как можно организовать взаимодействие MySQL и Perl (особенно если сравнить этот язык с PHP или Java).

Perl работает практически в любой операционной системе, включая UNIX, Windows и Macintosh. Хотя на Perl можно писать CGI-сценарии, все примеры из этой главы – обычные программы, не предназначенные для исполнения Web-сервером. Я покажу, как инсталлировать дополнительные модули Perl для поддержки MySQL, предполагая, что на вашем компьютере установлена версия не ниже 5.004.

Рис. 7.1. Добрые люди из компании ActiveState (www.activestate.com) поддерживают ActivePerl – наиболее распространенный дистрибутив Perl для Windows

Установка Perl с поддержкой MySQL на платформу Windows

Хотя изначально язык Perl создавался без учета Windows, программирование в этой среде на Perl не вызывает никаких трудностей благодаря продукту ActivePerl, который бесплатно распространяется компанией ActiveState на сайте www.activestate.com (рис. 7.1). ActivePerl – это полнофункциональный дистрибутив Perl, который легко устанавливается на любую платформу Windows.

Помимо Perl для взаимодействия с MySQL вам понадобятся модули DBI и DBD. В состав дистрибутива ActivePerl входит менеджер пакетов программ (Programmer's Package Manager – PPM), который упрощает задачу установки модулей из архива ресурсов, относящихся к Perl (Comprehensive Perl Archive Network – CPAN). Единственное требование для инсталляции всех нижеописываемых продуктов – наличие соединения с Internet.

Порядок установки Perl с поддержкой MySQL на платформу Windows

1. Загрузите последнюю версию ActivePerl с сайта www.activestate.com.

   При установке ActivePerl на Windows 2000 или XP нужно загрузить только пакет для инсталлятора Windows Installer (файл с расширением .msi). Что касается других ОС, мастер загрузки перечислит относящиеся к ним требования. На момент написания этой книги последней была версия ActivePerl 5.6.1.

2. Запустите загруженную программу, дважды щелкнув по ее иконке мышью (рис. 7.2).

   Программа установки предложит выполнить несколько шагов, подсказывая в некоторых местах, что делать дальше. Если не хотите лишней головной боли, то лучше всего согласитесь с настройками по умолчанию.

   На этом инсталляция ActivePerl в вашей системе завершена. Теперь осталось установить модули для работы с базой данных.

3. Запустите сеанс DOS.

   Это можно сделать разными способами – в частности, щелкнуть по меню **Start** (Пуск), выбрать пункт **Run** (Выполнить) и ввести строку `cmd` в появившейся форме, после чего нажать клавишу **Enter** (**Return**).

4. Проверьте, успешно ли был установлен Perl, для чего введите команду

   ```
   perl -v
   ```

   и нажмите **Enter** (**Return**) – рис. 7.3.

   Вашему вниманию будут представлены версия Perl и ряд других сведений.

Рис. 7.2. Мастер установки ActivePerl будет направлять вас в процессе установки Perl

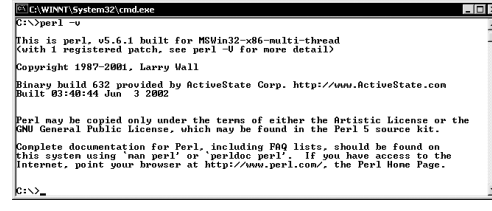

Рис. 7.3. Если компонент ActivePerl был установлен нормально, после ввода команды `perl -v` вы должны увидеть такое сообщение

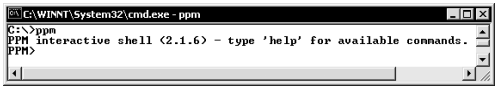

Рис. 7.4. Менеджер пакетов PPM упрощает установку дополнительных модулей Perl

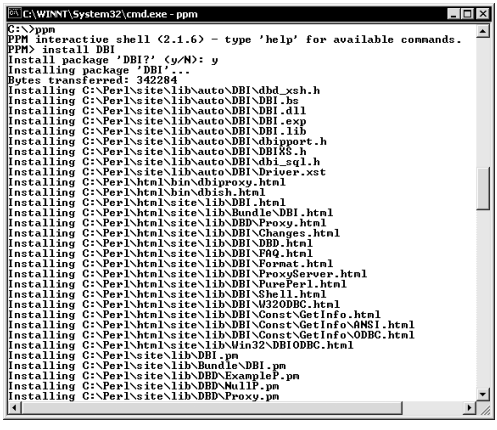

Рис. 7.5. Пользуясь PPM, установите сначала модуль DBI, реализующий общую инфраструктуру для доступа к базам данных

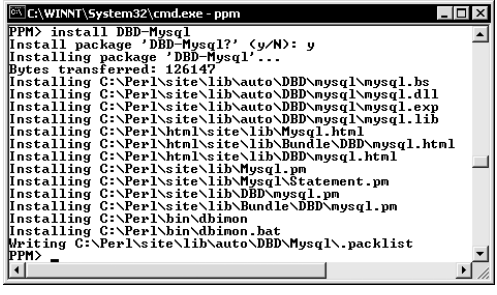

Рис. 7.6. Чтобы из Perl-сценария можно было обращаться к базе данных MySQL, необходимо установить модуль DBD-MySQL

5. В ответ на приглашение системы введите команду `ppm` (рис. 7.4).

   Эта команда запускает менеджер пакетов, входящий в состав ActivePerl. Вы должны увидеть приглашение `PPM>`.

6. Установите модуль DBI (рис. 7.5):

   ```
   install DBI
   ```

   Получив подтверждение команды инсталлировать пакет, PPM загрузит и установит все необходимые файлы.

7. Установите модуль для работы с MySQL (рис. 7.6):

   ```
   install DBD-MySQL
   ```

   После инсталляции пакета DBI необходимо установить модули для конкретных баз данных; DBD-MySQL – один из них. В Perl-сценариях для соединения с базой данных используется комбинация модулей DBI и DBD.

8. Выйдите из PPM, введя команду `quit` и нажав клавишу **Enter** (**Return**).

П Имеются также версии ActivePerl для операционных систем Linux и Solaris, хотя чаще всего этот продукт используется именно на платформах Windows.

С Если вам нужна дополнительная информация о Perl и CPAN, посетите соответственно сайты www.perl.com и www.cpan.org.

С Если вы намереваетесь писать на Perl CGI-сценарии, выполните в PPM команду `install CGI`[1].

С Находясь в PPM, вы можете узнать, какие модули установлены в вашей системе: для этого достаточно выполнить команду `query` (после действий, описанных в п. 4).

[1] Модуль CGI.pm уже входит в состав дистрибутива ActivePerl, поэтому устанавливать его нет необходимости. – *Прим. переводчика.*

Установка Perl с поддержкой MySQL на платформы UNIX и Mac OS X

Подобно PHP, язык Perl уже предустановлен в большинстве UNIX-подобных систем, включая Mac OS X. Кроме Perl для доступа к MySQL понадобится несколько дополнительных модулей, в первую очередь DBI и DBD-MySQL. Текущие версии этих модулей, а также модуля Data-Dumper можно найти на странице www.mysql.com/downloads/api-dbi.html.

Есть два варианта установки:

- воспользоваться CPAN;
- собрать и установить модули самостоятельно.

Первый способ гораздо проще и хорошо знаком всем пользователям Perl. Поэтому ниже я покажу, как собирать и инсталлировать модули вручную. Все инструкции предполагают, что на вашем компьютере установлены клиентские библиотеки MySQL и, конечно же, сам Perl.

Порядок установки Perl с поддержкой MySQL

1. Войдите в систему как пользователь root:

   ```
   su root
   ```

   Для выполнения нижеописанных действий требуются полномочия администратора.

2. Загрузите с вышеупомянутой страницы три необходимых модуля: Data-Dumper, DBI и Msql-Mysql.

   На сайте MySQL находятся последние версии файлов, которые также можно загрузить с сайта www.cpan.org.

Perl на платформе Mac OS X

Perl, как и PHP, входит в комплектацию Mac OS X и не требует предварительной активации. Поскольку для самостоятельного построения и установки модулей требуются определенный опыт и дополнительные инструменты, находящиеся на диске Mac OS X Developer's CD-ROM, я рекомендую начинающим программистам на этой платформе взять уже собранные версии из архива CPAN.

Архив CPAN создавался как упорядоченная система для решения определенного вида задач. В числе прочего CPAN содержит средства установки требуемых вам модулей (иными словами, способен обновляться).

Для доступа к CPAN запустите приложение Terminal и введите следующую команду:

```
perl -MCPAN -e shell
```

При первом заходе на CPAN вам придется ответить на несколько вопросов, большинство которых касается установок по умолчанию. Открыв CPAN, установите несколько пакетов, в том числе Bundle::libnet, DBI и DBD::mysql. Для инсталляции любого пакета наберите команду `install` *`имя_пакета`*.

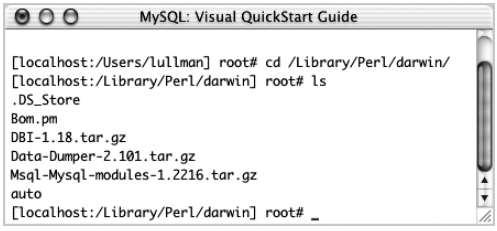

Рис. 7.7. Установка дополнительных модулей в стандартный каталог Perl (он может зависеть от вашей операционной системы)

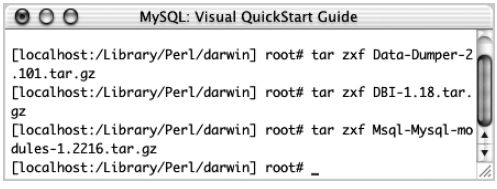

Рис. 7.8. Прежде чем начинать сборку модуля, нужно развернуть дистрибутив с помощью команды `tar`

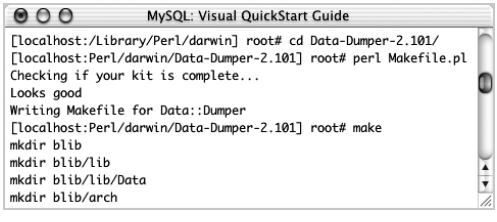

Рис. 7.9. Первые два этапа сборки любого модуля – выполнение сценария Makefile.pl и команды `make`

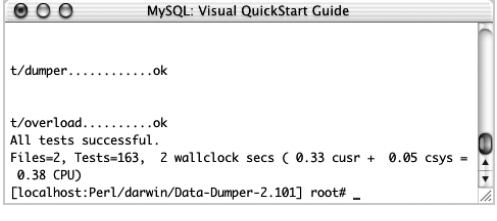

Рис. 7.10. Если команда `make test` завершилась успешно, можно запускать `make install`

3. Переместите загруженные файлы в подходящий каталог (рис. 7.7).

   Можете либо воспользоваться командой `mv`, либо в графическом файловом менеджере перетащить файлы в выбранный каталог (зависящий от того, как установлен язык Perl на вашей машине). Я отвел для этой цели папку /Library/Perl/darwin.

4. Распакуйте все три файла (рис. 7.8):

   ```
   tar zxf Data-Dumper-2.101.tar.gz
   tar zxf DBI-1.18.tar.gz
   tar zxf Msql-Mysql-modules-
   ⇨ 1.2216.tar.gz
   ```

   Команда `tar` развернет каждый файл в отдельный каталог. При необходимости измените имена сжатых файлов, если вы загрузили другие версии.

5. Войдите в каталог Data-Dumper:

   ```
   cd Data-Dumper-2.101
   ```

6. Выполните сценарий Makefile.pl (см. рис. 7.9):

   ```
   perl Makefile.pl
   make
   ```

   Для сборки любого стандартного модуля нужно сначала выполнить сценарий Makefile.pl, а затем запустить команду `make`.

7. Выполните команды тестирования и установки модуля (рис. 7.10):

   ```
   make test
   make install
   ```

   Внимательно следите за ходом выполнения команды `make test`. Если при тестировании не выявлено никаких ошибок или обнаружены только несущественные, вы можете переходить к установке модуля. В противном случае

сообщения об ошибках должны подсказать вам, что необходимо изменить.

Начинающему пользователю следует иметь в виду, что тесты разрабатывались для разных платформ, поэтому некоторые из них могут завершаться с ошибкой при выполнении в системе, на которую не рассчитаны. Это нормально и не является причиной для паники.

8. Повторите процедуру для установки модуля DBI.

Успешно собрав модуль Data-Dumper, выполните действия, описанные в пп. 5–7, для инсталляции DBI.

9. Еще раз повторите те же операции для установки модуля Msql-Mysql (рис. 7.11).

Обратите внимание, что сценарий Makefile.pl для этого модуля более сложен – он требует ответа на несколько вопросов, в том числе:

- какой драйвер устанавливать (для mSQL, MySQL или оба);
- где находятся библиотеки MySQL (или mSQL);
- каково имя тестовой базы данных;
- каковы имя хоста, имя и пароль пользователя.

Сценарий предлагает варианты по умолчанию для большинства вопросов, но вы должны быть уверены в правильности своих ответов, особенно если установка MySQL производится нестандартным образом. В самом конце сценарий попытается установить соединение с базой данных, чтобы проверить возможность доступа из Perl к MySQL.

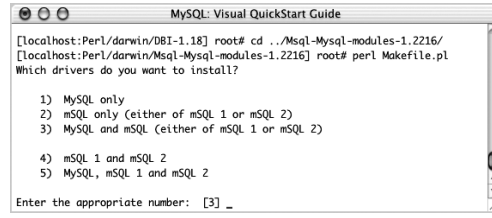

Рис. 7.11. Сценарием Makefile.pl, входящим в состав дистрибутива Msql-Mysql, предусмотрен ряд вопросов, касающихся конфигурации

С Чтобы проверить, не включена ли уже в вашу установку Perl поддержка MySQL, наберите команду `perldoc DBD::mysql`.

П В поставку Red Hat Linux уже включен модуль DBI для Perl. Установить драйвер для MySQL можно из RPM-пакета или из CPAN.

С На сайте MySQL имеется список рассылки, специально посвященный модулю Msql-Mysql. Если у вас возникнут проблемы, можете поискать решение там. Но, прежде чем отправлять новое сообщение, познакомьтесь с архивами.

Листинг 7.1. Этот простой Perl-сценарий проверяет наличие драйвера MySQL

Листинг

```
#!/usr/bin/perl

# Листинг 7.1, 'test.pl'

# Включить используемые модули.
use strict;
use DBI;

# Создать массив @drivers.
my @drivers = DBI->available_drivers();

# Напечатать имя каждого драйвера.
foreach (@drivers) {
    print $_ . "\n";
}
```

Тестирование Perl и MySQL

Прежде чем подробно объяснять, как устанавливается соединение с MySQL из Perl-сценария, я хочу предложить вам одно небольшое упражнение. Рассматриваемый ниже сценарий иллюстрирует мой стиль написания программ на Perl в этой главе. Кроме того, с его помощью мы проверим, что поддержка MySQL установлена.

Хотя эту главу ни в коей мере нельзя считать заменой полноценному учебнику по языку Perl, я все же вкратце расскажу об основных частях простого Perl-сценария. Даже программисты, ранее работавшие на других языках (к примеру, PHP), смогут, следуя этим инструкциям, писать не слишком сложные программы на Perl. Кроме того, в следующих разделах предполагается, что вы владеете нижеприведенной информацией, и потому она не будет комментироваться повторно.

Написание простого Perl-сценария

1. Откройте в текстовом редакторе новый документ (листинг 7.1).

   Поскольку Perl-сценарии – это обычные текстовые файлы, выбор редактора не имеет значения.

2. Включите строку, начинающуюся с символов `#!` (для пакета ActivePerl на платформе Windows это лишнее):

   ```
   #!/usr/bin/perl
   ```

   Эта строка говорит операционной системе, что для интерпретации файла нужен язык Perl. После символов `#!` должен быть указан полный путь к интерпретатору `perl` на вашем компьютере. Имейте в виду, что помимо каталога /usr/bin Perl часто устанавливается в каталог /usr/local/bin. При использовании пакета ActivePerl на платформе Windows эту строку можно вообще опустить.

3. Задайте режим строгой проверки:

```
use strict;
```

Команда `use strict` включает режим строгой проверки, который почти не замедляет работу сценария. Я буду использовать ее всюду в этой главе. Основное последствие этой директивы – необходимость объявлять переменные до их использования (см. п. 5).

Для программистов, незнакомых с Perl, отмечу, что в этом языке каждая строка, кроме строки `#!` и управляющих конструкций, должна завершаться точкой с запятой. Однострочные комментарии начинаются со знака # и продолжаются до конца строки.

4. Включите модуль DBI:

```
use DBI;
```

Эта строка будет присутствовать во всех сценариях из данной главы, поскольку сообщает сценарию о необходимости включить модуль DBI, необходимый для взаимодействия с MySQL.

5. Объявите и инициализируйте переменную:

```
my @drivers = DBI->available_drivers();
```

Массив `@drivers` будет содержать все драйверы баз данных в этой установке Perl, о которых «знает» модуль DBI. Их находит метод `available_drivers()` класса DBI. Слово `my` необходимо в режиме `strict`, иначе компилятор выдаст ошибку.

6. Переберите весь массив и напечатайте имя каждого драйвера:

```
foreach (@drivers) {
  print $_ . "\n";
}
```

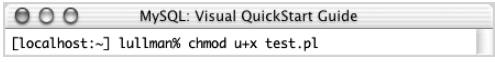

Рис. 7.12. Чтобы сделать файл исполняемым (то есть для его запуска достаточно будет ввести команду `./имя_файла.pl`), нужно изменить права доступа к нему

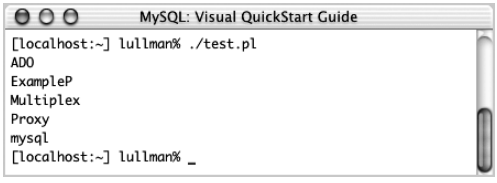

Рис. 7.13. После запуска сценария на экране появляется перечень драйверов DBI, которые теперь можно использовать при работе с Perl

Эта конструкция обходит все элементы массива `@drivers`. Внутри цикла на текущий элемент ссылается специальная переменная `$_`. Оператор `print` распечатывает ее значение, а затем выводит символ перехода на новую строку (`\n`).

7. Сохраните сценарий в файле test.pl.

   В этой главе я буду использовать для Perl-сценариев расширение .pl.

8. Измените права доступа к файлу из командной строки (рис. 7.12):

   ```
   chmod u+x test.pl;
   ```

   В операционных системах UNIX и Mac OS X необходимо сообщить ОС о том, что файл должен быть исполняемым: введя команду `chmod`, вы добавляете право на исполнение (`x`) для всех пользователей (`u`). В системе Windows этого не требуется.

9. Запустите файл на исполнение.

   В Windows:

   – введите команду `perl C:\путь\к\test.pl` в сеансе DOS и нажмите клавишу **Enter** (**Return**)

     или

   – дважды щелкните по названию файла test.pl в программе Windows Explorer (впрочем, этот метод я не рекомендую).

   В системах UNIX и Mac OS X:

   – введите команду `./test.pl` и нажмите клавишу **Enter** (**Return**) – рис. 7.13

     или

   – введите команду `perl /путь/к/test.pl` и нажмите **Enter** (**Return**).

Имейте в виду: чтобы первый способ сработал, сценарий должен начинаться со строки `#!`, а файл – быть исполняемым. Для второго способа не требуется ни то, ни другое.

Каким бы образом вы ни запустили сценарий, в списке имен драйверов должна присутствовать строка `mysql` (рис. 7.13). Она подтверждает возможность подключиться к MySQL из Perl.

C Чтобы проверить правильность синтаксиса, не выполняя сценарий, введите команду `perl -c /путь/к/файлу.pl`.

C Если вы собираетесь использовать Perl для разработки Web-приложений, то знайте, что для сервера Apache есть модуль mod_perl[1]. Если вы будете работать именно с ним, то следует включать модуль Apache::DBI, а не просто DBI. Также вам понадобится модуль CGI.

[1] Использовать модуль mod-perl для разработки Web-приложений на Perl не обязательно, хотя он намного увеличивает скорость выполнения сценария за счет того, что сохраняет в памяти его откомпилированное представление, а не компилирует при каждом обращении заново (и это не единственное его достоинство). Того же эффекта можно добиться, используя модуль Apache mod-fcgi. – *Прим. переводчика.*

Обработка ошибок в Perl

Если задан параметр `RaiseError`, то при возникновении ошибки в MySQL драйвер вызовет функцию `die()`, которая печатает сообщение и немедленно завершает сценарий[1]. Если же задан параметр `PrintError`, то сообщение печатается, но сценарий продолжает работу. Для использования режима `PrintError` следует написать такое предложение:

```
$dbh = DBI->connect('DBI:mysql:
⇨ имя_базы_данных:имя_хоста',
⇨ 'имя_пользователя', 'пароль',
⇨ {RaiseError => 1});
```

Альтернативный способ – применение конструкции `or exit()`. Сценарий при этом прекращает работу, но у вас остается возможность самостоятельной обработки ошибок.

Метод `DBI->err()` возвращает код ошибки, а метод `DBI->errstr()` – ее словесное описание. Оба метода ссылаются на ошибку, обнаруженную при самом последнем обращении к базе.

[1] На самом деле функция `die()` не завершает программу, а «поднимает» исключение, которое сценарий может перехватить и обработать с помощью оператора `eval`. В режиме же `PrintError` возникновение ошибки индицируется кодом возврата, который необходимо проверять после каждого вызова метода, обращающегося к базе данных. – *Прим. переводчика.*

Соединение с MySQL

Когда вы убедитесь, что сможете заставить Perl-сценарий работать и что модуль DBI для MySQL установлен, самое время приступить к написанию сценариев для доступа к базе данных. Прежде всего необходимо установить соединение с сервером:

```
$dbh = DBI->connect('DBI:mysql:
⇨ имя_базы_данных:имя_хоста',
⇨ 'имя_пользователя', 'пароль',
⇨ {RaiseError => 1});
```

Первый аргумент – `DBI:mysql:`*имя_базы_данных*`:`*имя_хоста* – обычно называют *именем источника данных* и иногда присваивают переменной `$dsn`, которую и передают методу `connect`. Значения *имя_пользователя* и *пароль* – это имя и пароль пользователя MySQL, зарегистрированного в базе данных mysql. Наконец, фрагмент `{RaiseError => 1}` определяет способ извещения об ошибках (см. врезку «Обработка ошибок»). В модуле DBI реализована мощная система диагностики ошибок, и такой способ обеспечивает наилучший способ обращения к ней. Результат выполнения метода `DBI->connect()` присваивается переменной `$dbh` (описателю базы данных), которая будет использоваться в остальных частях сценария. Обратите внимание, что в Perl имя базы данных указывают при соединении с сервером MySQL.

По завершении работы с базой данных необходимо вызвать метод `disconnect()` для разрыва установленного соединения:

```
$dbh->disconnect();
```

В первом Perl-сценарии я воспользуюсь вышеописанными приемами для того, чтобы просто установить и разорвать соединение с базой данных. После того как сценарий заработает, мы сможем приступить к исполнению запросов.

Подключение к MySQL

1. Откройте в текстовом редакторе новый документ.
2. Введите строку `#!` (листинг 7.2):

```
#!/usr/bin/perl
```

3. Включите модуль DBI и установите режим строгой проверки:

```
use DBI;
use strict;
```

4. Установите соединение с MySQL:

```
my $dbh = DBI->connect('DBI:mysql:
⇨ accounting', 'имя_пользователя',
⇨ 'пароль', {RaiseError => 1});
```

   В этом примере я воспользуюсь базой данных accounting, разработанной ранее. Замените *имя_пользователя* и *пароль* именем и паролем пользователя, имеющего полномочия для доступа к базе на вашем сервере.

5. Введите сообщение об успешном соединении:

```
if ($dbh) {
    print "Successfully connected to
    ⇨ the database! \n";
}
```

Листинг 7.2. Этот сценарий проверяет, возможно ли установить соединение с MySQL

Листинг

```
#!/usr/bin/perl

# Листинг 7.2, 'mysql_connect.pl'

# Включить используемые модули:
use DBI;
use strict;

# Установить соединение с базой данных:
my $dbh = DBI->connect('DBI:mysql:
⇨ accounting', 'имя_пользователя',
⇨ 'пароль',
             {RaiseError => 1});

# Сообщить об удавшейся попытке
# установить соединение:
if ($dbh) {
    print "Successfully connected to the
    ⇨ database! \n";
}

# Разорвать соединение:
$dbh->disconnect;
```

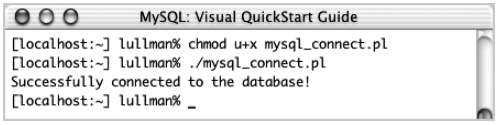

Рис. 7.14. Если у вас есть надлежащие полномочия на доступ к базе, вы увидите такое сообщение

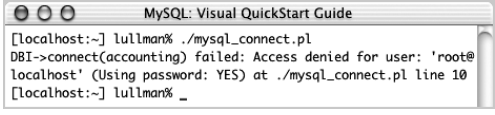

Рис. 7.15. Если полномочий недостаточно, то появится сообщение об ошибке, и сценарий аварийно завершит работу

Переменная `$dbh` содержит ссылку на объект, представляющий подключение к базе данных. Ее можно проверить, чтобы узнать, было ли установлено соединение. Поскольку был выбран режим `RaiseError`, в случае неудачи сценарий будет завершен, так что выводить сообщение о невозможности соединиться не нужно.

6. Закройте соединение:

```
$dbh->disconnect;
```

Это важный шаг, поскольку в данный момент освобождаются ресурсы, захваченные MySQL и Perl.

7. Сохраните сценарий в файле mysql_connect.pl, при необходимости измените права доступа к нему и запустите сценарий (рис. 7.14, 7.15).

Исходя из предположения, что ранее написанная программа test.pl завершилась успешно, результат работы данного сценария зависит исключительно от наличия или отсутствия полномочий указанного пользователя на доступ к базе. Проверьте это в первую очередь, если выдается ошибка.

П В Perl нельзя устанавливать постоянные соединения с MySQL (в отличие от PHP), если только сценарий не работает под управлением Apache-модуля mod-perl.

С В имени источника данных (Data Source Name, DSN) слово `mysql` должно быть записано строчными буквами, а имя хоста можно опускать, если оно совпадает с `localhost`.

Простые запросы

Теперь вы знаете, как соединиться с MySQL, и можете заняться выполнением запросов к базе данных. Есть два вида запросов: возвращающие записи (то есть запросы `SELECT`) и все остальные (`ALTER`, `CREATE`, `UPDATE` и `DELETE`). Запросы первого типа сложнее, поэтому их рассмотрение мы отложим до следующего раздела. А сейчас я покажу, как выполнять простые запросы с помощью метода `do()`:

```
$query = $dbh->do("запрос");
```

Метод `do()` принадлежит объекту `$dbh`, представляющему базу данных и созданному в результате установления соединения. Если не было ошибки, метод возвращает число строк, затронутых при выполнении запроса (в тех случаях, когда это имеет смысл).

Для примера я напишу Perl-сценарий, который будет добавлять новых пользователей в базу accounting. Напомню, что в главе 5 описывалось включение в эту базу таблицы users; при заполнении ее полей используются функции `PASSWORD()` и `ENCODE()`.

Выполнение простых запросов

1. Откройте в текстовом редакторе новый файл и введите стандартный пролог Perl-сценария (листинг 7.3):

   ```
   #!/usr/bin/perl
   use DBI;
   use strict;
   ```

2. Запросите имя нового пользователя:

   ```
   print "Enter the new username: ";
   my $username = <STDIN>;
   ```

   `Enter the new username` — «Введите имя нового пользователя».

Листинг 7.3. Сценарий add_user.pl добавляет в базу данных новые записи

Листинг

```
#!/usr/bin/perl

# Листинг 7.3, 'add_user.pl'

# Включить используемые модули.
use DBI;
use strict;

# Запросить информацию о пользователе.
print "Enter the new username: ";
my $username = <STDIN>;
print "Enter the password: ";
my $pass1 = <STDIN>;
print "Confirm the password: ";
my $pass2 = <STDIN>;

# Проверить, что оба введенных пароля совпадают.
while ($pass1 ne $pass2) {
  print "The passwords you entered did not match! Try again!\n";
  print "Enter the password: ";
  $pass1 = <STDIN>;
  print "Confirm the password: ";
  $pass2 = <STDIN>;
}

# Соединиться с базой данных.
my $dbh = DBI->connect('DBI:mysql:accounting', 'имя_пользователя', 'пароль',
            {RaiseError => 1});

# Выполнить запрос к базе.
my $sql = "INSERT INTO users (user_pass, user_name) VALUES
            ⇨ (PASSWORD('$pass1'), ENCODE('$username', 'w1cKet'))";
my $query = $dbh->do ($sql);

# Сообщить о результате.
if ($query == 1) {
  print "The user has been added! \n";
} else {
  print "The user could not be added! \n";
}

# Разорвать соединение.
$dbh->disconnect;
```

Имя и пароль нового пользователя читаются из стандартного ввода <STDIN>. Введенное значение присваивается переменной $username.

3. Запросите пароль:

```
print "Enter the password: ";
my $pass1 = <STDIN>;
print "Confirm the password: ";
my $pass2 = <STDIN>;
```

Enter the password – «Введите пароль».

Confirm the password – «Подтвердите пароль».

Чтобы защититься от случайных ошибок при вводе пароля, вы просите ввести его дважды, а затем сравниваете значения.

4. Проверьте, совпадают ли оба введенных пароля:

```
while ($pass1 ne $pass2) {
    print "The passwords you entered
    ⇨ did not match! Try again!\n";
    print "Enter the password: ";
    $pass1 = <STDIN>;
    print "Confirm the password: ";
    $pass2 = <STDIN>;
}
```

The passwords you entered did not match! Try again! – «Введенные вами пароли не совпадают! Повторная попытка!».

В этом цикле проверяется, что оба введенных пароля идентичны. Если это не так, выводится сообщение об ошибке, и пользователю предоставляется возможность еще раз набрать оба значения. Процедура повторяется до тех пор, пока пароли не совпадут.

5. Установите соединение с базой данных:

```
my $dbh = DBI->connect('DBI:mysql:
⇨ accounting', 'имя_пользователя',
⇨ ' пароль', {RaiseError => 1});
```

6. Создайте запрос и отправьте его серверу:

```
my $sql = "INSERT INTO users
⇨ (user_pass, user_name) VALUES
⇨ (PASSWORD('$pass1' ),
⇨ ENCODE('$username' , 'w1cKet' ))";
my $query = $dbh->do ($sql);
```

Этот запрос почти совпадает с приведенным в главе 5, только в него подставляются не фиксированные, а введенные пользователем значения полей. Текст запроса передается методу `do()`.

7. Напечатайте сообщение о результате выполнения запроса:

```
if ($query == 1) {
    print "The user has been added! \n";
} else {
    print "The user could not be
    ⇨ added! \n";
}
```

`The user has been added!` – «Добавлена запись о новом пользователе».

`The user could not be added` – «Запись о новом пользователе не удалось добавить».

Метод `do()` возвращает число затронутых строк для таких запросов, как `INSERT`, `UPDATE` или `DELETE`. Это число присвоено переменной `$query`; если оно равно 1, то вставлена ровно одна запись, а значит, все в порядке.

8. Закройте соединение с базой данных:

```
$dbh->disconnect;
```

9. Сохраните сценарий в файле add_user.pl, при необходимости измените права доступа к нему и запустите сценарий (рис. 7.16, 7.17).

С Не ставьте в конце SQL-запроса точку с запятой, как это принято в мониторе mysql.

С Чтобы не возникло путаницы с различными видами кавычек в запросе, пользуйтесь оператором `qq{}`, который закавычивает всю фразу, правильно обрабатывая при этом одиночные и двойные кавычки:

```
$query = qq{UPDATE имя_таблицы
⇨ SET имя_колонки='значение'
⇨ WHERE имя_колонки='другое_
⇨ значение'}
```

С В целях повышения безопасности можете воспользоваться регулярными выражениями для контроля корректности данных, введенных пользователем.

П Вышеупомянутый оператор `qq{}` не экранирует символы одиночной и двойной кавычки в значениях, полученных из стандартного ввода. Вы должны сделать это сами, например с помощью регулярных выражений.

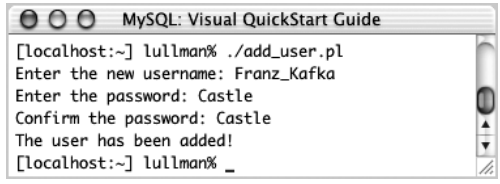

Рис. 7.16. Сценарий выведет три вопроса, а после ответа на них добавит запись о новом пользователе в базу данных

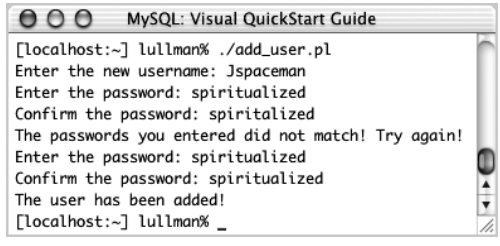

Рис. 7.17. Если два введенных пароля не совпадают, сценарий предложит ввести их еще раз

Как сослаться на колонку по ее имени

Метод `fetchrow_array()` – самый эффективный и распространенный способ извлечения результатов запроса, но у него есть существенный недостаток: он не позволяет ссылаться на колонки по имени. Впрочем, есть и другие возможности.

Первая и самая простая из них – воспользоваться методом `fetchrow_ hashref()`. Работает он так же, как `fetchrow_ array()`, но допускает обращение к колонкам по имени:

```
while (@row = $query->fetchrow_
⇨ hashref()) {
    print "$row->{'имя_колонки'}\n";
}
```

К сожалению, этот метод действует гораздо медленнее, чем `fetchrow_array()`.

Есть и другой вариант. Если вы точно знаете, сколько колонок будет возвращено и в каком порядке, то можете присвоить их значения переменным:

```
while ( ($имя_колонки1,
⇨ $имя_колонки2, $имя_колонки3) =
⇨ $query->fetchrow_array()) ) { ...
```

Еще одна альтернатива – применить метод `bind_col()` для предварительного связывания номеров колонок с именами переменных. Для этого требуется написать больший объем кода, так что за подробностями отсылаю вас к документации по модулю DBI.

Выборка данных

Извлечение результатов запроса к MySQL в Perl-сценариях чуть сложнее, чем выполнение простых запросов. Начнем с того, что сама отправка запроса серверу – это двухступенчатый процесс, состоящий из вызова методов `prepare()` и `execute()`.

```
$query = $dbh->prepare("SQL-запрос");
```

Результат, возвращенный методом `prepare()`, присваивается переменной, которую я назвал `$query` (часто ее называют также `$sth`, от statement handle – «описатель предложения»). Прежде чем вызывать метод `execute()`, надо удостовериться в успешном завершении операции `prepare()`. Простейший способ – использование функции `defined()`:

```
if (defined($query)) {
    $query->execute();
} else {
    print "Could not execute the query!\n";
}
```

`Could not execute the query!` – «Не удалось выполнить запрос».

После выполнения запроса можно извлечь его результаты с помощью метода `fetchrow_array()`. Он возвращает каждую запись в виде массива, который индексируется начиная с 0 (см. также врезку «Как сослаться на колонку по ее имени»):

```
while (@row = $query->fetchrow_
⇨ array()) {
    print "$row[1]\n";
}
```

Если колонка содержит `NULL`, то Perl вернет значение `undef`, поэтому стоит снова применить функцию `defined()` для того, чтобы проверить полученное значение перед тем, как его использовать. Отметим, однако, что для пустых строк функция `defined()` возвращает 1, а не 0.

После того как все строки извлечены, следует закончить запрос:

```
$query->finish();
```

В качестве простого примера напишем Perl-сценарий, который получает на входе имя базы данных и распечатывает названия всех таблиц в этой базе.

Извлечение результатов запроса

1. Создайте новый Perl-сценарий (листинг 7.4):

```
#!/usr/bin/perl
use DBI;
use strict;
```

Листинг 7.4. Сценарий show_tables.pl выводит перечень таблиц в базе данных

Листинг

```
#!/usr/bin/perl

# Листинг 7.4, 'show_tables.pl'

# Включить используемые модули.
use DBI;
use strict;

# Сценарий принимает один аргумент - имя базы данных.
my $database = @ARGV[0];

if (defined($database)) {

    # Установить соединение с базой данных.
    my $dbh = DBI->connect("DBI:mysql:$database", 'имя_пользователя', 'пароль',
                           {RaiseError => 1});

    # Сформулировать запрос к базе.
    my $sql = "SHOW TABLES";

    my $query = $dbh->prepare ($sql);

    if (defined($query)) {
        $query->execute();
```

2. Проверьте, введено ли имя базы данных, к которой нужно будет обращаться:

```
my $database = @ARGV[0];
if (defined($database)) {
```

Этот сценарий принимает один аргумент – имя базы данных. Его можно получить из массива @ARGV и присвоить переменной $database. Если у переменной есть значение, можно продолжать работу.

3. Установите соединение с MySQL и подготовьте запрос:

```
my $dbh = DBI->connect("DBI:mysql:
⇨ $имя_базы_данных",
⇨ 'имя_пользователя', 'пароль',
⇨ {RaiseError => 1});
my $sql = "SHOW TABLES";
my $query = $dbh->prepare ($sql);
```

Строка соединения в этом сценарии выглядит несколько иначе, чем раньше, поскольку в качестве имени базы данных фигурирует значение переменной. Поэтому пришлось вместо одиночных кавычек использовать двойные,

Листинг 7.4. Сценарий show_tables.pl выводит перечень таблиц в базе данных (окончание)

Листинг

```
  my @row;
  while (@row = $query->fetchrow_array()) {
     foreach (@row) {
        print "$row[0] \n";
     }
  }
    }
    $query->finish();

    # Разорвать соединение.
    $dbh->disconnect;

} else {
    print "Please enter a database name when calling this script! \n";
}
```

чтобы Perl-сценарий мог заменить имя переменной ее значением (*интерполировать* переменную).

4. Выполните запрос:

```
if (defined($query)) {
    $query->execute();
```

Если операция `prepare()` завершилась успешно, то переменная `$query` будет иметь определенное значение, и вы можете переходить к выполнению запроса.

5. Извлеките и распечатайте каждую запись:

```
my @row;
while (@row = $query->fetchrow_
⇨ array()) {
    foreach (@row) {
       print "$row[0]\n";
    }
}
```

Здесь показан самый безопасный метод получения всех колонок из каждой возвращенной строки. Вы печатаете по одному элементу на строке. Результат выполнения предложения `SHOW TABLES` – набор строк, каждая из которых содержит имя одной таблицы.

6. Завершите запрос и закройте соединение с базой данных:

```
}
$query->finish();
$dbh->disconnect;
```

7. Завершите основную ветвь условного оператора:

```
} else {
    print "Please enter a database
    ⇨ name when calling this
    ⇨ script! \n";
}
```

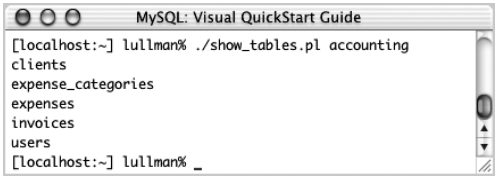

Рис. 7.18. Сценарий show_tables.pl распечатывает имена всех таблиц в указанной базе данных (в этом примере – accounting)

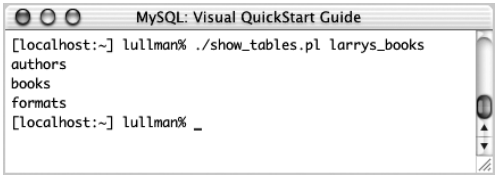

Рис. 7.19. Если у сценария есть полномочия на доступ к базе данных, он распечатает имена таблиц без какой-либо модификации

Если пользователь забыл указать имя базы данных, сценарий печатает соответствующее сообщение: `Please enter a database name when calling this script!` – «Пожалуйста, введите имя базы данных при обращении к этому сценарию».

8. Сохраните сценарий в файле show_tables.pl, при необходимости измените права доступа к нему и запустите сценарий командой `./show_tables.pl` *имя_базы_данных* или `perl show_tables.pl` *имя_базы_данных* (рис. 7.18, 7.19).

   Напомню, что у сценария должны быть полномочия на доступ к указанной базе данных. Поэтому проще всего в методе `connect()` ввести имя и пароль пользователя root, так как он имеет доступ ко всем базам.

С Поскольку вышеописанный запрос возвращает строки, состоящие всего из одной колонки, цикл `while` можно упростить:

```
while (@row = $query-
>fetchrow_array()) {
   print "$row[0]\n";
}
```

С В Perl нельзя узнать, сколько записей вернуло предложение `SELECT`, не перебрав их все. Но можно для этой цели воспользоваться функцией `COUNT()`, имеющейся в MySQL.

С Если вы работаете с таблицами, поддерживающими транзакции (например, типа InnoDB), то можете воспользоваться средствами поддержки транзакций, реализованными в модуле DBD::mysql начиная с версии 1.2216. За подробностями обратитесь к документации по MySQL.

П Зарезервированное слово `undef` в Perl означает то же самое, что и `NULL` в SQL: отсутствие какого бы то ни было значения.

Значения автоинкрементного поля

В главе 5 я рассказал о функции LAST_INSERT_ID(), применяемой для получения значения автоинкрементного поля в последней вставленной записи. В главе 6 была продемонстрирована функция PHP mysql_insert_id(), служащая той же цели. В Perl можно использовать такую конструкцию:

```
$query = $dbh->do("INSERT INTO
⇨ имя_таблицы (имя_колонки1, имя_колонки2)
⇨ VALUES(значение1, 'значение2')");

$insert_id = $dbh->{'mysql_insertid'};
```

Давайте модифицируем сценарий add_user.pl так, чтобы он возвращал идентификатор только что добавленного пользователя.

Получение значения автоинкрементного поля

1. Откройте в текстовом редакторе файл add_user.pl (см. листинг 7.3).
2. После выполнения запроса (строка 31) добавьте такую строку (листинг 7.5):

   ```
   my $user_id = $dbh->{'mysql_insertid'};
   ```

   При вставке записи о пользователе в таблицу для поля user_id указывается значение NULL. Поскольку для этого поля задан атрибут AUTO_INCREMENT, то MySQL вместо NULL заносит в него следующее по порядку целое число. Сценарий получает это значение и записывает в переменную $userid.

Листинг 7.5. Сценарий add_user2.pl извлекает значение автоинкрементного поля из только что вставленной в таблицу users записи

Листинг

```
#!/usr/bin/perl

# Листинг 7.5, 'add_user2.pl'

# Включить используемые модули:
use DBI;
use strict;

# Запросить информацию о пользователе:
print "Enter the new username: ";
my $username = <STDIN>;
print "Enter the password: ";
my $pass1 = <STDIN>;
print "Confirm the password: ";
my $pass2 = <STDIN>;

# Проверить, что оба введенных пароля
# совпадают:
while ($pass1 ne $pass2) {
  print "The passwords you entered did
  ⇨ not match! Try again!\n";
  print "Enter the password: ";
  $pass1 = <STDIN>;
  print "Confirm the password: ";
  $pass2 = <STDIN>;
}

# Соединиться с базой данных:
my $dbh = DBI->connect('DBI:mysql:
⇨ accounting', 'имя_пользователя',
⇨ 'пароль',
            {RaiseError => 1});

# Выполнить запрос к базе:
my $sql = "INSERT INTO users (user_pass,
⇨ user_name) VALUES
⇨ (PASSWORD('$pass1'),
⇨ ENCODE('$username',
⇨ 'w1cKet'))";
my $query = $dbh->do ($sql);
```

Листинг 7.5. Сценарий add_user2.pl извлекает значение автоинкрементного поля из только что вставленной в таблицу users записи (окончание)

Листинг

```
# Получить идентификатор пользователя
# user_id:
my $userid = $dbh->{'mysql_insertid'};

# Сообщить о результате:
if ($query == 1) {
  print "User number $userid has been
  ⇨ added! \n";
} else {
  print "The user could not be added! \n";
}

# Разорвать соединение.
$dbh->disconnect;
```

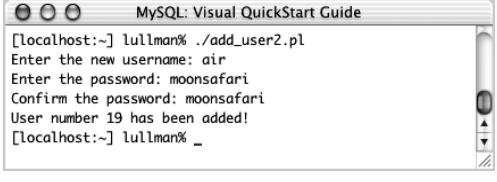

Рис. 7.20. Сценарий add_user2.pl возвращает идентификатор нового пользователя

3. Проверив успешность выполнения предыдущей операции, распечатайте идентификатор, присвоенный новому пользователю:

   ```
   print "User number $userid has been
   ⇨ added! \n";
   ```

   В этом примере использование идентификатора сводится просто к его печати.

   `User number $userid has been added!` — «Пользователь номер $userid был добавлен».

4. Сохраните сценарий в файле add_user2.pl, измените права доступа и запустите сценарий на исполнение (рис. 7.20).

Получить значение автоинкрементного поля можно и следующим образом:

```
$query->{'insertid'}.
```

Безопасность

Начиная с версии MySQL 3.22 появилась возможность хранить в конфигурационном файле информацию, которая автоматически используется клиентами MySQL и функциями API. Эта процедура не только проще, но и безопаснее ручного ввода значений.

Модули Msql-Mysql (начиная с версии 1.2009) позволяют использовать один из таких конфигурационных файлов, чтобы не «зашивать» информацию о доступе к базе непосредственно в сценарий. Для этого нужно добавить в имя источника данных строку:

```
mysql_read_default_file=имя_файла
```

Например, вот как могла бы выглядеть модифицированная строка соединения:

```
$dbh=DBI->connect("DBI:mysql:
⇨имя_базы_данных;mysql_read_
⇨default_file=
⇨/путь/к/.my.cnf", $user, $password);
```

В таком случае значения переменных `$user` и `$password` будут прочитаны из конфигурационного файла, который, надо полагать, хранится в безопасном месте.

В MySQL есть много различных конфигурационных файлов, используемых для разных целей (см. табл. 7.1). Коротко говоря, любой параметр, который может быть указан в командной строке при запуске приложения (например, `--database` или `--host`), можно сохранить и в конфигурационном файле. Отмечу, что MySQL читает конфигурационные файлы в том порядке, в котором они перечислены в табл. 7.1; при этом значения, прочитанные позже, замещают те, что были прочитаны раньше (другими словами, пароль, обнаруженный в пользовательском файле .my.cnf, замещает пароль, хранящийся

Листинг 7.6. Конфигурационный файл будет использоваться всеми клиентами MySQL, так что больше не надо хранить имя и пароль пользователя в тексте сценария

Листинг

```
# Мой конфигурационный файл
[client]
user=имя_пользователя
password=пароль
```

в глобальном файле my.cnf). Кроме того, параметры, заданные в командной строке, имеют больший приоритет, чем хранящиеся в файле.

Использование конфигурационного файла

1. Создайте в редакторе новый документ (листинг 7.6):

   ```
   # Мой конфигурационный файл.
   ```

   Комментарии начинаются со знака #, как в Perl-сценариях.

2. Запишите имя и пароль, которыми будет пользоваться клиент MySQL:

   ```
   [client]
   user=имя_пользователя
   password=пароль
   ```

   Заголовок раздела `[client]` говорит о том, что следующие строки относятся ко всем клиентским приложениям, включая монитор mysql и любой Perl-сценарий. Заключать имя и пароль пользователя в кавычки не надо.

3. Сохраните файл под названием:
   - ~/.my.cnf – для платформ UNIX и Mac OS X (вместо символа ~ подставьте путь к вашему начальному каталогу);

Таблица 7.1. Сервер MySQL, стандартные клиентские приложения и даже Perl-сценарии могут пользоваться указанными конфигурационными файлами для хранения значений параметров

Имя файла	Платформа	Содержимое
/etc/my.cnf	UNIX и Mac OS	Глобальные параметры
<DATADIR>/my.cnf	UNIX и Mac OS	Параметры сервера (<DATADIR> – это каталог, где хранятся данные MySQL)
~/.my.cnf	UNIX и Mac OS	Пользовательские параметры (вместо символа ~ указывается начальный каталог пользователя)
windows-system-directory\my.ini	Windows	Глобальные параметры
C:\my.cnf	Windows	Глобальные параметры

– C:\my.cnf, C:\WINNT\my.ini или C:\WINDOWS\my.ini – для платформы Windows (имя диска и путь к системному каталогу измените в соответствии с тем, где на вашем компьютере установлена система Windows).

При сохранении следите за тем, чтобы не затереть уже имеющийся конфигурационный файл, который иногда бывает скрытым. Например, приложение WinMySQLAdmin в Windows могло уже создать файл my.ini. Кроме того, на платформах UNIX и Mac OS X не забудьте указать начальную точку в имени файла.

4. Измените права доступа к файлу:

```
chmod 600 .my.cnf
```

В системах UNIX и Mac OS X эта команда разрешает доступ к файлу только его владельцу.

Теперь, подготовив конфигурационный файл, я могу повысить безопасность сценария, исключив из него имя и пароль пользователя.

5. Откройте в редакторе файл mysql_ connect.pl (листинг 7.2).

6. Определите две новые переменные (листинг 7.7):

```
my $user;
my $password;
```

Поскольку ранее вы установили режим строгой проверки, эти переменные должны быть объявлены до первого использования. Сценарий прочтет их значения из конфигурационного файла.

Листинг 7.7. Сценарий mysql_connect2.pl стал более безопасным и переносимым, поскольку в нем больше не «зашиты» имя и пароль пользователя

Листинг

```
#!/usr/bin/perl

# Листинг 7.7, 'mysql_connect2.pl'

# Включить используемые модули:
use DBI;
use strict;

# Соединиться с базой данных:
my $user;
my $password;
my $dbh = DBI->connect('DBI:mysql:
⇨ accounting;mysql_read_default_file=
/Users/lullman/.my.cnf', $user,
⇨ $password, {RaiseError => 1});

# Сообщить об успешном установлении
# соединения:
if ($dbh) {
    print "Successfully connected
    ⇨ to the database! \n";
}

# Разорвать соединение:
$dbh->disconnect;
```

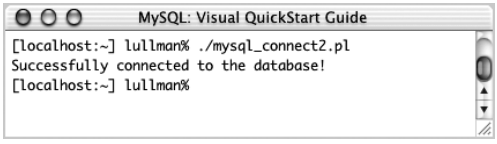

Рис. 7.21. Сценарий, устанавливающий соединение, работает так же, как и раньше, но он стал более безопасным

Вы должны изменить строку так, чтобы она ссылалась на файл, который был создан на этапе 3. Можно использовать как абсолютное (например, `/users/lullman/.my.cnf`), так и относительное (например, `../.my.cnf`) имя. Символ ~ (тильда) означает, что конфигурационный файл следует искать в начальном каталоге пользователя, от имени которого исполняется сценарий.

7. Изменим строку соединения, указав в ней имя конфигурационного файла:

```
my $dbh = DBI->connect('DBI:mysql:
⇨ accounting;mysql_read_default_file=
⇨ /Users/lullman/.my.cnf', $user,
⇨ $password, {RaiseError => 1});
```

8. Сохраните файл (рекомендую переименовать его в mysql_connect2.pl) и запустите на своем сервере (рис. 7.21).
9. Если хотите, повторите действия, описанные в пп. 5–8, для всех созданных ранее Perl-сценариев.

П Созданный конфигурационный файл будет использоваться всеми клиентскими приложениями MySQL, а не только Perl-сценариями.

С Чтобы Perl-сценарий автоматически читал конфигурационный файл, хранящийся в вашем начальном каталоге (в системах UNIX и Mac OS X), можно применить такую конструкцию: `mysql_read_default_file=$ENV{HOME}/.my.cnf`.

П Подробнее о конфигурационных файлах рассказывается на странице www.mysql.com/doc/O/p/Option_files.html.

8

MySQL и Java

За последние несколько лет Java стал одним из самых популярных языков программирования – во многом из-за своей платформенной независимости. Java можно использовать для создания аплетов, работающих в составе Web-страниц, для написания JSP-страниц (Java Server Pages), а также для разработки автономных приложений.

Доступ к базам данных программы на языке Java всегда обеспечивается при помощи стандарта JDBC (Java DataBase Connectivity). Поэтому, если код написан надлежащим образом и следует стандарту SQL, Java-приложение можно легко переносить между разными платформами и разными СУБД.

В двух предыдущих главах я уже подчеркивал, что не ставлю себе задачей научить читателя программированию на том или ином языке, хотя и буду объяснять, как и с какой целью написаны приведенные в книге сценарии. Изучив эти примеры, начинающий пользователь сможет написать простое Java-приложение. Но основное внимание будет обращено на применение JDBC и драйвера MM.MySQL для взаимодействия с СУБД MySQL. В самом руководстве по MySQL эта тема практически не освещается, но вы можете обратиться к документации по JDBC и MM.MySQL (в приложении 3 приведены соответствующие ссылки).

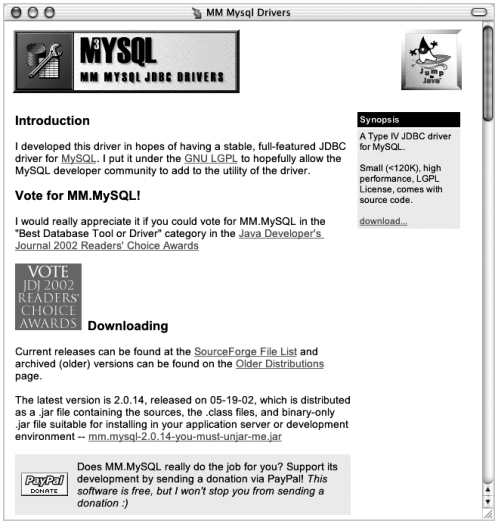

Рис. 8.1. Драйвер MM.MySQL находится на сайте SourceForge.net наряду с тысячами других проектов с открытым исходным кодом

Установка поддержки Java для MySQL

Для доступа к базе данных MySQL из приложения на языке Java необходимы:

- Java SDK версии 1.1 или старше (включающей JDBC);
- пакет MM.MySQL.

Установка самой среды разработки Java в книге не рассматривается, но для среднего пользователя это не слишком сложная задача. Большинству пользователей будет достаточно издания Java 2 Platform, Standard Edition (J2SE), которое можно загрузить с сайта http://java.sun.com. Пользователи Mac OS X получают все преимущества тесной интеграции Java с операционной системой (система Mac OS X версии 10.1.5 поставляется с версией J2SE версии 1.3.1). В Windows эта технология поддерживалась на некоторых стадиях эволюции системы.

В настоящем разделе я покажу, как включить драйвер MM.MySQL в вашу инсталляцию Java.

Порядок установки драйвера MM.MySQL

1. Загрузите текущую версию MM.MySQL с сайта http://mmmysql.sourceforge.net (рис. 8.1).

   В настоящее время Марк Мэтьюс (Mark Matthews), создатель этого драйвера, выложил на сайт версию 2.0.14 в виде jar-файла.

2. Распакуйте загруженный файл с помощью одного из следующих приложений:
   - `jar xvf` */путь/к/файлу*`.jar` (из командной строки);
   - StuffIt (Macintosh);
   - WinZip (Windows).

Расширение .jar говорит о том, что файл представляет собой Java-архив, который нужно распаковать. Можно сделать это с помощью программы jar, входящей в комплект Java, или множества других приложений-архиваторов.

На момент написания этой книги драйвер MM.MySQL распространялся в виде одного большого jar-файла. После его раскрытия вы обнаружите еще один jar-файл – это, собственно, и есть драйвер. Наличие двух архивов может стать источником путаницы, но, к счастью, в имени дистрибутивного файла есть фраза «you-must-unjar-me» (меня надо распаковать).

3. Переместите драйвер туда, где он должен находиться (рис. 8.2):

```
cp mm.mysql-2.0.14-bin.jar /путь/к
⇨ /папке
```

Вместо командной строки можно воспользоваться приложением Windows Explorer или любым другим файловым менеджером.

На самом деле точное местонахождение драйвера не так важно. Главное, чтобы приложение могло его найти. Обычно используются следующие пути:

- `C:\java`
- `C:\Program Files\j2sdk1.4.01\lib\ext`
- `/usr/local/lib/mysql`
- `<JAVA_HOME>/jre/lib/ext`

Подойдет любой из этих вариантов. Последний ссылается на корневой каталог инсталляции Java на вашем компьютере. Еще одна альтернатива – поместить драйвер в тот же каталог, где находятся использующие его сценарии.

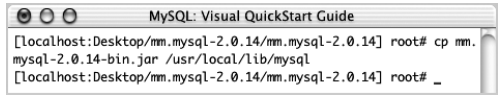

Рис. 8.2. JDBC-драйвер должен находиться в определенном месте. Переменную `CLASSPATH` также следует подредактировать

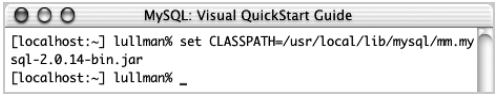

Рис. 8.3. Установка переменной CLASSPATH необходима для того, чтобы приложение могло найти драйвер

4. Добавьте путь к файлу в переменную окружения CLASSPATH (рис. 8.3):

```
java set CLASSPATH=/путь/к/
⇨ mm.mysql-2.0.14-bin.jar
```

Если вы решили воспользоваться двоичной версией драйвера, то вышеупомянутая команда сообщит виртуальной машине Java, где искать драйвер. Можно было бы также задать путь к классам в момент запуска сценария. А если драйвер находится в том же каталоге, что и сценарий, или в подкаталоге lib/ext корневого каталога Java, то шаг 4 можно вовсе пропустить.

С При развертывании Web-приложения поместите JDBC-драйвер (jar-файл) в каталог WEB-INF/lib.

П На время разработки можно также пользоваться исходными файлами классов, составляющих mm.mysql, добавив каталог org и его содержимое к переменной CLASSPATH вместо файла mm.mysql-2.0.14-bin.jar.

П Можно указывать несколько путей к классам (обычно так и делается), отделяя один от другого точкой с запятой.

С Записывая команды, имена классов и файлов, не забывайте, что в языке Java прописные и строчные буквы различаются.

Соединение с базой данных

Перед началом работы с JDBC неплохо убедиться, что соединение с базой данных устанавливается успешно. Часто это самый длительный этап во всей процедуре. Для начала укажите, что вы хотите использовать классы для работы с SQL:

```
import java.sql.*;
```

Затем, если вы применяете менеджер драйверов JDBC, определите, какой драйвер использовать:

```
Class.forName("org.gjt.mm.mysql.Driver");
```

Теперь можно устанавливать соединение с базой данных, пользуясь классом `DriverManager`:

```
Connection con=DriverManager.
⇨ getConnection(url,
⇨ "имя_пользователя", "пароль");
```

Значением переменной `url` должна быть строка в формате `jdbc:`*`протокол`*`:`*`подпротокол`*, содержащая имя базы данных, имя хоста и другую информацию. В нашем случае *`протокол`* – это `mysql`, а *`подпротокол`* включает имя хоста, номер порта и имя базы данных, например:

```
jdbc:mysql://[имя_хоста][:порт]
⇨ /имя_базы_данных
```

В квадратные скобки здесь заключены необязательные параметры.

Так, для соединения с базой данных test без указания хоста значение `url` записывается следующим образом:

```
jdbc:mysql:///test
```

Если вы хотите указать имя хоста и порт, напишите:

```
jdbc:mysql://localhost:3306/test
```

В конце сценария по завершении взаимодействия с MySQL следует закрыть соединение с базой данных и освободить ресурсы методом `close()`:

```
con.close();
```

Установление соединения с MySQL

1. Создайте новый Java-класс в текстовом редакторе или в среде разработки (листинг 8.1).

Листинг 8.1. Простой Java-класс для установления соединения с базой данных MySQL

Листинг

```
import java.sql.*;

public class Connect {

   public static void main(String argv[]) throws Exception {
      Connection con = null;

     try {
       String url = "jdbc:mysql:///test";
       Class.forName("org.gjt.mm.mysql.Driver");
       con = DriverManager.getConnection(url, "", "");

      if (con != null) {
          System.out.println("A database connection has been established!");
      }
     }

     finally {
       if( con != null ) {
         try {
              con.close();
          } catch( Exception e ) {
              System.out.println(e.getMessage());
          }
       }
     }
   }
}
```

Писать Java-сценарии можно практически в любом текстовом редакторе. Ну а если у вас есть специализированная среда разработки для кодирования, компиляции и запуска Java-программ, тем лучше!

2. Включите классы `sql` и определите собственный класс:

```
import java.sql.*;
public class Connect {
    public static void main(String
    ⇨ argv[]) throws Exception {
```

Наше первое приложение будет называться Connect и состоять только из одного метода `main`. В дальнейшем я буду использовать аналогичные конструкции.

3. Инициализируйте переменную класса `Connection`:

```
Connection con = null;
```

Переменная `con` принадлежит классу `java.sql.Connection`, импортированному вместе с прочими директивой `import java.sql.*`. Здесь `con` присваивается начальное значение `null`.

4. Установите соединение с базой данных test:

```
try {
    String url = "jdbc:mysql:///test";
    Class.forName("org.gjt.mm.mysql.
    ⇨ Driver");
    con = DriverManager.get
    ⇨ Connection(url, "", "");
```

Во второй строке определяется переменная `url`, содержащая строку соединения. В третьей строке указывается, какой драйвер загрузить, а в последней осуществляется попытка установить соединение и записать ссылку на него

в ранее объявленную переменную con. Здесь я не указываю ни имени хоста, ни имени и пароля пользователя. Это имеет смысл, если MySQL на вашем компьютере разрешает доступ к базе данных test анонимному пользователю без пароля с любого хоста – иначе в данной точке приложение завершится с ошибкой, поэтому вам придется подкорректировать текст.

5. Напечатайте сообщение, если соединение было установлено:

```
if (con != null) {
    System.out.println("A database
    ⇨ connection has been
    ⇨ established!");
}
```

A database connection has been established! – «Соединение с базой данной установлено».

Вообще-то этот код можно считать излишним, поскольку при возникновении ошибки соединения с базой данных будет возбуждено исключение. С другой стороны, стоит как-то отметить, что подключение осуществилось.

6. Завершите предложение try и весь класс:

```
}
finally {
if( con != null ) {
    try {
       con.close();
    } catch( Exception e ) {
       System.out.println
       ⇨ (e.getMessage());
    }
}
}
}
}
```

С точки зрения MySQL, единственно важная часть в данном фрагменте – `try { con.close(); }`. Здесь закрывается соединение. Если по какой-то причине сделать этого не удастся, будет напечатано сообщение об ошибке.

7. Сохраните сценарий в файле Connect.java.

   Правила Java требуют, чтобы имя файла в точности совпадало с именем содержащегося в нем класса.

8. Откомпилируйте файл Connect.java. Это можно сделать двумя способами:
   - набрать команду `javac /путь/к/Connect.java`;
   - воспользоваться средой разработки для компиляции класса.

9. Выполните программу Connect (рис. 8.4, 8.5). Здесь тоже есть две возможности:
   - открыв каталог, в котором находится файл Connect.java, набрать команду `java Connect.java`;
   - воспользоваться средой разработки для исполнения класса.

   Если вы не добавили путь к драйверу в переменную `CLASSPATH`, не поместили драйвер в тот же каталог, где находится файл Connect.java, и вообще никоим образом не сообщили виртуальной машине, где искать драйвер, нужно будет указать его местонахождение непосредственно при запуске программы. Для этого предназначена такая команда:

```
java -cp .;/путь/к/mm.mysql-
⇨ 2.0.14-bin.jar Connect
```

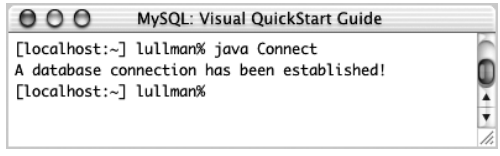

Рис. 8.4. Если приложение завершилось успешно, на экране появляется такое сообщение

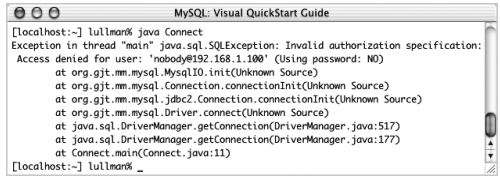

Рис. 8.5. Если программа Connect не смогла соединиться с базой данных, выводится сообщение об ошибке – в данном случае `Access denied for user ...` (Пользователю ... доступ запрещен)

П Имя `org.gjt.mm.mysql.Driver` ссылается на файлы классов драйвера, которые находятся в подкаталоге org/gjt/mm/mysql папки дистрибутива пакета MM.MySQL.

С Иногда при указании строки `localhost` в качестве имени хоста Java заменяет ее на IP-адрес хоста (например, 192.168.1.1 или 127.0.0.1), что может привести к конфликтам полномочий, хранящихся в базе данных MySQL.

С В строку `url` можно включать дополнительные параметры в следующем формате:

```
jdbc:mysql:///
test?имя_переменной1=
⇨ значение1&имя_переменной2=значение2
```

Например:

```
jdbc:mysql:///test?user=
⇨ John&password=Javaman
```

Простые запросы

Успешно установив соединение с базой данных, можно приступать к взаимодействию с ней. Для начала выполним простой запрос, изменяющий базу данных, но не возвращающий строк. В эту категорию попадают запросы, начинающиеся со слов `INSERT`, `ALTER`, `CREATE`, `DELETE` и `UPDATE`.

Выполнение простого запроса делится на два этапа. Сначала создается переменная, представляющая предложение запроса, а затем с помощью этой переменной он выполняется. Типичный код выглядит следующим образом:

```
Statement stmt;
stmt = con.createStatement();
stmt.executeUpdate("DELETE FROM
⇨ tablename WHERE column='value'");
stmt.close();
```

Метод `createStatement()` создает открытый канал, через который запрос передается серверу. Метод `executeUpdate()` отсылает запрос, а метод `close()` освобождает ресурсы, выделенные запросу. В качестве примера я напишу приложение, которое получает аргументы из командной строки и заполняет таблицу clients в базе данных accounting.

Выполнение простого запроса

1. Создайте в текстовом редакторе или среде разработки для Java новое приложение (листинг 8.2):

```
import java.sql.*;
public class Insert {
    public static void main(String
    ⇨ args[]) throws Exception {
```

Начало этого сценария отличается от того, что вы видели в листинге 8.1, только именем класса: `Insert` вместо `Connect`.

2. Инициализируйте переменные:

```
Connection con = null;
Statement stmt = null;
```

Листинг 8.2. Java-класс `Insert` принимает три аргумента в командной строке и интерпретирует их как имя клиента, имя и фамилию контактного лица в новой записи таблицы clients

Листинг

```
import java.sql.*;

// Листинг 8.2 "Insert.java"

public class Insert {

   public static void main(String args[]) throws Exception {
       Connection con = null;
       Statement stmt = null;

       try {
           String url = "jdbc:mysql:///accounting";
           Class.forName("org.gjt.mm.mysql.Driver");
           con = DriverManager.getConnection(url, "имя_пользователя", "пароль");

           stmt = con.createStatement();
           stmt.executeUpdate("INSERT INTO clients (client_name, " +
              contact_first_name, contact_last_name) VALUES ('" +
             args[0] + "', '" + args[1] + "', '" + args[2] + "')");
           System.out.println("The values were added to the database!");
       }

       catch (SQLException e) {
          e.printStackTrace();
       }

       finally {
          if ( con != null ) {
             try {
                con.close();
                stmt.close();
             } catch( Exception e ) {
                System.out.println(e.getMessage());
             }
          }
       }
   }
}
```

Помимо переменной, представляющей соединение, для выполнения простого запроса нужна переменная типа `Statement`.

3. Установите соединение с базой данных accounting:

```
try {
    String url = "jdbc:mysql:///
    ⇨ accounting";
    Class.forName("org.gjt.mm.mysql.
    ⇨ Driver");
    con = DriverManager.
    ⇨ getConnection(url,
    ⇨ "имя_пользователя", "пароль");
```

По сравнению с предыдущим сценарием нужно внести три очень важных изменения: указать имя базы данных accounting вместо test, а также поменять имя и пароль пользователя. Не забывайте, что данный пользователь должен обладать правами на соединение с базой данных accounting и на ее модификацию.

4. Выполните запрос `INSERT`:

```
stmt = con.createStatement();
stmt.executeUpdate("INSERT INTO
⇨ clients (client_name,
⇨ contact_first_name, contact_
⇨ last_name) VALUES ('" + args[0]
⇨ + "', '" + args[1] + "', '"
⇨ + args[2] + "')");
System.out.println("The values were
⇨ added to the database!");
```

Первый шаг – создание переменной `stmt` методом `createStatement()` объекта `con`. Затем текст запроса передается в качестве аргумента методу `executeUpdate()` объекта `stmt`. В данном случае я указываю в запросе имя клиента, а также имя и фамилию контактного лица. Все эти значения были заданы в

командной строке при запуске приложения. Для доступа к ним используются переменные `args[0]`, `args[1]` и `args[2]`, переданные средой исполнения методу `main()` класса `Insert`.

`The values were added to the database!` — «Значения добавлены в базу данных».

5. Завершите предложение `try` и перехватите возможные исключения:

```
}
catch (SQLException e) {
   e.printStackTrace();
}
```

Код в блоке `catch` сообщает об ошибках MySQL, произошедших во время выполнения запроса. По способу использования и полученному результату это аналогично команде `System.out.println (e.getMessage())` в классе `Connect`.

6. Завершите класс:

```
finally {
if ( con != null ) {
   try {
        con.close();
        stmt.close();
   } catch( Exception e ) {
   System.out.println
   ⇨ (e.getMessage());
   }
}
}
}
}
```

Одно дополнение по сравнению с предыдущим листингом: помимо соединения `con` здесь закрывается также предложение `stmt`. При этом освобождаются ресурсы, захваченные приложением.

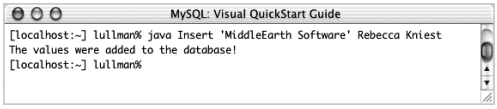

Рис. 8.6. Из Java-программы можно добавлять записи в базу данных, указав значения полей в командной строке

7. Сохраните текст программы в файле Insert.java.
8. Откомпилируйте файл Insert.java.

   Для запуска приложения выполните действия, описанные в предыдущем разделе, но не забудьте указать в командной строке необходимые параметры (имя клиента, имя и фамилию контактного лица):

```
java -cp .;/путь/к/mm.mysql-
⇨ 2.0.14-bin.jar Insert
⇨ имя_клиента контакт1 контакт2
```

Если значение аргумента должно содержать пробелы, заключите его в кавычки (рис. 8.6).

Выборка данных

Для реализации запроса, возвращающего данные из базы, потребуется чуть более сложная программа.

Прежде всего, теперь для выполнения запроса следует вызвать метод `executeQuery()`, а не `executeUpdate()`. Этот метод возвращает объект класса `ResultSet`, с помощью которого можно получить доступ к каждой из выбранных строк. Проще всего сделать это в цикле `while`, пользуясь методом `next()` для перехода к следующей записи:

```
while (results.next()) {
// Как-либо обработать результаты.
}
```

В ходе обработки результатов можно записать значения колонок в переменные соответствующих типов. Так, метод `getInt()` извлекает целочисленное значение, а метод `getString()` – текстовое. Тот и другой принимают в качестве аргумента имя или номер колонки (нумерация начинается с 1).

```
while (rs.next()) {
   int key = rs.getInt("номер_колонки");
   String value = rs.getString
   ⇨ ("строковое_имя_колонки")
}
```

Извлечение данных

1. Создайте в текстовом редакторе или среде разработки для Java новое приложение (листинг 8.3):

```
import java.sql.*;
public class Select {
    public static void main(String
    args[]) throws Exception {
```

2. Инициализируйте переменные:

```
Connection con = null;
Statement stmt = null;
ResultSet rs = null;
```

Листинг 8.3. Класс `Select` выполняет простой запрос к базе и выводит результаты

Листинг

```
import java.sql.*;

// Листинг 8.3 'Select.java'

public class Select {

   public static void main(String args[]) throws Exception {
       Connection con = null;
       Statement stmt = null;
       ResultSet rs = null;

      try {
          String url = "jdbc:mysql:///accounting";
          Class.forName("org.gjt.mm.mysql.Driver");
          con = DriverManager.getConnection(url, "имя_пользователя", "пароль");

          stmt = con.createStatement();
          rs = stmt.executeQuery("SELECT client_id, client_name
          ⇨ FROM clients LIMIT 5");

          while (rs.next()) {
              int key = rs.getInt("client_id");
              String value = rs.getString("client_name");

              System.out.println(key + ": " + value + "\n");
          }
      }
```

В этом приложении появляется новая переменная `rs` типа `ResultSet`. Она используется для доступа к результатам запроса.

3. Установите соединение с базой данных accounting:

```
try {
    String url = "jdbc:mysql:///
    ⇨ accounting";
    Class.forName("org.gjt.mm.mysql.
    ⇨ Driver");
    con = DriverManager.
    getConnection(url,
    ⇨ "имя_пользователя", "пароль");
```

Как и раньше, не забудьте, что указанный пользователь должен иметь права на соединение с базой данных и осуществление выборки из нее.

4. Выполните запрос `SELECT`:

```
stmt = con.createStatement();
rs = stmt.executeQuery("SELECT
⇨ client_id, client_name
⇨ FROM clients LIMIT 5");
```

И в этом случае сначала методом `createStatement()` объекта `con` создается переменная `stmt`. Но затем текст запроса передается методу `executeQuery()`, а не `executeUpdate()`. Данный демонстрационный фрагмент позволяет извлечь пять записей о клиентах. Естественно, вы можете передать на выполнение любой запрос, совместимый со стандартом SQL92.

5. Распечатайте возвращенные данные:

```
while (rs.next()) {
    int key = rs.getInt("client_id");
    String value = rs.getString
    ⇨ ("client_name");
   System.out.println(key + ": " +
    ⇨ value + "\n");
}
```

Листинг 8.3. Класс `Select` выполняет простой запрос к базе и выводит результаты (окончание)

Листинг

```
        catch (SQLException e) {
           e.printStackTrace();
        }

        finally {
           if (rs != null) {
              try {
                   rs.close();
               } catch (SQLException e) {
                  e.printStackTrace();
              }
           }
           if (stmt != null) {
              try {
                   stmt.close();
               } catch (SQLException e) {
                  e.printStackTrace();
              }
           }
           if (con != null) {
              try {
                   con.close();
               } catch (SQLException e) {
                  e.printStackTrace();
              }
           }
        }
    }
}
```

В этом цикле мы извлекаем каждую из отобранных записей (их не должно быть больше пяти), а затем с помощью методов `getInt()` и `getString()` выводим из записи значения целой и строковой колонок соответственно. Имена колонок задаются явно.

6. Завершите предложение `try` и перехватите возможные ошибки:

```
}
catch (SQLException e) {
    e.printStackTrace();
}
```

7. Завершите класс и освободите все ресурсы:

```
finally {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (con != null) {
        try {
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
}
}
```

К освобождению ресурсов следует относиться с особым вниманием, если речь идет о запросах типа SELECT. И чем сложнее запрос, тем более это существенно. В конце программы производится попытка закрыть результирующий набор, объект Statement и объект Connection (в указанном порядке), если ранее им были присвоены значения.

8. Сохраните текст в файле Select.java и запустите приложение (рис. 8.7).

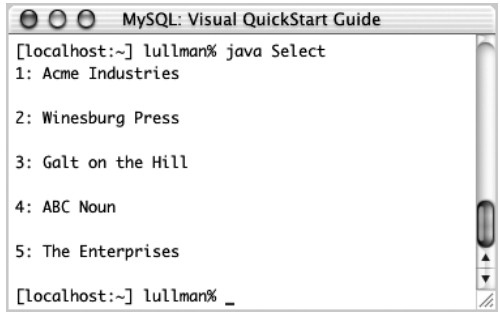

Рис. 8.7. Класс Select выполняет запрос и выводит результаты

П Начиная с версии JDBC 2.0 результирующий набор можно прокручивать как вперед – методом next(), так и назад – методом previous().

П Методы getInt() и getString() работают быстрее, если ссылаться на колонки по их номеру, а не по имени.

С Для получения информации о конкретной колонке (имени, типе данных и т.д.) можно воспользоваться методом getMetaData().

Использование файлов свойств

В этой главе я жестко «зашивал» в код каждого приложения имя базы данных, а также имя и пароль пользователя. Такое решение вполне допустимо, но не является идеальным ни с точки зрения стиля программирования, ни с точки зрения безопасности.

Поэтому я познакомлю вас еще с одной концепцией – применением файла свойств, который во многом напоминает конфигурационный файл, знакомый программисту на PHP, а также файл .my.cnf, используемый Perl-сценариями: в нем содержатся параметры приложения. При изменении любого из них, например имени базы данных или пароля, достаточно отредактировать только файл свойств, а не каждый сценарий. А для программистов на Java еще более важно, что в этом случае не нужно перекомпилировать классы.

Файл свойств – это обычный текстовый файл со строками вида *имя=значение*. Обычно ему присваивается расширение .properties.

Написав файл свойств, вы можете получить доступ к нему с помощью класса `ResourceBundle` (из пакета `java.util`):

```
ResourceBundle bundle =
⇨ ResourceBundle.getBundle
⇨ ("имя_файла_свойств");
String name = bundle.getString
⇨ ("имя_параметра");
```

Замечу, что имя файла свойств указывается без расширения.

Теперь я создам простой файл свойств для всех сценариев, приведенных в этой главе, а кроме того, модифицирую сценарий Connect.java так, чтобы он читал параметры из этого файла.

Использование файла свойств

1. Создайте в редакторе новый текстовый документ (листинг 8.4).
2. Присвойте значения свойствам `Driver`, `URL`, `User` и `Password`:

   ```
   Driver=org.gjt.mm.mysql.Driver
   URL=jdbc:mysql:///accounting
   user=имя_пользователя
   password=пароль
   ```

   В файле свойств будут храниться четыре параметра (можете при желании включить в URL имя хоста). Обратите внимание, что значения параметров не заключаются в кавычки.
3. Сохраните текст в файле Accounting.properties.

   Лучше сохранять файл свойств в том же каталоге, где находится использующий его сценарий.

   Коль скоро файл свойств создан, я применю его в сценарии Connect.java.
4. Откройте файл Connect.java (листинг 8.1) в текстовом редакторе или в среде разработки для Java.
5. Импортируйте классы из пакета `java.util` после строки, содержащей команду импорта классов для работы с базами данных:

   ```
   import java.util.*;
   ```

   Класс `ResourceBundle` находится в пакете `java.util`. Если импортировать пакет целиком, этот класс можно будет применять для считывания значений из внешнего источника.
6. Измените название класса:

   ```
   public class Connect2 {
   ```

   Эта операция не обязательна – я воспользовался ей для удобства.

Листинг 8.4. В файле свойств хранятся параметры приложения

Листинг

```
Driver=org.gjt.mm.mysql.Driver
URL=jdbc:mysql:///accounting
user=имя_пользователя
password=пароль
```

7. Измените соответствующую строку исходного файла, присвоив значение переменной класса `ResourceBundle`:

```
ResourceBundle bundle =
⇨ ResourceBundle.getBundle
⇨ ("Accounting");
```

Переменная `bundle` будет нужна для доступа к информации, хранящейся в файле свойств.

Листинг 8.5. Модифицированный сценарий Connect.java (переименованный в Connect2.java), в котором не «зашиты» параметры соединения

Листинг

```
import java.sql.*;
import java.util.*;

// Листинг 8.5 "Connect2.java"

public class Connect2 {

   public static void main(String args[]) throws Exception {
      Connection con = null;
       ResourceBundle bundle = ResourceBundle.getBundle("accounting");

     try {
       String url = bundle.getString("URL");;
       Class.forName(bundle.getString("Driver"));
       con = DriverManager.getConnection(url, bundle.getString("user"),
                   bundle.getString("password"));

      if (con != null) {
          System.out.println("A database connection has been established!");
       }
     }

     finally {
       if (con != null ) {
           try { con.close(); }
                catch( Exception e ) { }
       }
     }
   }
}
```

8. Модифицируйте предложение `try` так, чтобы читать значения из файла свойств:

```
try {
    String url = bundle.getString
    ⇨ ("URL");
    Class.forName(bundle.getString
    ⇨ ("Driver"));
    con = DriverManager.
    ⇨ getConnection(url,bundle.
    ⇨ getString("User"), bundle.
    ⇨ getString("Password"));

if (con != null) {
    System.out.println("A database
    ⇨ connection has been
    ⇨ established!");
    }
}
```

`A database connection has been established!` – «Соединение с базой данных установлено».

Здесь используются прочитанные из файла, а не «зашитые» значения: имя драйвера, URL, имя и пароль пользователя. При такой структуре сценарий без изменений будет работать на разных платформах и с разными базами данных – достаточно лишь отредактировать файл свойств.

9. Сохраните модифицированный текст в файле Connect2.java, откомпилируйте и запустите его на выполнение (рис. 8.8).

Если вы не изменили имя класса, то текст следует сохранить в файле Connect.java. В любом случае файл свойств должен находиться там же, где файл сценария.

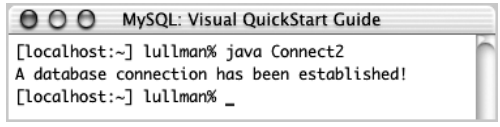

Рис. 8.8. Сценарий Connect2 отличается от своего предшественника (рис. 8.4) только источником значений параметров соединения

С Описанное выше применение класса `ResourceBundle` – не самое удачное решение. Было бы лучше вместо него воспользоваться классом `Properties`, но этот способ более сложен.

МЕТОДЫ ПРОГРАММИРОВАНИЯ БАЗ ДАННЫХ

9

В этой главе – последней из тех, которые посвящены программированию, – я покажу некоторые специальные приемы работы с MySQL из программ на разных языках. В предыдущих главах речь шла о базовых способах взаимодействия с MySQL программ на языках PHP, Perl и Java, а сейчас мы поговорим скорее об общих принципах программирования баз данных, нежели о специфике MySQL: обсудим сохранение и выборку двоичных данных, создание простейшей поисковой машины, генерирование страниц результатов и вопросы безопасности.

Продемонстрированные в настоящей главе методы раскрывают, как мне кажется, наиболее типичные подходы к решению вышеупомянутых задач. Скорее всего, вы выработаете для себя несколько иные приемы, но изучение приведенного материала даст вам общее представление о теоретических основах.

В каждом примере будет использоваться один из рассмотренных выше языков (в основном PHP и Perl), но вам не составит труда реализовать описанные идеи на любом другом языке. (Если позволит время и будет достаточно много заявок, я размещу на сайте www.dmcinsights.com/mysql версии примеров и на других языках.)

Хранение и выборка двоичных данных

Очень часто приходится слышать вопрос о том, как хранить и извлекать двоичные данные в MySQL. К двоичным данным можно отнести изображения, файлы в формате PDF и многое другое. Двоичные данные противопоставляются базовым текстовым и числовым данным, которые обычно хранятся в базе. Для хранения двоичных данных следует завести колонку типа BLOB[1].

```
CREATE TABLE имя_таблицы(
id INT UNSIGNED NOT NULL AUTO_INCREMENT,
binary_item BLOB
);
```

MySQL поддерживает поля типа BLOB разных размеров: TINYBLOB (маленький), MEDIUMBLOB (средний) и BLOB (обычный), хотя в стандарте SQL они не определены. На самом деле тип BLOB – это то же самое, что тип TEXT, только первый чувствителен к регистру, а второй – нет.

Определив поле, в котором могут храниться двоичные данные, вы можете поместить туда значение при помощи функции LOAD_FILE, а выбрать значение – при помощи команды SELECT:

```
INSERT INTO имя_таблицы SET
⇨ image=LOAD_FILE('/путь/к/файлу.ext')

SELECT image FROM имя_таблицы
```

Другой способ – прочитать двоичный файл из программы и сохранить его в базе данных. Для примера я создам PHP-сценарий, который загружает файл, указанный в форме, и отображает его на Web-странице. Предполагается, что вы работаете

Следует ли хранить двоичные данные в базе?

Среди разработчиков баз данных не стихают споры о том, следует ли хранить двоичные данные в базе. Предлагаемая альтернатива состоит в том, чтобы сам файл хранился в файловой системе сервера, а в базе находилось только его имя. У каждого метода есть как достоинства, так и недостатки.

С одной стороны, хранение двоичных данных в базе позволяет делать их резервную копию вместе со всеми остальными данными. Кроме того, для их выборки не нужно никаких полномочий, кроме права доступа к базе. С другой стороны, для хранения и выборки двоичных данных нужно писать дополнительный код; при этом возможно некоторое снижение производительности.

В конечном счете решение зависит от разработчика и нужд конкретного приложения, но сам факт, что MySQL предоставляет выбор, не может не радовать. Рекомендую попробовать оба подхода и самостоятельно решить, какой для вас предпочтительнее.

[1] BLOB – Binary Large OBject (большой двоичный объект). – *Прим. переводчика.*

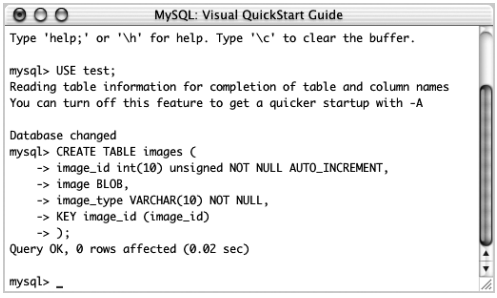

Рис. 9.1. Перед написанием примера я создал таблицу, пользуясь монитором mysql

с версией PHP 4.2 – последней на момент написания этой книги. При использовании более ранних версий, возможно, придется изменить имена некоторых переменных.

Хранение и извлечение двоичных данных

1. Создайте таблицу со следующей структурой (рис. 9.1):

```
CREATE TABLE images (
image_id INT(10) UNSIGNED NOT NULL
⇨ AUTO_INCREMENT,
image BLOB,
image_type VARCHAR(10) NOT NULL,
KEY image_id(image_id)
);
```

   Для примера я организовал новую таблицу images в базе данных test. Если вы решите изменить структуру этой таблицы, то должны будете соответствующим образом модифицировать запросы в приведенном ниже PHP-сценарии.

2. Создайте в текстовом редакторе новый PHP-сценарий (листинг 9.1):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD ⇨
XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/2000/
        ⇨ REC-xhtml1-20000126/DTD/
        ⇨ xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
⇨ 1999/xhtml">
<head>
   <title>Storing Images in MySQL
   ⇨ </title>
</head>
<body>
<?php
```

   Заголовок страницы – `Storing Images in MySQL` («Сохранение изображений в MySQL»).

На этой странице используется комбинация XHTML и PHP для отображения формы и обработки загруженного файла. Выше приведен стандартный пролог XHTML, за которым следует открывающий тэг PHP.

3. Проверьте, была ли форма отправлена:

```
if (isset($_POST['submit'])) {
```

Листинг 9.1. Сценарий store_binary.php предоставляет пользователю возможность выбрать изображение, которое требуется сохранить в базе данных

Листинг

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Storing Images in MySQL</title>
</head>
<body>
<?php

// ***** store_binary.php *****
// ***** Листинг 9.1 *****
// Этот сценарий сохраняет изображение в базе данных.

if (isset($_POST['submit'])) { // Если форма была отправлена...

  // Информация о доступе к базе данных:
  DEFINE (DB_USER, "имя_пользователя");
  DEFINE (DB_PASSWORD, "пароль");
  DEFINE (DB_HOST, "localhost");
  DEFINE (DB_NAME, "test");

  // Установить соединение с MySQL:
  $db_connection = mysql_connect (DB_HOST, DB_USER, DB_PASSWORD) or
  ⇨ die ('Could not connect to MySQL: ' . mysql_error());

  // Выбрать базу данных:
  mysql_select_db (DB_NAME) or die ('Could not select the database: '
  ⇨ . mysql_error());

  // Прочитать загруженный файл:
  $image = addslashes(fread(fopen($_FILES['имя_файла']['временное_имя'],
  ⇨ "r"), $_FILES['имя_файла']['размер']));
```

Поскольку один и тот же сценарий осуществляет и вывод формы, и ее обработку, мы должны выяснить, что именно нужно делать, – для этого и предназначен главный условный оператор. Его первая ветвь выполняется, когда форма отправлена (следовательно, POST-переменная `submit` имеет значение).

Для пущей безопасности здесь можно было бы проверить, действительно ли загружен файл (ведь можно отправить форму, не указав его).

Листинг 9.1. Сценарий store_binary.php предоставляет пользователю возможность выбрать изображение, которое требуется сохранить в базе данных (окончание)

Листинг

```
  // Сформировать запрос:
  $query = "INSERT INTO images VALUES (0, '$image', '{$_FILES['имя_файла']['тип']}')";

  // Выполнить запрос и сообщить о результате:
  if (mysql_query ($query)) {
           echo 'Image number ' . mysql_insert_id() . ' has been stored!';
  }  else {
           echo 'The image could not be stored in the database! ' . mysql_error();
  }

  // Закрыть соединение с базой данных:
  mysql_close();

} else { // Вывести форму:
?>
  <form action="store_binary.php" method="post"
               enctype="multipart/form-data">
  <input type="hidden" name="MAX_FILE_SIZE" value="200000" />
  Select a file to upload: <input type="file" name="имя_файла" />
  <br />
  <input type="submit" name="submit" value="Submit!" />
  </form>
  <?php
}
?>
</body>
</html>
```

4. Установите соединение с базой данных:

```
DEFINE (DB_USER, "имя_пользователя");
DEFINE (DB_PASSWORD, "пароль");
DEFINE (DB_HOST, "localhost");
DEFINE (DB_NAME, "test");
$db_connection = mysql_connect
⇨ (DB_HOST, DB_USER, DB_PASSWORD)
⇨ or die ('Could not connect to
⇨ MySQL: ' . mysql_error());
mysql_select_db (DB_NAME) or die
⇨ ('Could not select the
⇨ database: ' . mysql_error());
```

Если вы изучили главу 6, эта процедура вам уже знакома. Сообщение об ошибке звучит так: `Could not select the database` – «Не удалось выбрать базу данных». При желании можете поместить все параметры соединения в отдельный конфигурационный файл и включить его.

5. Прочитайте загруженный файл с изображением в строку:

```
$image =
⇨ addslashes(fread(fopen($_FILES
⇨ ['имя_файла']['временное_имя'],
⇨ "r"), $_FILES['имя_файла']
⇨ ['размер']));
```

Это самая важная строка во всем сценарии. Здесь загруженный файл (на который мы ссылаемся как на `$_FILES ['имя_файла']['размер']`) считывается в строковую переменную `$image`. Файл сначала открывается функцией `fopen()`. Открытый файл считывается функцией `fread()`, которой в качестве числа байтов, подлежащих чтению, передается размер всего файла, а затем пропускается через функцию `addslashes()`.

6. Отправьте запрос базе данных и сообщите о результате выполнения:

```
$query = "INSERT INTO images VALUES
⇨ (0, '$image', '{$_FILES
⇨ ['имя_файла'] ['тип']}')";
if (mysql_query ($query)) {
    echo 'Image number ' .
    ⇨ mysql_insert_id() .
    ⇨ ' has been stored!';
}  else {
    echo 'The image could not be
    ⇨ stored in the database! ' .
    ⇨ mysql_error();
}
```

Этот очень простой запрос всего лишь вставляет новую запись в таблицу images. В качестве значений полей указаны 0, вместо которого в автоинкрементный первичный ключ будет записано очередное значение; строка $image, содержащая собственно данные; наконец, тип изображения, извлеченный из MIME-типа загруженного файла. Можно было бы также сохранить в базе размеры изображения в пикселях.

Image number ... has been stored – «Изображение номер ... было сохранено».

The image could not be stored in the database! – «Изображение не удалось сохранить в базе данных».

По-настоящему осторожный программист мог бы перед попыткой вставки проверить, что переменная $image имеет значение (то есть файл был успешно прочитан).

7. Закройте соединение с базой данных и завершите главный условный оператор, добавив ветвь для вывода формы:

```
mysql_close();

} else {
?>
    <form action="store_binary.php"
    ⇨ method="post" enctype=
    ⇨ "multipart/form-data">
    <input type="hidden" name=
    ⇨ "MAX_FILE_SIZE" value=
    ⇨ "200000" />
    Select a file to upload: <input
    ⇨ type="file" name="имя_файла" />
    <br />
    <input type="submit" name="submit"
    ⇨ value="Submit!" />
    </form>
    <?php
}
```

В коде формы стоит отметить значение тэга `action` (он ссылается на тот же самый сценарий), тип кодировки `enctype` (только такой тип позволяет загружать файлы) и имя загружаемого файла – оно должно совпадать с указанным в коде обработки формы.

Следует также задать значение параметра `MAX_FILE_SIZE` равным максимальному приемлемому для вас размеру файла. В моем примере будут успешно сохраняться файлы размером до 97 Кб.

8. Завершите сценарий:

```
?>
</body>
</html>
```

9. Сохраните текст сценария в файле store_binary.php, загрузите его на Web-сервер и протестируйте в браузере (рис. 9.2, 9.3).

Рис. 9.2. При помощи этой HTML-формы пользователь выбирает на своем диске файл, который будет загружен в базу

Рис. 9.3. После того как изображение сохранено, сценарий сообщает, какой идентификатор ему присвоен. При выборке изображения из базы понадобится указать это число

10. Создайте еще один PHP-сценарий (листинг 9.2).
11. Проверьте, включен ли в URL идентификатор изображения:

```
if (isset($_GET['i'])) {
```

Листинг 9.2. Сценарий view_image.php (см. рис. 9.4)

Листинг

```
<?php

// ***** view_image.php *****
// ***** Листинг 9.2 *****
// Этот сценарий выводит изображение, хранящееся в базе данных.

if (isset($_GET['i'])) { // Если есть номер изображения, вывести его.

  // Информация о доступе к базе данных:
  DEFINE (DB_USER, "имя_пользователя");
  DEFINE (DB_PASSWORD, "пароль");
  DEFINE (DB_HOST, "localhost");
  DEFINE (DB_NAME, "test");

  // Установить соединение с MySQL:
  $db_connection = mysql_connect (DB_HOST, DB_USER, DB_PASSWORD) or
  ⇨ die ('Could not connect to MySQL: ' . mysql_error());

  // Выбрать базу данных:
  mysql_select_db (DB_NAME) or die ('Could not select the database: ' . mysql_error());

  // Извлечь само изображение и информацию о нем:
  $query = "SELECT image, image_type FROM images " . "WHERE image_id={$_GET['i']}";
  if ($query_result = mysql_query ($query)) {
             $image = mysql_fetch_array($query_result);
             header ("Content-type: $image[1]");
            echo $image[0];
  }

  // Закрыть соединение с базой данных:
  mysql_close();

}
?>
```

Адрес этой страницы записывается в виде `view_image.php?i=x`, где *x* – идентификатор изображения, присвоенный ему при сохранении в базе. Данный оператор как раз и проверяет, указан ли идентификатор в URL.

12. Установите соединение с базой данных:

```
DEFINE (DB_USER, "имя_пользователя");
DEFINE (DB_PASSWORD, "пароль");
DEFINE (DB_HOST, "localhost");
DEFINE (DB_NAME, "test");
$db_connection = mysql_connect
⇨ (DB_HOST, DB_USER, DB_PASSWORD)
⇨ or die ('Could not connect to
⇨ MySQL: ' . mysql_error());
mysql_select_db (DB_NAME) or die
⇨ ('Could not select the
⇨ database: ' . mysql_error());
```

`Could not connect to MySQL` – «Не удалось соединиться с MySQL».

`Could not select the database` – «Не удалось выбрать базу данных».

13. Извлеките изображение из базы данных:

```
$query = "SELECT image, image_type
⇨ FROM images WHERE
⇨ image_id={$_GET['i']}";
if ($query_result = mysql_query
⇨ ($query)) {
    $image = mysql_fetch_array
    ⇨ ($query_result);
    header ("Content-type: $image[1]");
   echo $image[0];
}
```

Этот запрос возвращает само изображение и информацию о его типе по значению поля `image_id` (полученному из параметра `i`, переданного в запросе). Если

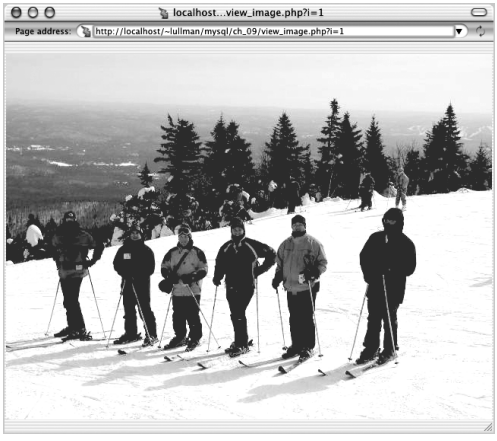

Рис. 9.4. Сценарий view_image.php извлекает изображение из базы данных и посылает его браузеру

запрос выполнен успешно, то переменной `$image` присваивается массив. Затем с помощью функции `header()` вы информируете браузер о типе отправленных данных, для чего посылаете ему MIME-тип, хранящийся в базе. Ну и напоследок отправляете само изображение.

14. Завершите сценарий:

```
 mysql_close();
}
?>
```

15. Сохраните текст сценария в файле view_image.php, загрузите его на Web-сервер и протестируйте в браузере (рис. 9.4). Не забудьте дописать в конец URL идентификатор изображения:

```
www.someaddress.com/view_image.php?i=1
```

П Чтобы вы могли сохранить двоичный объект в базе данных MySQL, соответствующий файл должен находиться непосредственно на сервере (удаленная загрузка невозможна).

П Пользователь MySQL, выполняющий функцию `LOAD_FILE()`, должен иметь полномочие `FILE` (это одна из причин, по которым я воспользовался другим методом).

С Сценарий view_image.php можно использовать для размещения изображения в любом месте Web-страницы. Для этого достаточно добавить в нужное место тэг `<img src="view_image.php?i=89">`.

С Если при загрузке или сохранении изображения в базе происходит ошибка, проверьте его размер (он должен быть меньше 100 000 байт). Если изображение показывается в браузере только частично, измените тип колонки на `BLOB` большего размера.

Создание поисковой машины

Один из самых полезных аспектов хранения информации в базе данных – это возможность осуществлять поиск. Поисковые машины (например, Google) применяются, главным образом, в Internet, но и в обычных приложениях и операционных системах могут принести пользу.

После того как информация помещена в базу данных, превратить последнюю в поисковую машину можно двумя способами:

- воспользоваться полнотекстовыми индексами и полнотекстовым поиском;
- вручную написать необходимые запросы к базе данных.

Для крупных приложений с большими объемами данных полнотекстовая индексация предпочтительнее. Мы обсудим эту тему в главе 11, а сейчас я продемонстрирую, как с помощью Perl можно построить простую поисковую машину.

Создание поисковой машины

1. Создайте базу данных (рис. 9.5):

```
CREATE DATABASE movies_db;
USE movies_db;
CREATE TABLE directors(
  director_id SMALLINT(5) UNSIGNED
 ⇨ NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(20),
  last_name VARCHAR(40),
  PRIMARY KEY(director_id),
  KEY last_name(last_name)
);
CREATE TABLE movies(
movie_id INT(10) UNSIGNED NOT NULL
⇨ AUTO_INCREMENT,
  title VARCHAR(100),
```

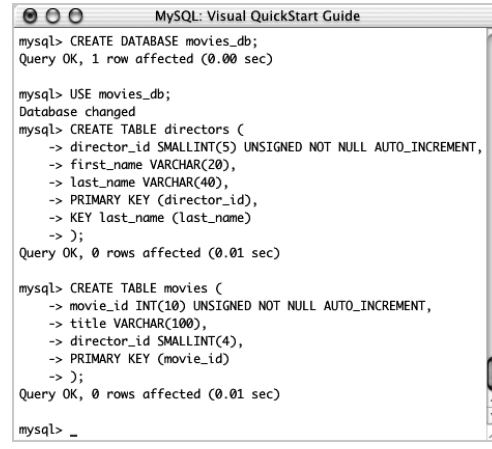

Рис. 9.5. Для примера я создал базу данных movies_db

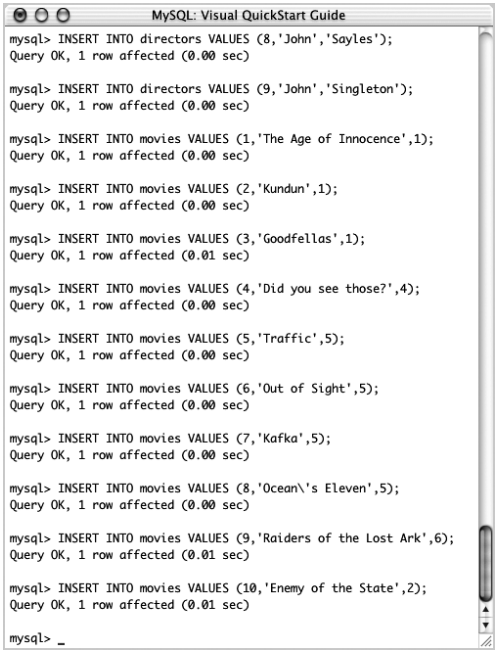

Рис. 9.6. Для выполнения поиска надо сначала ввести какие-то данные

```
    director_id SMALLINT(4),
    PRIMARY KEY(movie_id)
);
```

В этом примере работа ведется с базой movies_db, к которой мы будем обращаться и впоследствии.

2. Заполните таблицы (рис. 9.6).

   Если база данных еще пуста, добавьте в нее хотя бы несколько записей с помощью команды INSERT.

   Итак, база данных подготовлена, и мы приступаем к написанию Perl-сценария, который принимает от пользователя до двух слов и ищет их в базе данных movies_db.

3. Создайте в текстовом редакторе новый Perl-сценарий (листинг 9.3):

```
#!/usr/bin/perl
use DBI;
use strict;
```

Листинг 9.3. Perl-сценарий search_movies.pl показывает, как реализовать простую поисковую машину

Листинг

```
#!/usr/bin/perl

# Листинг 9.3, 'search_movies.pl'

# Включить используемые модули:
use DBI;
use strict;

# Соединиться с базой данных:
my $user;
my $password;
my $dbh = DBI->connect('DBI:mysql:movies_db;' .
                       mysql_read_default_file=/Users/lullman/.my.cnf',
                      $user, $password, {RaiseError => 1});

# Если соединение установлено, выполнить запрос:
if ($dbh) {
```

4. Установите соединение с базой данных:

```
my $user;
my $password;
my $dbh = DBI->connect ('DBI:mysql:
⇨ movies_db;mysql_read_default_
⇨ file=/Users/lullman/.my.cnf',
⇨ $user, $password, {RaiseError =>
⇨ 1});
```

Листинг 9.3. Perl-сценарий search_movies.pl показывает, как реализовать простую поисковую машину (продолжение)

Листинг

```
# Получить ключевые слова:
print "Enter a keyword to be searched: ";
my $term1 = <STDIN>;
print "Enter a second keyword to be searched: ";
my $term2 = <STDIN>;

# Создать предложение SQL:
my $sql = "SELECT CONCAT(directors.first_name, ' ', " . "directors.last_name)
⇨ AS Director, title AS Title " . "FROM directors, movies " .
⇨ "WHERE directors.director_id=movies.director_id";

# Добавить к предложению SQL ключевые слова:
if (length($term1) > 1) {
    chop($term1);
    $sql .= " AND ( (directors.last_name LIKE '%$term1%' OR " .
    ⇨ " directors.first_name LIKE '%$term1%' OR " .
    ⇨ " movies.title LIKE '%$term1%')";

    if (length($term2) > 1) {
        chop($term2);
        $sql .= " AND (directors.last_name LIKE '%$term2%' OR " .
        ⇨ "directors.first_name LIKE '%$term2%' OR " .
        ⇨ "movies.title LIKE '%$term2%')";
    }

    $sql .= ")";
}

# Выполнить запрос:
my $query = $dbh->prepare ($sql);
my @row;
my $n;
if (defined($query)) {
```

Согласно рекомендациям, приведенным в главе 7, информация о доступе к базе данных будет храниться в файле .my.cnf. Если хотите, можете «зашить» эти значения непосредственно в код.

5. Начните главный условный оператор:

```
if ($dbh) {
```

Если соединение удалось установить, сценарий продолжает работу.

6. Получите ключевые слова, по которым нужно произвести поиск:

```
print "Enter a keyword to be
⇨ searched: ";
my $term1 = <STDIN>;
```

Листинг 9.3. Perl-сценарий search_movies.pl показывает, как реализовать простую поисковую машину (окончание)

Листинг

```
        print "\n";
        $query->execute();
        for ($n=0; $n < $query->{'NUM_OF_FIELDS'}; $n++) {
             print @{$query->{'NAME'}}[$n];
            print "\t";
        }
        print "\n";
        while (@row = $query->fetchrow_array()) {
            foreach (@row) {
                print "$_ \t";
            }
            print "\n";
        }
    } else {
        print "The query was not executed because MySQL reported " . DBI->errstr() . ".
        ⇨ \n";
    }
    print "\n";

    # Завершить запрос и закрыть соединение с базой данных:
    $query->finish();
    $dbh->disconnect;

} else {
    print "Could not connect to the database! \n";
}
```

```
print "Enter a second keyword
⇨ to be searched: ";
my $term2 = <STDIN>;
```

`Enter a keyword to be searched` – «Введите ключевое слово для поиска».

`Enter a second keyword to be searched` – «Введите второе ключевое слово для поиска».

Таким образом пользователю дается возможность ввести до двух ключевых слов, поиск которых будет вестись в таблицах directors (режиссеры) и movies (фильмы). Можете усложнить сценарий, изменив входные данные (например, позволив пользователю указать таблицу или колонку, где должен осуществляться поиск).

7. Начните SQL-запрос:

```
my $sql = "SELECT CONCAT
⇨ (directors.first_name, ' ',
⇨ directors.last_name) AS
⇨ Director, title AS Title
⇨ FROM directors, movies
⇨ WHERE directors.director_id=
⇨ movies.director_id";
```

Этот запрос вернет фамилию режиссера и название фильма из каждой записи, удовлетворяющей критерию поиска. С помощью слова `AS` я изменил названия колонок на псевдонимы Director (режиссер) и Title (название). В дальнейшем вы увидите, для чего это нужно.

8. Добавьте к запросу ключевые слова:

```
if (length($term1) > 1) {
    chop($term1);
    $sql .= " AND ( (directors.
    ⇨ last_name LIKE '%$term1%'
    ⇨ OR directors.first_name
    ⇨ LIKE '%$term1%' OR
    ⇨ movies.title LIKE
    ⇨ '%$term1%')";
```

```
    if (length($term2) > 1) {
          chop($term2);
          $sql .= " AND (directors.
          ⇨ last_name LIKE
          ⇨ '%$term2%' OR
          ⇨ directors.first_name
          ⇨ LIKE '%$term2%' OR
          ⇨ movies.title LIKE
          ⇨ '%$term2%')";
    }

          $sql .= ")";
}
```

В базе данных movies_db есть три поля, где в принципе можно производить поиск: directors.first_name (режиссеры.имя), directors.last_name (режиссеры.фамилия) и movies.title (фильмы.название). К каждому полю применено условие `LIKE '%`*ключевое_слово*`%'`. Также будем считать, что если введены два слова, то поисковому запросу удовлетворяют только записи, содержащие оба.

При построении запроса вы сначала проверяете, заданы ли оба ключевых слова (не предполагая, что пользователь вообще что-то ввел), а затем отбрасываете последний символ каждого, если он совпадает с символом возврата каретки. После этого остается добавить нужные условия в уже сформированный запрос.

Эта часть сценария и лежит в основе поисковой машины. Тут вам предоставляется большой простор для различных изменений и улучшений. Можно, например, ограничить поиск конкретными колонками, разрешить ввод более двух ключевых слов или изменить логику поиска, то есть соединять условия связкой `OR` (ИЛИ) либо `AND` (И) по указанию пользователя.

9. Выполните запрос и выведите результаты:

```
my $query = $dbh->prepare ($sql);
my @row;
my $n;
if (defined($query)) {
   print "\n";
   $query->execute();
   for ($n=0; $n < $query->
 ⇨ {'NUM_OF_FIELDS'}; $n++) {
        print @{$query->{'NAME'}}[$n];
        print "\t";
   }
   print "\n";
   while (@row = $query->
   ⇨ fetchrow_array()) {
        foreach (@row) {
          print "$_ \t";
        }
        print "\n";
   }
} else {
   print "The query was not
   ⇨ executed because MySQL
   ⇨ reported " . DBI->errstr() .
   ⇨ ". \n";
}
print "\n";
```

Логика вывода результатов мало отличается от того, что мы делали в сценарии browse_table.pl из главы 7 (рис. 7.4). Я лишь слегка изменил формат, но все остальное вам уже знакомо.

`The query was not executed because MySQL reported` – «Запрос не был выполнен из-за ошибки MySQL».

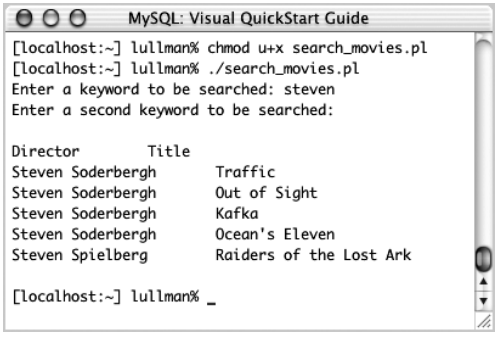

Рис. 9.7. По ключевому слову `steven` найдены все фильмы, снятые Стивеном Содербергом (Steven Soderbergh) и Стивеном Спилбергом (Steven Spielberg). Здесь же могли бы упоминаться фильмы, в названии которых есть имя Steven, а также снятые режиссером по фамилии Stevens

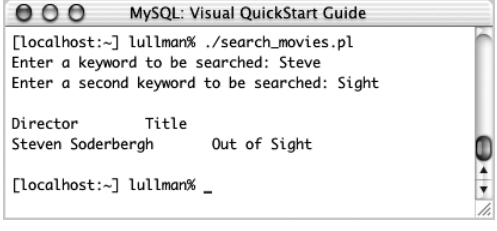

Рис. 9.8. Поиск по двум ключевым словам возвращает меньше записей

10. Освободите ресурсы MySQL и завершите сценарий:

```
$query->finish();
$dbh->disconnect;
} else {
    print "Could not connect to the
    ⇨ database! \n";
}
```

`Could not connect to the database!` – «Не удалось соединиться с базой данных».

Ветвь `else` выполняется, если не удалось установить соединение с базой данных (см. проверку переменной `$dbh`).

11. Сохраните сценарий в файле search_movies.pl (в том же каталоге, где находится файл .my.cnf, если вы им пользовались) и протестируйте его (рис. 9.7, 9.8).

П Одно из преимуществ полнотекстового поиска состоит в том, что он возвращает результаты, отсортированные по релевантности.

С Для повышения эффективности поиска по строковым колонкам постройте индексы по этим колонкам[1].

[1] Это не приведет к повышению эффективности, если в предикате `LIKE` указывать аргумент, начинающийся с метасимвола, например `LIKE '%Stevens%'`. Если же аргумент начинается с обычного символа (`LIKE 'Stevens%'`), некоторое ускорение действительно возможно. – *Прим. переводчика.*

Разбиение результатов поиска на страницы

Если вы имеете дело не с маленькими базами данных, предложения SELECT подчас возвращают десятки, сотни, а то и тысячи строк. Вы вряд ли захотите, чтобы все они отображались одновременно при выводе результатов (на Web-страницу или иным способом). Обычно в таких случаях отчет о результатах разбивается на страницы, скажем, по 25 записей.

Существуют два способа разбиения на страницы – выбор того или иного варианта зависит от используемого языка программирования и потребностей приложения:

- некоторой переменной присваивается ссылка на объект-запрос и передается между страницами (см. врезку «Использование ссылки на запрос»);
- запрос для формирования каждой страницы выполняется заново, причем номера начальной и конечной записей меняются (с помощью слова LIMIT).

Для большинства целей второго способа достаточно, и я продемонстрирую его на примере PHP-сценария. Он будет показывать все записи о фильмах в базе данных movies_db, отсортированные по названию.

Использование ссылки на запрос

Если язык программирования позволяет не закрывать соединение с базой данных при каждом запуске сценария (то есть поддерживает постоянные соединения), можно применить более простой подход.

Обычно результаты запроса представляют собой объект, который можно присвоить переменной. Записи из результирующего набора выбираются в цикле методом этого объекта. Но если такой объект может существовать и после завершения работы сценария, то его можно передавать между страницами и, следовательно, избежать повторного запроса к базе.

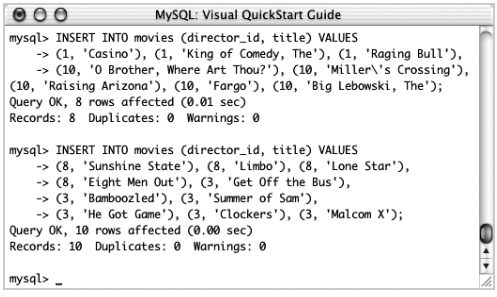

Рис. 9.9. Чтобы в полной мере продемонстрировать разбиение на страницы, я добавлю в базу данных еще несколько записей

Формирование страницы результатов

1. Создайте и заполните базу данных movies_db (рис. 9.9).

   Если вы уже сделали это (см. п. 1 предыдущей инструкции), можете либо пропустить этот шаг, либо добавить новые записи – и чем больше, тем лучше.

2. Создайте в текстовом редакторе новый PHP-сценарий (листинг 9.4). Заголовок страницы – `Browse the Movie Titles` (Вести поиск по названиям фильмов):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD ⇨
XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/2000/REC-
⇨ xhtml1-20000126/DTD/xhtml1-
⇨ transitional.dtd">
<html xmlns="http://www.w3.org/
⇨ 1999/xhtml">
<head>
    <title>Browse the Movie Titles
    ⇨ </title>
</head>
<body>
<?php
```

Листинг 9.4. Этот PHP-сценарий показывает, насколько просто разбить результат на страницы

Листинг

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
          "http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Browse the Movie Titles</title>
</head>
<body>
<?php

// ***** browse_movies.php *****
// ***** Листинг 9.4 *****
// Этот сценарий формирует страницы перечня фильмов в базе movies_db.
```

Листинг 9.4. Этот PHP-сценарий показывает, насколько просто разбить результат на страницы (продолжение)

Листинг

```
// Число записей на странице:
$display_number = 5;

// Информация о соединении с базой данных:
DEFINE (DB_USER, "имя_пользователя");
DEFINE (DB_PASSWORD, "пароль");
DEFINE (DB_HOST, "localhost");
DEFINE (DB_NAME, "movies_db");

// Установить соединение с MySQL:
$db_connection = mysql_connect (DB_HOST, DB_USER, DB_PASSWORD) or
      die ('Could not connect to MySQL: ' . mysql_error());

// Выбрать базу данных:
mysql_select_db (DB_NAME) or
      die ('Could not select the database: ' . mysql_error());

// Определить общее число записей:
if (isset($_GET['np'])) {
     $num_pages = $_GET['np'];
} else {
     $query = "SELECT CONCAT(directors.first_name, ' ', " .
                "directors.last_name) AS Director, title AS Title " .
               "FROM directors, movies " .
                "WHERE directors.director_id=movies.director_id " .
               "ORDER BY movies.title ASC";
     $query_result = mysql_query ($query) or die (mysql_error());
     $num_records = @mysql_num_rows ($query_result);

     if ($num_records > $display_number) {
          $num_pages = ceil ($num_records/$display_number);
     } else {
         $num_pages = 1;
     }
}

// Определить номер первой записи на странице:
if (isset($_GET['s'])) {
     $start = $_GET['s'];
} else {
     $start = 0;
}
```

3. Задайте число записей на одной странице:

   ```
   $display_number = 5;
   ```

   Мне удобно представить это число как значение переменной. Тогда для изменения размера страницы придется модифицировать только одну строку, хотя само значение встречается в тексте сценария несколько раз.

4. Установите соединение с базой данных:

   ```
   DEFINE (DB_USER, "имя_пользователя");
   DEFINE (DB_PASSWORD, "пароль");
   DEFINE (DB_HOST, "localhost");
   DEFINE (DB_NAME, "movies_db");
   ```

Листинг 9.4. Этот PHP-сценарий показывает, насколько просто разбить результат на страницы (продолжение)

Листинг

```
// Выбрать и вывести результаты:
$query = "SELECT CONCAT(directors.first_name, ' ', " .
            "directors.last_name) AS d, title " .
            "FROM directors, movies " .
             "WHERE directors.director_id=movies.director_id " .
             "ORDER BY movies.title ASC LIMIT $start, $display_number";
$query_result = mysql_query ($query) or die (mysql_error());

// Вывести все записи:
while ($row = mysql_fetch_array ($query_result, MYSQL_ASSOC)) {
  echo "{$row['d']} <i>{$row['title']}</i><br />\n";
}

// При необходимости сформировать ссылки на другие страницы:
if ($num_pages > 1) {

    echo '<hr width="50%" align="left" />';

    // Определить, какая страница текущая:
    $current_page = ($start/$display_number) + 1;

    // Если текущая страница не первая, добавить кнопку Previous (Предыдущая):
    if ($current_page != 1) {
        echo '<a href="browse_movies.php?s=' .
        ($start - $display_number) . '&np=' . $num_pages .
        '">Previous</a> ';
    }
```

```
$db_connection = mysql_connect
⇨ (DB_HOST, DB_USER, DB_PASSWORD)
⇨ or die ('Could not connect to
⇨ MySQL: ' . mysql_error());
mysql_select_db (DB_NAME) or
⇨ die ('Could not select the
⇨ database: ' . mysql_error());
```

`Could not connect to MySQL` – «Не удалось соединиться с MySQL».

`Could not select the database` – «Не удалось выбрать базу данных».

Можно было бы поместить эту информацию и в отдельный конфигурационный файл.

Листинг 9.4. Этот PHP-сценарий показывает, насколько просто разбить результат на страницы (окончание)

Листинг

```
  // Вывести номера всех страниц:
    for ($i = 1; $i <= $num_pages; $i++) {
         if ($i != $current_page) {
              echo '<a href="browse_movies.php?s=' .
              (($display_number * ($i - 1))) . '&np=' . $num_pages . '">' .
             $i . '</a> ';
         } else {
            echo $i . ' ';
         }
    }

     // Если страница не последняя, добавить кнопку Next (Следующая):
     if ($current_page != $num_pages) {
  echo '<a href="browse_movies.php?s=' . ($start + $display_number) .
               '&np=' . $num_pages . '">Next</a> ';
    }

}

// Освободить ресурсы:
mysql_free_result($query_result);
mysql_close();
?>
</body>
</html>
```

5. Определите, сколько страниц понадобится для вывода всех записей:

```
if (isset($_GET['np'])) {
    $num_pages = $_GET['np'];
} else {
    $query = "SELECT CONCAT
    ⇨ (directors.first_name, ' ', " .
    ⇨ "directors.last_name) AS
    ⇨ Director, title AS Title " .
    ⇨ "FROM directors, movies " .
    ⇨ "WHERE directors.director_id=
    ⇨ movies.director_id " .
    ⇨ "ORDER BY movies.title ASC";
    $query_result = mysql_query
    ⇨ ($query) or die
    ⇨ (mysql_error());
    $num_records = @mysql_num_rows
    ⇨ ($query_result);
   if ($num_records >
    ⇨ $display_number) {
        $num_pages = ceil
        ⇨ ($num_records/
        ⇨ $display_number);
    } else {
        $num_pages = 1;
}
}
```

Первым делом необходимо выяснить, сколько всего записей и, стало быть, сколько страниц понадобится для их вывода. Если в URL есть параметр `np` (общее число страниц), то эта величина уже была вычислена раньше, так что снова обращаться к ней нет нужды. Если же параметр `np` отсутствует, его придется вычислить.

Чтобы узнать число страниц, мы один раз запрашиваем базу данных и подсчитываем, сколько возвращено строк. Если их больше, чем помещается на страницу, то полученное значение делится на размер (высоту) страницы

и результат округляется до ближайшего большего целого. В противном случае будет достаточно одной страницы.

6. Определите номер первой записи на странице:

```
if (isset($_GET['s'])) {
    $start = $_GET['s'];
} else {
    $start = 0;
}
```

В самом запросе мы воспользуемся фразой `LIMIT` для выборки `$display_number` записей. Но сначала нужно выяснить, какое именно значение указать первым для параметра `LIMIT`: `0` (начать с первой записи) или что-то другое. Если в URL есть параметр *s* (номер начальной записи), то он и будет использоваться в запросе. В противном случае мы предполагаем, что начинать надо с самой первой записи.

7. Выведите все записи на данной странице:

```
$query = "SELECT CONCAT
⇨ (directors.first_name, ' ', " .
⇨ "directors.last_name) AS d,
⇨ title " . "FROM directors,
⇨ movies " . "WHERE directors.
⇨ director_id=movies.director_id " .
⇨ "ORDER BY movies.title ASC LIMIT
⇨ $start, $display_number";
$query_result = mysql_query
⇨ ($query) or die (mysql_error());
while ($row = mysql_fetch_array
⇨ ($query_result, MYSQL_ASSOC)) {
    echo "{$row['d']} <i>
    ⇨ {$row['title']}</i><br />\n";
}
```

Этот запрос в точности совпадает с тем, что использовался для определения числа записей и страниц, если не считать фразы LIMIT, как раз и позволяющей выбрать именно те записи, которые должны быть выведены на данную страницу. Так, при просмотре первой страницы пользователь увидит записи 1–5 (результат применения LIMIT 0, 5). Для второй страницы будет использоваться ограничение LIMIT 5, 5, для третьей – LIMIT 10, 5 и т.д.

Помимо выполнения запроса в этой части сценария производится вывод данных в очень простом формате.

8. Начните формировать ссылки на другие страницы результатов:

```
if ($num_pages > 1) {
    echo '<hr width="50%"
    ⇨ align="left" />';
    $current_page = ($start/
    ⇨ $display_number) + 1;
```

Если для представления результата требуется несколько страниц, то нужно вывести ссылки, позволяющие пользователю перемещаться между страницами. В данном случае мы сформируем ссылки на предыдущую и следующую страницу, а также номера всех возможных страниц. Для вычисления номера текущей страницы я прибавляю 1 к частному от деления первой записи на число записей. Например, для первой записи $start равно 0, а $display_number равно 5. Тогда $start / $display_number + 1 = 1. Для четвертой страницы $start равно 15 (это четвертая группа из пяти записей), а $display_number по-прежнему равно 5.

9. Создайте ссылку **Previous** (Предыдущая):

```
if ($current_page != 1) {
    echo '<a href="browse_
    ⇨ movies.php?s=' . ($start -
    ⇨ $display_number) . '&np=' .
    ⇨ $num_pages . '">Previous
    ⇨ </a> ';
}
```

Если текущая страница – не первая, то мы размещаем ссылку на предыдущую страницу. Ссылка имеет вид `browse_movies.php?s=x&np=y`, где `s` – номер первой записи на странице, а `np` – общее число страниц. Значение `x` вычисляется на основе номера текущей страницы и ее размера (проще говоря, начало предыдущей страницы отстоит от начала текущей на `$display_number` записей). Величина `y` равна уже известному к этому моменту полному числу страниц.

10. Сгенерируйте ссылки на номера страниц:

```
for ($i = 1; $i <= $num_pages;
⇨ $i++) {
    if ($i != $current_page) {
        echo '<a href="browse_
        ⇨ movies.php?s=' .
        ⇨ (($display_number *
        ⇨ ($i - 1))) . '&np=' .
        ⇨ $num_pages . '">' .
        ⇨ $i . '</a> ';
    } else {
        echo $i . ' ';
    }
}
```

Чтобы пользователь мог сразу перейти на любую страницу списка фильмов, ссылки формируются на каждую из них. Для этого в цикле от 1 до числа страниц создаются ссылки на каждую страницу, кроме текущей (ведь

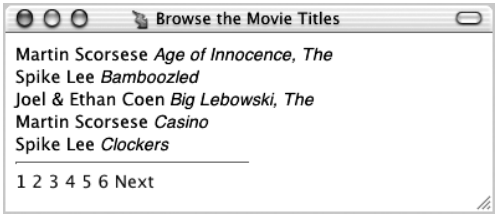

Рис. 9.10. При первом обращении пользователя к сценарию browse_movies.php выводятся информация о пяти фильмах, отсортированных по названиям, и ссылки на другие страницы

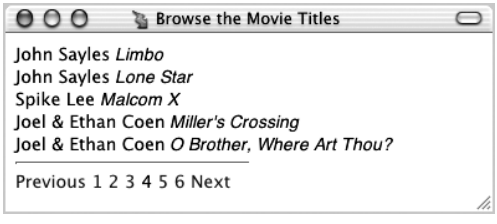

Рис. 9.11. Страница открывается при щелчке по ее номеру

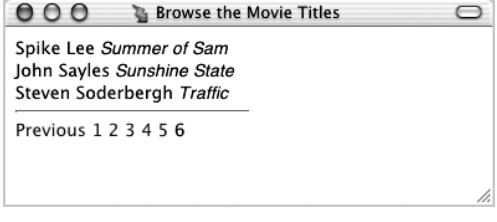

Рис. 9.12. На последней странице нет ссылки **Next**, а на первой – ссылки **Previous** (см. рис. 9.10)

ссылаться на ту страницу, где мы уже находимся, не имеет никакого смысла). Значение параметра `s` равно произведению `$i - 1` на размер (высоту) страницы. Для второй страницы `$i - 1` равно 1, а после умножения на 5 получаем 5, то есть вторая страница будет начинаться со второй группы из пяти записей.

11. Сформируйте ссылку **Next** (Следующая):

```
if ($current_page != $num_pages) {
    echo '<a href="browse_movies.
    ⇨ php?s=' . ($start +
    ⇨ $display_number) . '&np=' .
    ⇨ $num_pages . '">Next</a> ';
}
```

Ссылка **Next** будет присутствовать на каждой странице, кроме последней.

12. Завершите условный оператор, закройте соединение и HTML-страницу:

```
}
mysql_free_result($query_result);
mysql_close();
?>
</body>
</html>
```

13. Сохраните сценарий в файле browse_movies.php, загрузите его на Web-сервер и протестируйте в браузере (см. рис. 9.10–9.12).

Безопасность базы данных

Безопасность – это важная тема, которую нельзя обойти вниманием. База данных ценна, только если содержит корректные данные, а это в немалой степени зависит от мер предосторожности, принятых вами при разработке приложений.

Безопасность базы данных в контексте программирования подразумевает три условия. Во-первых, нужно правильно распределить полномочия, так чтобы сценарии, которые должны осуществлять только выборку, исполнялись от имени пользователя, имеющего лишь право `SELECT`, а уж никак не от имени root. Во-вторых, следует надежно хранить имена и пароли пользователей. В-третьих, введенные пользователями данные надлежит проверять перед их записью в базу. Последнее касается не только безопасности, но и целостности данных.

В табл. 9.1 приведены различные функции и операторы PHP и Perl, так или иначе связанные с безопасностью. Возможно, для проверки введенной пользователями информации лучше всего применять регулярные выражения, но это тема довольно сложная – явно не для начинающих разработчиков.

Таблица 9.1. Некоторые из наиболее распространенных средств повышения безопасности приложений для работы с базами данных

Функция	Язык	Назначение
`addslashes()`	PHP	Экранирует некоторые символы
`ereg()`	PHP	Применяет регулярное выражение к данным
`is_numeric()`	PHP	Проверяет, является ли значение числом
`mysql_escape_string`	PHP	Экранирует символы в строке из расчета на MySQL
`s//`	Perl	Применяет регулярное выражение к данным
`q{}`	Perl	Используется для экранирования одиночных кавычек

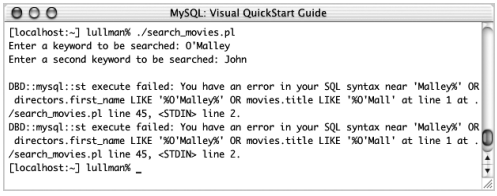

Рис. 9.13. Первоначальная версия сценария search_movies.pl завершается с ошибкой, если в ключевом слове есть апостроф

В следующем примере будет показано, как можно использовать регулярные выражения в Perl для экранирования одиночных кавычек. Если хотите узнать подробнее о регулярных выражениях, обратитесь к главе 11 или сами поищите нужные материалы в Internet.

Повышение безопасности Perl-сценария

1. Откройте файл search_movies.pl в текстовом редакторе.

   Один из недостатков прежней версии сценария (см. листинг 9.3) состоит в том, что он аварийно завершается, когда пользователь вводит ключевое слово, содержащее апостроф (рис. 9.13). MySQL выдает ошибку потому, что нарушается баланс одиночных кавычек в запросе.

2. После строки 28 в исходном сценарии добавьте такую строку (см. листинг 9.5):

   ```
   $term1 =~s/'/\\'/g;
   ```

   Это пример применения регулярных выражений в Perl. Смысл его в том, что каждая одиночная кавычка в значении переменной `$term1` экранируется, то есть заменяется последовательностью `\\'`. Два знака `\` нужны потому, что сам символ косой черты является специальным, поэтому первая косая черта экранирует вторую.

3. После строки 32 в исходном сценарии добавьте такую строку:

   ```
   $term2 =~s/'/\\'/g;
   ```

   Здесь то же самое регулярное выражение применяется ко второму ключевому слову.

4. Сохраните файл (я назвал новую версию search_movies2.pl) и протестируйте его (рис. 9.14).

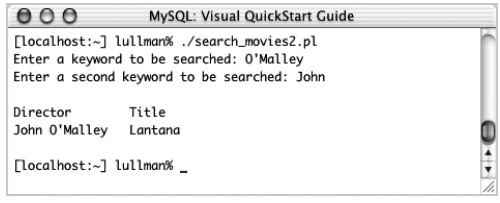

Рис. 9.14. В результате применения регулярных выражений сценарий стал работать более корректно (ср. рис. 9.13)

С Чтобы экранировать двойные кавычки в ключевых словах, воспользуйтесь такими регулярными выражениями:

```
$term1 =~s/"/\\"/g;
$term2 =~s/"/\\"/g;
```

Листинг 9.5. В модифицированной версии сценария search_movies.pl устранена причина возникновения ошибок SQL из-за неэкранированных одиночных кавычек

Листинг

```
#!/usr/bin/perl

# Листинг 9.5, 'search_movies2.pl'

# Включить используемые модули:
use DBI;
use strict;

# Установить соединение с базой данных:
my $user;
my $password;
my $dbh = DBI->connect('DBI:mysql:movies_db;' .
                       mysql_read_default_file=/Users/lullman/.my.cnf',
                      $user, $password, {RaiseError => 1});

# Если соединение установлено, выполнить запрос:
if ($dbh) {
    # Получить ключевые слова:
    print "Enter a keyword to be searched: ";
    my $term1 = <STDIN>;
    print "Enter a second keyword to be searched: ";
    my $term2 = <STDIN>;

    # Создать предложение SQL:
    my $sql = "SELECT CONCAT(directors.first_name, ' ', " .
              "directors.last_name) AS Director, title AS Title " .
```

Листинг 9.5. В модифицированной версии сценария search_movies.pl устранена причина возникновения ошибок SQL из-за неэкранированных одиночных кавычек (продолжение)

Листинг

```
                "FROM directors, movies " .
                  "WHERE directors.director_id=movies.director_id";

    # Добавить к предложению SQL ключевые слова:
    if (length($term1) > 1) {
        chop($term1);
        $term1 =~s/'/\\'/g;
        $sql .= " AND ( (directors.last_name LIKE '%$term1%' OR " .
                  " directors.first_name LIKE '%$term1%' OR " .
                  " movies.title LIKE '%$term1%')";

        if (length($term2) > 1) {
            chop($term2);
            $term2 =~s/'/\\'/g;
            $sql .= " AND (directors.last_name LIKE '%$term2%' OR " .
                      "directors.first_name LIKE '%$term2%' OR " .
                      "movies.title LIKE '%$term2%')";
        }

        $sql .= ")";
    }

    # Выполнить запрос:
    my $query = $dbh->prepare ($sql);
    my @row;
    my $n;
    if (defined($query)) {
        print "\n";
        $query->execute();
        for ($n=0; $n < $query->{'NUM_OF_FIELDS'}; $n++) {
            print @{$query->{'NAME'}}[$n];
            print "\t";
        }
        print "\n";
        while (@row = $query->fetchrow_array()) {
            foreach (@row) {
                print "$_ \t";
            }
            print "\n";
        }
    } else {
        print "The query was not executed because MySQL reported " .
              DBI->errstr() . ". \n";
    }
    print "\n";
```

Листинг 9.5. В модифицированной версии сценария search_movies.pl устранена причина возникновения ошибок SQL из-за неэкранированных одиночных кавычек (окончание)

Листинг

```
  # Завершить запрос и закрыть соединение с базой данных:
    $query->finish();
    $dbh->disconnect;

} else {
    print "Could not connect to the database! \n";
}
```

Администрирование MySQL

10

Теперь, когда вы постигли науку установки и запуска MySQL, создания баз данных и даже написания приложений на языках PHP, Perl и Java, пора приступить к изучению практики администрирования базы данных. Администрирование подразумевает несколько аспектов, начиная от резервного копирования и кончая повышением безопасности и производительности. К счастью, MySQL – настолько надежный продукт, что обслуживания почти не требуется, но если проблемы все же возникнут, они, как правило, легко решаются.

Демонстрацию различных приемов я буду проводить на примере операционных систем Windows 2000, Mac OS X и Red Hat Linux. Вы узнаете, что в новых версиях MySQL задачи, для которых раньше нужны были отдельные утилиты (например, mysqldump), теперь можно решить с использованием монитора mysql, при помощи специальных команд SQL. В таких случаях, прежде всего, будет рассматриваться специализированное приложение, а во врезках вы найдете информацию о командах SQL.

Файлы данных MySQL

По мере наполнения базы данных информацией все большую актуальность приобретает регулярное создание резервных копий. MySQL хранит всю информацию, в совокупности составляющую базу данных (как структуру, так и сами данные) в файлах на диске сервера. В зависимости от операционной системы, дистрибутива и конфигурационных параметров эти файлы обычно располагаются в следующих местах:

- C:\mysql\data;
- /usr/local/mysql/data;
- /usr/local/var;
- /var/lib/mysql.

Если вы не знаете, где на вашем компьютере размещаются файлы данных, откройте каталог с исполняемыми файлами MySQL – это может быть, например, /usr/local/mysql/bin или C:\mysql\bin – и запустите следующую команду, указав имя пользователя root и соответствующий пароль (рис. 10.1):

```
mysqladmin -u root -p variables
```

В этом каталоге MySQL создает подкаталоги для каждой базы данных, а уже внутри них – файлы базы данных. Например, если ваша база называется alpacas, то соответствующие ей файлы окажутся в каталоге C:\mysql\data\alpacas. Каждая таблица представлена тремя файлами (рис. 10.2):

- файл *имя_таблицы*.FRM содержит описание структуры таблицы: имена и типы колонок и т.п.;
- файл *имя_таблицы*.MYD содержит данные, хранящиеся в таблице;
- файл *имя_таблицы*.MYI содержит построенные над таблицей индексы.

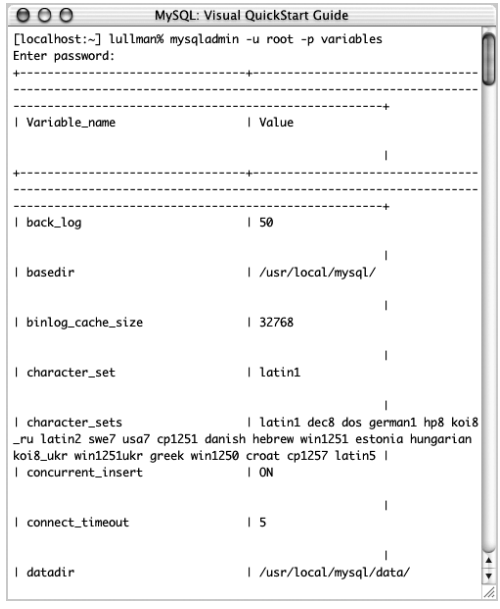

Рис. 10.1. Команда `mysqladmin variables` выводит список параметров MySQL, в том числе значение переменной `datadir`, указывающей на каталог с данными

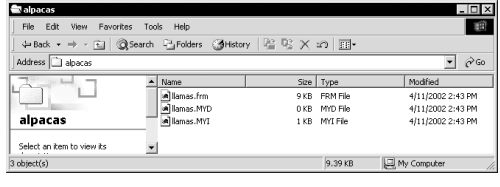

Рис. 10.2. Файлы базы данных – по три для каждой таблицы – хранятся в каталоге, который называется так же, как сама база

Рис. 10.3. При копировании файлов вручную следует предварительно остановить сервер MySQL

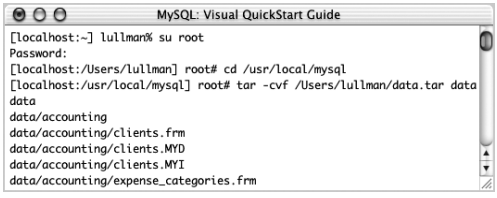

Рис. 10.4. В системах UNIX и Mac OS X для копирования файлов данных можно воспользоваться командой `tar`

Так хранятся все таблицы типа MyISAM. По умолчанию MySQL создает именно такие таблицы; ISAM – это сокращение от Indexed Sequential Access Method (индексно-последовательный метод доступа). В принципе для создания резервной копии достаточно просто скопировать все файлы базы.

Порядок копирования каталога с данными

1. Остановите сервер MySQL одной из следующих команд (рис. 10.3):
   - `mysqladmin -u root -p shutdown` (UNIX/Mac OS X);
   - `NET STOP mysql` (Windows, система MySQL запущена как сервис);
   - `/etc/rc.d/init.d/mysql stop` (Red Hat Linux).

   Эти методы останова сервера рассматривались в главе 2. Для запуска утилиты mysqladmin вы должны перейти в соответствующий каталог (например, /usr/local/mysql/bin или C:\mysql\bin), имя которого зависит от конфигурации системы.

   Перед копированием файлов базы очень важно остановить сервер, чтобы исключить возможность модификации файлов во время копирования.
2. Скопируйте файлы в новое место (см. рис. 10.4):

```
cd /путь/к/mysql
tar -cvf /путь/к/месту_назначения
⇨ /data.tar data
cd /путь/к/месту_назначения/
tar -xf data.tar
```

Эти команды будут работать в системах UNIX и Mac OS X (Windows предусматривает копирование файлов с помощью программы Windows Explorer.) Вторая команда создает архив, содержащий все файлы из каталога data, и сохраняет его в целевом каталоге. Четвертая команда разворачивает архив, помещая находящиеся в нем файлы в текущий каталог. Остальные две строки предназначены для перехода в каталог; измените пути в соответствии с конфигурацией вашей системы.

Для выполнения этих операций обычно нужны полномочия суперпользователя root.

3. Восстановите «личность» владельца файлов (рис. 10.5):

```
chown -R mysql data/*
```

Чтобы сервер MySQL мог производить запись в базу данных, необходимо изменить владельца скопированных файлов. Поскольку сервер mysqld обычно запускается от имени пользователя mysql, каталог data тоже должен принадлежать именно ему. Можно указать одновременно владельца и группу командой `chown -R mysql.mysql`.

Приведенные выше команды будут работать в большинстве операционных систем. В Windows вы должны щелкнуть правой кнопкой по каталогу и установить права доступа во всплывающем окне (его вид зависит от версии Windows). А вообще-то, учитывая, как работает система разграничения доступа в Windows, можете не возиться с этим.

4. Запустите MySQL.

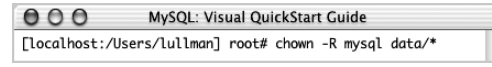

Рис. 10.5. Последний этап резервного копирования – восстановление надлежащего владельца

С При перемещении файлов базы данных в каталог, где уже есть одноименные файлы (то есть находится база данных с таким же именем), следует сначала удалить или переименовать старые файлы.

П В главном каталоге данных обычно тоже есть файлы, используемые MySQL. Это протоколы (ошибок и пр.). Подробнее о них будет рассказано ниже в этой главе.

С Для безопасного резервного копирования данных база должна быть заблокирована (об этом пойдет речь в главе 11). В качестве альтернативы можно остановить сервер mysqld, как показывает вышеприведенный пример.

Эквивалент в языке SQL, часть 1

В последних версиях MySQL можно выполнять резервное копирование (как это делает утилита mysqldump) непосредственно из монитора mysql. Вот какая команда для этого предназначена:

```
BACKUP TABLE имя_таблицы TO '/путь
⇨ /к/каталогу'
```

Этот запрос записывает в указанный каталог все данные из таблицы в виде файлов с расширениями .FRM, .MYD и .MYI. Можно сделать копии сразу нескольких таблиц, перечислив их имена через запятую.

Еще одну возможность предоставляет команда SELECT INTO:

```
SELECT * INTO
⇨ OUTFILE '/путь/к/каталогу
⇨ /имя_файла.txt'
⇨ FROM имя_таблицы
```

Таким образом можно создать и более специализированный запрос, результаты которого выводятся в файл. Но при этом сохраняются только данные без структуры таблицы.

Применяя SQL для резервного копирования, следует иметь в виду, что у MySQL должны быть права для создания файлов в указанном каталоге, а пользователь, пытающийся выполнить такой запрос, должен иметь полномочие FILE (кстати, это означает, что проводить подобную операцию лучше всего непосредственно на сервере). С учетом всего вышесказанного вы, возможно, предпочтете пользоваться утилитой mysqldump.

Резервное копирование базы данных

Есть и другой способ резервного копирования – использование утилиты mysqldump, входящей в состав MySQL. Вместо того чтобы копировать файл, mysqldump формирует файл, содержащий команды создания и заполнения таблиц. У этого метода есть еще одно преимущество – он позволяет легко перенести базу на другую операционную систему и даже в другую СУБД.

Утилита mysqldump вызывается следующим образом:

```
mysqldump [флаги] имя_базы_данных
```

Флаги можно не задавать вовсе. Чаще всего используются такие:

- `--add-drop-table` включает в выходной файл код для удаления таблицы перед ее созданием;
- `-d` показывает, что нужно вывести только структуру базы, но не сами данные;
- `-t` показывает, что нужно вывести только данные, но не структуру базы;
- `-T` позволяет указать каталог, в котором размещаются выходные файлы.

Последний флаг особенно полезен, так как в противном случае mysqldump направит весь вывод на экран, что не очень удобно. Например, следующая команда создаст в каталоге C:\backup по два файла для каждой таблицы из указанной базы:

```
mysqldump -u root -p -T C:\backup
⇨ имя_базы_данных
```

В принципе проще и безопаснее всего вызвать утилиту mysqldump с флагом `--opt`. Это предполагает установку стандартного набора параметров и создание одного большого файла вместо множества маленьких. В этом случае вы должны указать имя

файла, содержащего резервную копию, а не имя каталога.

```
mysqldump --opt имя_базы_данных
⇨ >/путь/к/файлу.sql
```

Давайте решим практическую задачу – сделаем резервную копию базы данных accounting, с которой мы работали на протяжении всей книги.

Использование утилиты mysqldump

1. Войдите в систему, используя командный интерпретатор.
2. Перейдите в каталог mysql/bin (или просто mysql, в зависимости от операционной системы):

   `cd /usr/local/mysql` (UNIX)
   `cd C:\mysql\bin` (Windows)

3. Создайте файл – резервную копию базы данных accounting (рис. 10.6). Для этого наберите

   ```
   mysqldump -u root -p --opt
   ⇨ accounting > /tmp/
   ⇨ accounting.sql
   ```

   или

   ```
   bin/mysqldump -u root -p --opt
   ⇨ accounting > /tmp/
   ⇨ accounting.sql
   ```

   Эта команда выведет в файл /tmp/accounting.sql последовательность команд SQL, воссоздающих структуру и данные всех таблиц из базы accounting. Если вы работаете с Windows, можете выбрать какой-нибудь другой каталог, например C:\backup (убедитесь предварительно, что он существует).

   Хотя для запуска mysqldump необязательно быть пользователем root, право доступа к копируемой базе у вас должно быть.

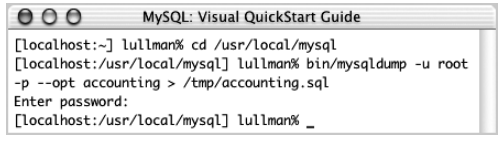

Рис. 10.6. При запуске утилиты mysqldump с флагом `--opt` создается один текстовый файл, содержащий копию всей базы

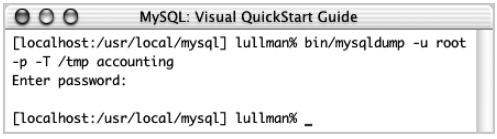

Рис. 10.7. Запуск утилиты mysqldump с флагом `-T` отличается от запуска с флагом `--opt` тем, что создается не один, а несколько файлов

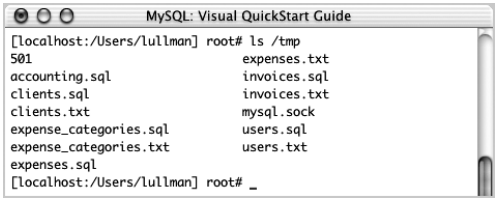

Рис. 10.8. Теперь в каталоге /tmp находятся две различных копии базы данных accounting

4. Сделайте резервную копию базы данных accounting в виде нескольких файлов (рис. 10.7):

```
mysqldump -u root -p -T /tmp
⇨ /accounting
```

При наличии флага `-T` создается по два текстовых файла для каждой таблицы: *имя_таблицы*.sql содержит команды для воссоздания структуры, а *имя_таблицы*. txt – команды для заполнения данными.

5. Просмотрите содержимое каталога /tmp (рис. 10.8):

```
ls /tmp
```

В этом каталоге теперь есть большой файл accounting.sql, содержащий копию всей базы, а также несколько файлов с расширениями .txt и .sql, представляющих отдельные таблицы.

С Не забывайте, что для использования утилиты mysqldump должен работать процесс mysqld.

С Чтобы познакомиться с результатами работы mysqldump, откройте какой-нибудь файл, созданный этой программой, в текстовом редакторе.

С Если вы часто пользуетесь утилитой mysqldump, то имеет смысл прочитать посвященный ей раздел руководства, чтобы лучше узнать обо всех ее возможностях.

С Вы можете сделать резервные копии всех имеющихся баз, воспользовавшись командой

```
mysqldump --opt --all-databases
⇨ > /путь/к/файлу.sql
```

П Одно из достоинств утилиты mysqldump заключается в том, что вы можете планировать ее регулярный запуск в удобное для вас время с помощью программы cron (в UNIX и Mac OS X) или соответствующего сервиса в Windows.

Командные файлы

Во всех примерах, рассматриваемых выше, запросы выполнялись непосредственно в мониторе mysql. Но MySQL позволяет посылать серверу запросы, не входя в монитор и не пользуясь API. Для этого предназначен так называемый *пакетный режим*. Вместо того чтобы набирать запросы вручную, вы можете поместить их в текстовый файл, а затем подать его на вход клиенту mysql:

```
bin/mysql -u имя_пользователя -p <
⇨ '/путь/к/файлу.txt'
```

Поскольку вы, скорее всего, хотите адресовать запросы конкретной базе данных, укажите ее имя с помощью флага `-D`:

```
bin/mysql -u имя_пользователя -p -D
⇨ имя_базы_данных < '/путь/к/файлу.txt'
```

В качестве примера я выполню посредством клиента mysql предложения SQL для создания таблиц в базе данных accounting (записанные в файл – см. предыдущий раздел).

Использование командных файлов

1. Создайте новую базу данных accounting (рис. 10.9):

   ```
   mysqladmin -u root -p create
   ⇨ accounting2;
   ```

   Поскольку при попытке создать таблицу с уже существующим именем возникнут проблемы, вам придется сначала создать новый экземпляр базы accounting. Если не хотите изменять уже имеющуюся базу с таким именем, назовите новую базу accounting2.

   Если вы запускали утилиту mysqldump с флагом `--add-drop-tables`, то SQL сначала удалит все таблицы, а затем воссоздаст их заново (пустыми).

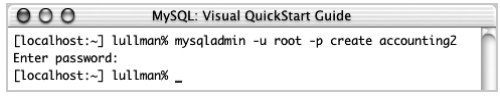

Рис. 10.9. Чтобы попрактиковаться в использовании командных файлов, не изменяя существующую базу данных, создайте новую с помощью утилиты mysqladmin

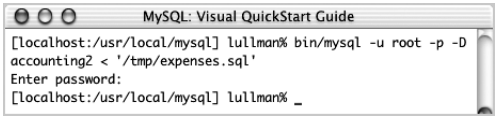

Рис. 10.10. Чтобы создать таблицу expenses, не вводя заново команды SQL, воспользуйтесь командным файлом, созданным утилитой mysqldump

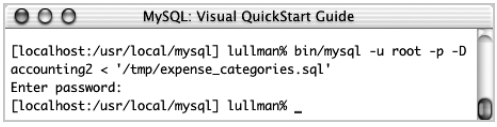

Рис. 10.11. Точно так же выполняются остальные командные файлы...

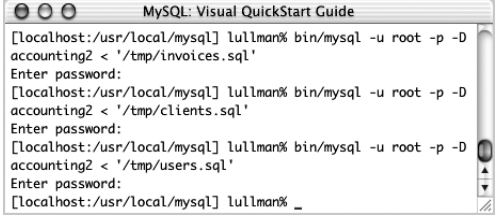

Рис. 10.12. ...и так до тех пор, пока не будут созданы все таблицы

2. Создайте таблицу expenses (рис. 10.10):

```
bin/mysql -u root -p -D accounting2
⇨ < '/tmp/expenses.sql'
```

Файл expenses.sql, созданный утилитой mysqldump, содержит команду `CREATE TABLE expenses...`. Подача этого файла на вход mysql избавляет вас от необходимости набирать команду вручную.

3. Создайте таблицу expense_categories (рис. 10.11):

```
bin/mysql -u root -p -D accounting2
⇨ < '/tmp/expense_categories.sql'
```

Для создания очередной таблицы используется та же команда, что и выше, только указано другое имя файла.

4. Создайте остальные таблицы (рис. 10.12):

```
bin/mysql -u root -p -D accounting2
⇨ < '/tmp/invoices.sql'
bin/mysql -u root -p -D accounting2
⇨ < '/tmp/clients.sql'
bin/mysql -u root -p -D accounting2
⇨ < '/tmp/users.sql'
```

Наконец-то все таблицы созданы – теперь можно приступать к их заполнению. Для этого пригодится утилита mysqlimport (см. следующий раздел).

П Еще одно полезное свойство пакетного режима mysql состоит в том, что запуск команды можно планировать с помощью cron (UNIX) или аналогичного сервиса в Windows.

С Если вы хотите сохранить протокол работы командного файла в другом файле, перенаправьте стандартный вывод. Так, команда

```
bin/mysql -u имя_пользователя -p
⇨ < '/путь/к/входному_файлу.txt' >
⇨ '/путь/к/выходному_файлу.txt'
```

сохранит протокол в файле */путь/к/выходному_файлу.txt*.

С Чтобы наблюдать за результатом исполнения командного файла, включите в командную строку конвейер `| more`. Следующая команда будет постранично выводить результаты на экран (для перехода к очередной странице нажимайте клавишу пробела, а для прекращения работы – клавишу **Q**):

```
bin/mysql -u имя_пользователя -p
⇨ < '/путь/к/файлу.txt' | more
```

Импорт данных

Итак, делать резервную копию базы вы умеете. Теперь надо бы научиться восстанавливать базу из имеющейся копии. Если вы скопировали все файлы из каталога mysql/data, то, поместив их обратно, восстано вите информацию (это было показано в разделе «Файлы данных MySQL»). Если же для копирования вы пользовались утилитой mysqldump, то выполнить обратную операцию поможет утилита mysqlimport, которая помещает данные из текстового файла в таблицу.

Утилита mysqlimport вызывается следующим образом:

```
mysqlimport -u имя_пользователя -p
⇨ имя_базы_данных '/путь/к/файлу.sql'
```

При этом предполагается, что, во-первых, таблица, куда вставляются данные, уже существует, а во-вторых, текстовый файл с данными называется так же, как таблица.

В качестве примера я продемонстрирую импорт данных в базу accounting, для которой в предыдущем разделе была создана резервная копия.

Порядок использования утилиты mysqlimport

1. Войдите в систему, используя командный интерпретатор.
2. Перейдите в каталог mysql/bin (или mysql, в зависимости от операционной системы):

   `cd /usr/local/mysql` (UNIX)

   `cd C:\mysql\bin` (Windows)

Эквивалент в языке SQL, часть 2

Импортировать данные в базу можно многими способами: например, вызвать утилиту mysqlimport, воспользоваться командными файлами или выполнить команды SQL из монитора mysql. В последнем случае пригодятся две команды SQL: `RESTORE` и `LOAD DATA`.

Команда `RESTORE` противоположна `BACKUP` и применяется для восстановления таблицы из файлов. Вот ее синтаксис:

```
RESTORE TABLE имя_таблицы
⇨ FROM 'путь/к/каталогу'
```

Команда `RESTORE`, как и `BACKUP`, может завершиться неудачно, если MySQL не имеет прав для копирования файла из одного каталога в другой (например, из /tmp в /usr/local/mysql/data/accounting).

Команда `LOAD DATA INFILE` противоположна команде `SELECT ... INTO OUTFILE`:

```
LOAD DATA INFILE '/путь/к/файлу.txt'
⇨ INTO TABLE имя_таблицы
```

Для выполнения `LOAD DATA` вам нужно полномочие `FILE`, а если вы собираетесь загружать файл с клиентской машины на удаленный сервер, то монитор mysql следует запускать с флагом `--local-infile`.

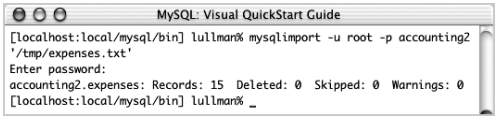

Рис. 10.13. Утилита mysqlimport импортирует в таблицу данные, которые содержит текстовый файл, в формате с разделителями – символами табуляции

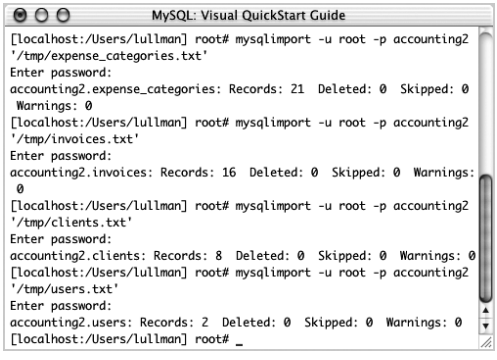

Рис. 10.14. Утилита mysqlimport информирует о результатах выполнения каждой операции, выводя число добавленных записей или сообщение об ошибке

3. Импортируйте записи в таблицу expenses (рис. 10.13):

```
mysqlimport -u root -p accounting2
⇨ '/tmp/expenses.txt'
```

Эта команда считывает записи из файла expenses.txt (созданного утилитой mysqldump) и вставляет их в таблицу expenses.

4. Импортируйте данные в другие таблицы (рис. 10.14):

```
mysqlimport -u root -p accounting2
⇨ '/tmp/expense_categories.txt'
mysqlimport -u root -p accounting2
⇨ '/tmp/invoices.txt'
mysqlimport -u root -p accounting2
⇨ '/tmp/clients.txt'
mysqlimport -u root -p accounting2
⇨ '/tmp/users.txt'
```

Подкорректируйте команды, если вы выгружали базу в другой каталог, и помните, что всякий раз вам придется вводить пароль пользователя root.

П Вышеописанные действия можно упростить, воспользовавшись командным файлом accounting.sql, который содержит все необходимые команды SQL. Но тогда вы бы не познакомились с утилитой mysqlimport!

П Утилита mysqlimport – это всего лишь реализация команды `LOAD DATA INFILE` без использования монитора mysql.

С Чтобы гарантированно очистить таблицу перед вставкой новых записей, вызовите mysqlimport с флагом `-d`.

Обслуживание базы данных

В задачи администратора, помимо резервного копирования, входит контроль целостности данных. Любой файл на сервере может быть запорчен, и файлы базы данных – не исключение. (Действительно, в ранних версиях Linux была ошибка, из-за которой файлы MySQL повреждались.) Чтобы проверить и при необходимости восстановить запорченные файлы, воспользуйтесь утилитой myisamchk:

```
myisamchk [флаги] имя_файла_таблицы
```

Эта программа отличается от рассмотренных выше, в частности, тем, что в качестве аргумента указывается физическое имя файла, содержащего индексы, например:

```
/usr/local/mysql/data/имя_базы_данных/⇨
имя_таблицы.MYI
```

или

```
C:\mysql\data\имя_базы_данных\
⇨ имя_таблицы.MYI
```

Кроме того, myisamchk работает только с таблицами типа MyISAM (о других типах таблиц рассказывается в главе 11).

У программы myisamchk три основных параметра:

- `-a` – анализировать таблицу;
- `-r` – «ремонтировать» таблицу;
- `-e` – выполнить «капитальный ремонт».

Я советую запускать `myisamchk -a` часто, а `myisamchk -r` – время от времени или в случае, когда это рекомендует сделать `myisamchk -a`. При возникновении серьезных проблем запустите команду `myisamchk -e`, которая обеспечивает более тщательную проверку, но и работает дольше. Используемый флаг и периодичность запуска

Эквивалент в языке SQL, часть 3

И снова вы можете воспользоваться средствами языка SQL вместо специализированной утилиты. Для быстрого анализа таблицы введите предложение

```
CHECK TABLE имя_таблицы;
```

В отличие от myisamchk этот запрос применим к таблицам типа MyISAM и InnoDB.

Уровень тщательности контроля можно изменить, выполнив команду

```
CHECK TABLE имя_таблицы EXTENDED;
```

Если команда `CHECK TABLE` сообщает о наличии проблемы, воспользуйтесь командой

```
REPAIR TABLE имя_таблицы EXTENDED;
```

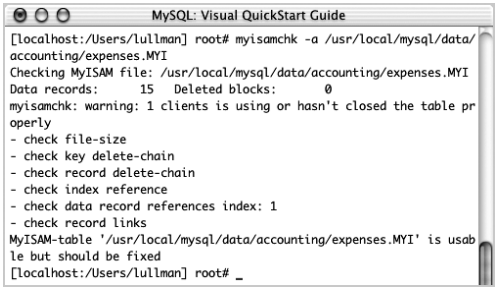

Рис. 10.15. Утилита myisamchk сообщает о состоянии таблиц типа MyISAM

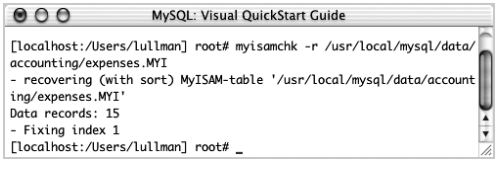

Рис. 10.16. Если myisamchk сообщает об ошибках, следует вновь запустить эту утилиту для «ремонта» таблицы

утилиты зависят от активности обращений к вашей базе и от того, как часто изменяются ее структура или данные.

Порядок использования утилиты myisamchk

1. Войдите в систему, используя командный интерпретатор.
2. Сделайте резервную копию базы, как описывалось выше.

   При выполнении прямых операций над файлами базы данных с помощью утилиты myisamchk всегда стоит заранее сохранить все, что возможно. Для резервного копирования вы вправе применить любой из рассмотренных выше подходов.
3. Проведите ускоренный анализ таблицы expenses (рис. 10.15):

   ```
   myisamchk -a /usr/local/mysql/data/
   ⇨ accounting/expenses.MYI
   ```

   Таким образом проверяется таблица expenses из базы данных accounting. В зависимости от операционной системы и конфигурации может потребоваться также указание флагов `-u` *`имя_пользователя`* `-p` (чтобы соединение с MySQL было установлено от имени пользователя root).
4. При обнаружении ошибок «отремонтируйте» таблицу (рис. 10.16):

   ```
   myisamchk -r /usr/local/mysql/data/
   ⇨ accounting/expenses.MYI
   ```

   Иными словами, если при запуске с флагом `-a` утилита myisamchk диагностировала ошибки (как показано на рис. 10.15), исправьте их, запустив ее же с флагом `-r`.

5. Повторите действия, описанные в пп. 3–4, применительно к другим таблицам (рис. 10.17):

   ```
   myisamchk -a /usr/local/mysql/data/
   ⇨ accounting/*.MYI
   ```

   Чтобы проанализировать сразу все таблицы в базе, воспользуйтесь метасимволом *. При этом myisamchk последовательно проверит каждую таблицу. Если будут найдены ошибки, запустите ту же утилиту с флагом -r или -e для поврежденных таблиц.

6. Перезагрузите сервер (рис. 10.18).

   Это гарантирует, что mysqld воспримет проделанные исправления.

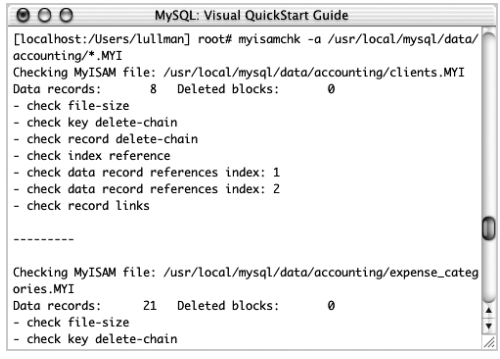

Рис. 10.17. Для проверки всех таблиц из базы укажите *.MYI в качестве имени файла

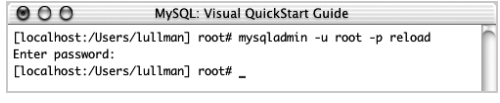

Рис. 10.18. После устранения ошибок в таблицах сервер следует перезагрузить

С В более ранних версиях MySQL использовались таблицы типа ISAM, которые были представлены файлами с расширениями .ISM и .ISD. Такие таблицы следует проверять утилитой isamchk, а не myisamchk.

С При запуске myisamchk с флагом -d выводится статистическая информация о таблицах. Если вы увидите большое число удаленных блоков, выполните команду myisamchk -r.

С Для запуска утилиты myisamchk с целью проверки всех таблиц во всех базах данных наберите предложение myisamchk [*флаги*] */путь/к/данным/*/*.MYI.

П Как и любую другую программу, myisamchk можно регулярно запускать с помощью планировщика cron или аналогичного сервиса в Windows.

Повышение производительности

При проектировании MySQL изначально ставилась задача создать быструю и надежную базу данных. И все-таки имеются дополнительные способы повышения производительности.

Очевидный путь – модернизировать аппаратные средства сервера. Потенциально «узким местом» может быть скорость соединения между клиентом и сервером (например, через Internet). Быстродействие процессора, объем оперативной памяти и типы дисков – все это влияет на производительность MySQL. Поскольку MySQL главным образом читает и записывает информацию в файлы на диске, медленно работающая аппаратура способна существенно затормозить эти процессы.

Второй способ «подстегнуть» СУБД связан с настройкой mysqld. Варьировать можно, например, следующие параметры:

- `key_buffer` (объем памяти, выделяемой для индексов);
- `max_connections` (сколько соединений разрешается устанавливать одновременно);
- `table_cache` (буфер для хранения таблиц).

Более подробные сведения об этих и других параметрах можно найти в документации по MySQL. Не забывайте, что все параметры, заданные для программы safe_mysqld, без изменения передаются демону mysqld.

Третья возможность повышения производительности – регулярный запуск утилиты myisamchk. Помимо анализа и «ремонта» таблиц (см. предыдущий раздел) ее мож-

но использовать для их оптимизации, указав флаг `--sort-index`. Того же результата вы достигнете, выполнив команду `OPTIMIZE` в мониторе mysql.

Оптимизация таблиц

1. Войдите в монитор mysql и выберите базу данных accounting:

   ```
   mysql -u имя_пользователя -p
   USE accounting;
   ```

   На протяжении всей этой главы я подчеркивал, что многие административные задачи можно решать как с помощью специализированных утилит, так и с помощью команд SQL. В начале этого раздела будет продемонстрирован второй подход, а затем я покажу, как сделать то же самое, используя программу myisamchk.

2. Оптимизируйте таблицу expenses (рис. 10.19):

   ```
   OPTIMIZE TABLE expenses;
   ```

   После выполнения этого запроса MySQL выведет информацию о состоянии таблицы.

3. Выйдите из монитора mysql:

   ```
   exit
   ```

4. Запустите утилиту myisamchk для всех таблиц (рис. 10.20):

   ```
   myisamchk --sort-index /usr/local/
   ⇨ mysql/data/accountinf/*.MYI
   ```

   Программа myisamchk удобна тем, что позволяет оптимизировать сразу все таблицы. Но не забывайте, что она работает только с таблицами типа MyISAM.

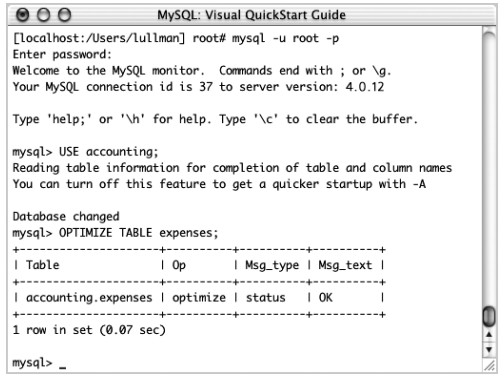

Рис. 10.19. Оптимизация таблиц выполняется по команде `OPTIMIZE`

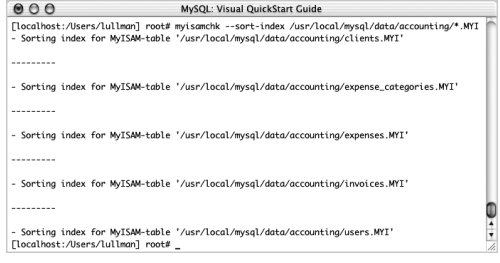

Рис. 10.20. Оптимизировать сразу все таблицы и даже все базы данных можно с помощью утилиты myisamchk

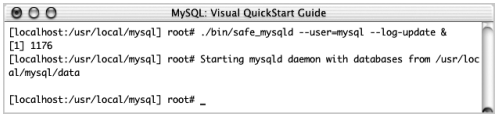

Рис. 10.21. Запуск mysqld с флагом `--log-update` включает режим протоколирования всех изменений в базах данных

Протоколирование операций MySQL

Любой администратор базы данных должен быть знаком с файлами протоколов. В MySQL есть два типа таких файлов: протокол ошибок сервера и протокол изменений данных.

Протокол ошибок располагается в каталоге данных и называется hostname.err, то есть обычно речь идет о файле /usr/local/mysql/data/localhost.err или C:\mysql\data\mysql.err. Здесь фиксируются все ошибки, обнаруженные программой, – прежде всего, те, что произошли во время запуска mysqld. Поэтому просмотр протокола ошибок может существенно упростить поиск и устранение проблем эксплуатации сервера.

В протоколе изменений фиксируются все изменения, внесенные в любую базу данных. Другими словами, полностью отслеживаются предложения SQL, приведшие к модификации структуры или содержания таблиц. В некотором смысле такой протокол можно назвать резервной копией базы данных.

Включение и просмотр протоколов

1. Остановите mysqld.

   Поскольку вам предстоит изменить параметры сервера (точнее, включить протоколирование изменений), придется остановить его и запустить заново. Если вы не помните, как останавливать mysqld, обратитесь к главе 2.

2. Перезапустите mysqld с флагом `--log-update` (рис. 10.21):

```
./bin/safe_mysqld --user=mysql
⇨ --log-update &
```

Флаг `--log-update` сообщает программе mysqld, что она должна протоколировать все действия, приведшие к изменению структуры или содержания любой таблицы во всех базах данных. По умолчанию этот режим отключен.

3. Войдите в монитор mysql и как-нибудь измените таблицу accounting (рис. 10.22), например:

```
USE accounting;
INSERT INTO expense_categories
VALUES(NULL, 'Employee Benefits');
```

Сервер mysqld протоколирует только те запросы, которые приводят к изменениям. Поэтому, чтобы увидеть результат, нужно ввести запрос типа `INSERT`, `UPDATE`, `DELETE` или `ALTER` (запросы `SELECT` не протоколируются). Что конкретно вы сделаете, не столь важно.

4. Выйдите из монитора и просмотрите протокол в текстовом редакторе (рис. 10.23):

```
exit
vi /usr/local/mysql/data/localhost.001
```

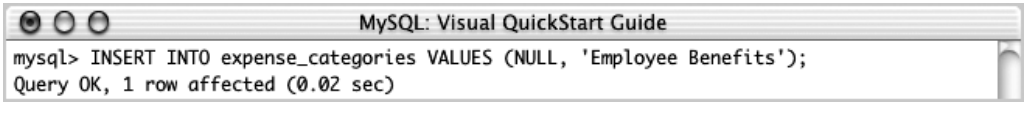

Рис. 10.22. Чтобы увидеть, как работает протоколирование изменений, выполните какие-нибудь запросы, модифицирующие базу

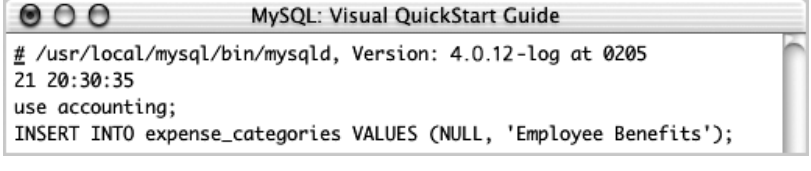

Рис. 10.23. В протоколе изменений localhost.001 фиксируются такие запросы, как `USE` и `INSERT`

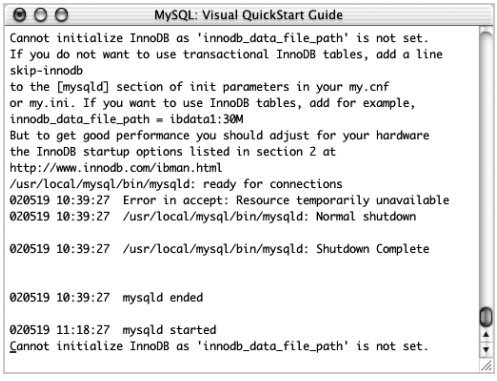

Рис. 10.24. В протокол ошибок localhost.err записываются сообщения, выданные сервером

Все протоколы изменений хранятся в каталоге data и называются по принципу *имя_хоста.номер*. Первому по порядку файлу присвоено имя `localhost.001`. Просмотреть его вы сможете в любом текстовом редакторе, например vi (UNIX или Mac OS X) или Notepad (Windows). Для выхода из редактора vi наберите команду `:q`.

5. Просмотрите файл localhost.err (см. рис. 10.24):

```
vi /usr/local/mysql/data/
⇨ localhost.err
```

Повторите действия, описанные в п. 4, для просмотра содержимого протокола ошибок. В нем уже должна быть какая-то информация.

С Не переусердствуйте с протоколами изменений MySQL, так как их ведение снижает производительность сервера.

П Протоколы изменений имеют тенденцию очень быстро расти, особенно если база данных активно используется.

С Чтобы восстановить состояние базы данных на основе протокола изменений, выполните его как командный файл:

```
mysql -u root -p <localhost.001
```

Безопасность

В заключение этой главы я хочу вкратце осветить некоторые вопросы безопасности, возникающие при использовании и администрировании баз данных. На эту тему уже многое говорилось в предыдущих главах, но имеет смысл подытожить вышеприведенную информацию. Итак, для обеспечения безопасности пользуйтесь следующими советами:

- не разрешайте анонимным пользователям подключаться к MySQL;
- всегда запрашивайте пароль при подключении;
- ограничивайте диапазон IP-адресов, с которых пользователи могут соединяться с сервером (какие бы неудобства это ни создавало);
- запускайте процесс mysqld не от имени суперпользователя root (в системах UNIX). Загружать MySQL следует командой `safe_mysqld --user=`*`имя_пользователя`* `&`. Обычно администраторы создают пользователя mysql специально для этой цели;
- при хранении в базе конфиденциальной информации, а особенно паролей, шифруйте ее с помощью функций `PASSWORD()` или `ENCODE()` – см. главу 5;
- выделяйте каждому пользователю минимальный набор полномочий;
- при запуске mysqld (опосредованно через `safe_mysqld`) задавайте флаги `--secure` и `--skip-name-resolve`. Смысл того и другого не вполне очевиден – пользуйтесь указанными флагами, только если хорошо понимаете, для чего они нужны;
- проверяйте данные, которые введены пользователем, перед их сохранением в базе;

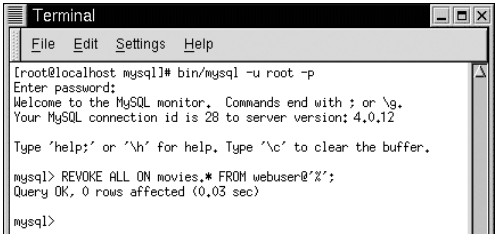

Рис. 10.25. Команда REVOKE, противоположная команде GRANT, предоставляет простейшую возможность отозвать полномочия, ранее предоставленные пользователю

- ограничьте пользователя root только правом доступа с локального компьютера;
- удалите базу данных test, к которой по умолчанию может обратиться любой пользователь;
- удаляйте пользователей, которым следует запретить доступ к данным, по нижеприведенной схеме.

Лишение пользователя полномочий

1. Войдите в систему, используя командный интерпретатор.
2. Перейдите в каталог mysql/bin (или mysql, в зависимости от операционной системы):

   `cd /usr/local/mysql` (UNIX)

   `cd C:\mysql\bin` (Windows)

3. Войдите в монитор mysql:

   `mysql -u` *`имя_пользователя`* `-p`

4. Отзовите полномочия пользователя (рис. 10.25):

   ```
   REVOKE ALL ON MOVIES.*
   ⇨ FROM webuser@'%';
   ```

   Таким образом пользователю webuser, созданному в главе 2, полностью запрещается доступ к базе данных movies. Если у вас нет пользователя или базы данных с таким именем, откорректируйте предложение в соответствии со своими настройками.

5. Выйдите из mysql и перезагрузите полномочия с помощью утилиты mysqladmin:

   ```
   exit
   mysqladmin -u root -p reload
   ```

Если вы добавляете или удаляете пользователя, то изменения вступают в силу только после перезагрузки полномочий.

6. Проверьте, может ли пользователь получить доступ к базе данных (рис. 10.26).

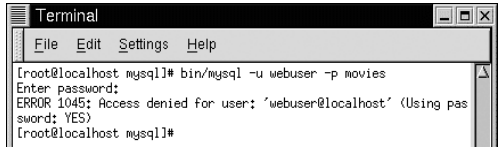

Рис. 10.26. После того как вы отзовете полномочия пользователя и произведете перезагрузку, он уже не сможет обратиться к базе данных

C Вместо того чтобы выходить из монитора и перезагружать MySQL, вы можете ввести в действие новые полномочия с помощью команды SQL `FLUSH PRIVILEGES`.

ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ

11

В этой главе мы рассмотрим несколько не связанных друг с другом тем, которые начинающему пользователю MySQL поначалу могут показаться неинтересными. Однако по мере накопления опыта вы будете все чаще обращаться к описанным здесь приемам. Я расскажу о таблицах типа InnoDB, об использовании транзакций, о регулярных выражениях и о выполнении полнотекстового поиска.

Некоторые из описываемых технологий появились в MySQL сравнительно недавно или будут иметь полноценную поддержку только в следующих версиях. Последнее относится к таблицам типа InnoDB, вместе с которыми в MySQL вошли некоторые черты, присущие коммерческим СУБД. Что касается регулярных выражений и полнотекстового поиска, то нужда в них возникает редко, но уж коли назреет, эти возможности оказываются поистине бесценными. Большинство примеров в данной главе не связано с ранее изученным материалом, а для демонстрации таблиц типа InnoDB понадобится специальная версия MySQL.

Использование таблиц типа InnoDB

MySQL поддерживает таблицы нескольких разновидностей, в том числе бывшего когда-то стандартным типа ISAM и ставшего таковым ныне типа MyISAM. Кроме того, поддерживаются типы InnoDB, BDB (Berkeley Database), HEAP и временные таблицы. У каждой разновидности свои преимущества и специфические требования. Из всех перечисленных наиболее важен тип InnoDB (конечно, после MyISAM), и на нем стоит остановиться подробнее. Тип таблиц InnoDB был разработан компанией Innobase Oy (рис. 11.1). В MySQL эта технология применяется на нижнем уровне доступа.

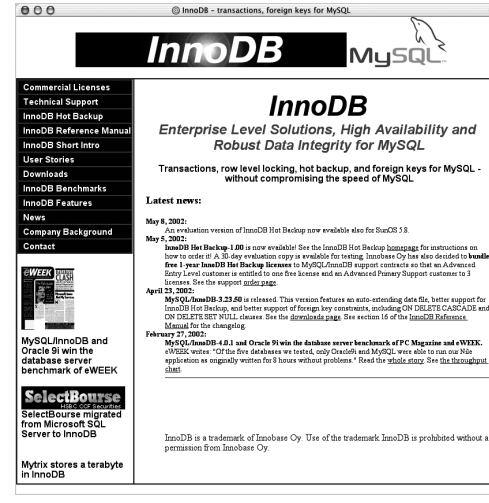

Рис. 11.1. Домашняя страница технологии InnoDB расположена по адресу www.innodb.com

Таблицы InnoDB устроены иначе, чем MyISAM (они хранятся в одном, а не в двух файлах), но, что гораздо важнее, они поддерживают транзакции. О том, что такое транзакции и почему им уделяется такое внимание, я расскажу в следующем разделе.

Чтобы создать таблицу типа InnoDB, вы выполняете обычное предложение `CREATE TABLE`, но в конце добавляете фразу `TYPE=InnoDB`. Так можно задать любой тип; если же эта фраза опущена, по умолчанию предполагается тип MyISAM.

Возможность работы с таблицами типа InnoDB обеспечена в последних версиях MySQL, а в версии 4 интеграция стала еще глубже. Я покажу, как создать такие таблицы, и опишу некоторые отличия в эксплуатации MySQL.

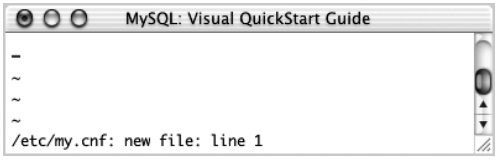

Рис. 11.2. Если на сервере еще нет файла my.cnf, придется его создать

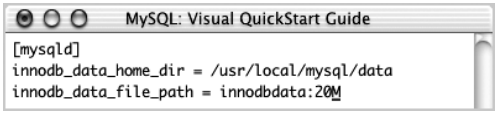

Рис. 11.3. Тип InnoDB можно использовать, только если в файле my.cnf (или my.ini) есть эти строки

Использование таблиц InnoDB

1. Создайте или отредактируйте файл my.cnf (рис. 11.2):

   `vi /etc/my.cnf` (UNIX или Mac OS X)

   Файл my.cnf содержит конфигурационные параметры, управляющие работой MySQL. Если он существует, то обычно находится в каталоге /etc в системах UNIX или в корневом каталоге диска C: системы Windows. Открыть my.cnf можно в любом текстовом редакторе. Если же такого файла нет, следует его создать.

   Если на вашем компьютере система загружается не с диска C:, создайте файл my.ini в каталоге WINDOWS или WINNT. Если при установке MySQL в системе Windows вы пользовались утилитой WinMySQLAdmin, то можете отредактировать файл my.ini прямо в ней. Не забудьте только по завершении работы нажать кнопку **Save Modifications** (Сохранить изменения)!

2. Добавьте в конфигурационный файл следующие строки (рис. 11.3) и сохраните файл:

   ```
   [mysqld]
   innodb_data_home_dir=/usr/local/
   ⇨ mysql/data
   innodb_data_file_path=innodbdata:20M
   ```

   Первая строка – `[mysqld]` – означает, что следующий за ней фрагмент относится к приложению mysqld. Вторая строка сообщает, где будут находиться таблицы типа InnoDB (измените ее в соответствии с конфигурацией системы на вашем компьютере). В третьей строке указаны имя и размер файла. Каждая таблица InnoDB хранится в отдельном файле под названием *имя_таблицы*.FRM, а вся прочая информация размещается в общем файле innodbdata.

Заметьте еще, что системе MySQL необходимо предоставить право создания файлов в указанном каталоге. Впрочем, в отношении стандартного каталога данных такое право уже действует.

3. Остановите процесс mysqld:

```
mysqladmin -u root -p shutdown
```

Поскольку вы изменяете параметры работы mysqld, выполняющийся процесс надо завершить. В системах Windows NT, 2000 и XP, где MySQL функционирует как сервис, можно остановить сервер с помощью иконки-светофора, находящейся в системном лотке.

4. Запустите mysqld или mysqld-max командой

```
./bin/safe_mysqld --user=mysql &
```

(UNIX и Mac OS X, см. рис. 11.4)

```
C:\mysql\bin\mysqld-max --user=mysql
```

(Windows)

В системах, отличных от Windows, перезапуск mysqld (с помощью `safe_mysqld`) после редактирования файла my.cnf обеспечит использование таблиц InnoDB. В Windows для этой цели нужно запускать именно сервер mysqldmax. В системах семейства Windows NT понадобится еще добавить флаг `--standalone`. (Разумеется, чтобы запустить mysqld-max, следует установить его из дистрибутива MySQL.)

5. Войдите в монитор mysql и создайте новую базу данных (рис. 11.5):

```
mysql -u root -p
CREATE DATABASE movies_db;
USE movies_db;
```

Эти действия ничем не отличаются от тех, что вы уже много раз производили

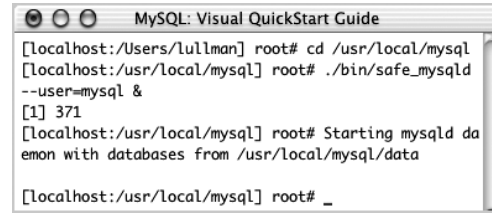

Рис. 11.4. Отредактировав файл my.cnf, перезапустите сервер, чтобы система MySQL восприняла изменения

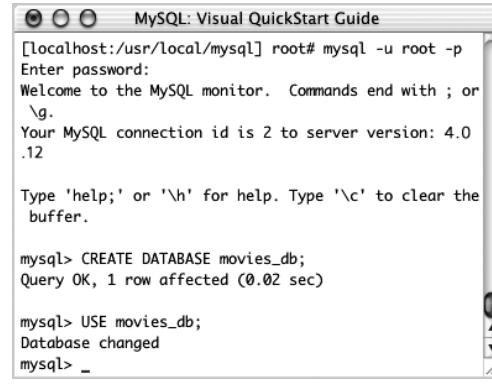

Рис. 11.5. Работа с таблицами типа InnoDB демонстрируется на примере новой базы данных movies_db

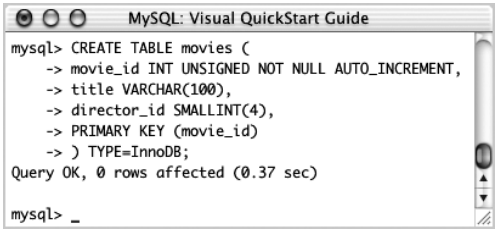

Рис. 11.6. Первая таблица типа InnoDB будет называться movies

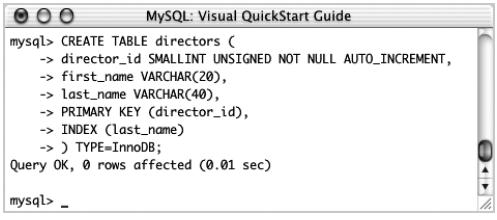

Рис. 11.7. Вторая таблица типа InnoDB будет содержать сведения о режиссерах

раньше. Вне зависимости от типа таблиц MySQL создает для каждой базы данных отдельный каталог, и синтаксис команд создания и выбора базы не меняется.

Вместо того чтобы использовать существующие базы данных, я предлагаю создать новую – для хранения информации о фильмах.

6. Создайте новую таблицу типа InnoDB (рис. 11.6):

```
CREATE TABLE movies (
movie_id INT UNSIGNED NOT NULL
⇨ AUTO_INCREMENT,
title VARCHAR(100),
director_id SMALLINT(4),
PRIMARY KEY (movie_id)
) TYPE=InnoDB;
```

В таблице movies будут поля title (название_фильма) и director_id (идентификатор_режиссера); последнее представляет собой внешний ключ в таблицу directors (режиссеры). Естественно, можно было бы включить еще много полей, но я ограничусь абсолютным минимумом. После закрывающей скобки указывается тип таблицы. Если в вашей системе таблицы InnoDB не поддерживаются, будет выдано сообщение об ошибке.

7. Создайте вторую таблицу типа InnoDB (рис. 11.7):

```
CREATE TABLE directors (
director_id SMALLINT(4) UNSIGNED
⇨ NOT NULL AUTO_INCREMENT,
first_name VARCHAR(20),
last_name VARCHAR(40),
PRIMARY KEY (director_id),
INDEX(last_name)
) TYPE=InnoDB;
```

Пока что в таблице directors будут храниться только имя и фамилия режиссера. Для повышения производительности я добавил индекс по колонке last_name (фамилия). Можно было бы также построить индекс по колонке title (название) таблицы movies (фильмы). Таблицы типа InnoDB можно индексировать, как любые другие.

8. Посмотрите, что оказалось в каталоге data (рис. 11.8):

```
ls -l /usr/local/mysql/data
```

В системах UNIX и Mac OS X команда `ls` показывает содержимое каталога. В Windows его можно посмотреть в программе Windows Explorer. Если все прошло без ошибок, то должны появиться файл innodbdata и некоторые специфичные для InnoDB файлы-протоколы. Кроме того, в каталоге movies_db окажутся файлы directors.frm и movies.frm.

С Изменить тип существующей таблицы можно, выполнив команду

```
ALTER TABLE имя_таблицы TYPE=InnoDB;
```

С Если вы сами собираете MySQL из исходных текстов, то для включения поддержки InnoDB нужно задать флаг `--with-innodb`.

П Для таблиц типы InnoDB распознаются внешние ключи (то есть можно сделать некоторую колонку внешним ключом) и проверяются соответствующие условия. Подробнее эта тема освещается в разделе документации по MySQL на странице www.mysql. com/doc/S/E/SEC446.html.

MySQL: Visual QuickStart Guide

```
[localhost:/Users/lullman] root# ls -l /usr/local/mysql/data
total 61728
drwxrwxrwx   17 mysql  wheel       534 May 21 20:04 accounting
drwx------   17 mysql  wheel       534 May 21 17:16 accounting2
drwxrwxrwx   11 mysql  wheel       330 May 18 16:53 beartrace
drwxrwxrwx   44 mysql  wheel      1452 May 22 09:39 bluechippick_com
drwxrwxrwx    8 mysql  wheel       264 Apr 15 09:07 community
drwx------    2 mysql  wheel       264 May 21 15:04 database
drwxrwxrwx   14 mysql  wheel       432 Apr 26 15:31 dmci_accounting
drwxrwxrwx  269 mysql  wheel      9102 Feb 22 12:56 dmcinsights_com
-rw-rw----    1 mysql  wheel     25088 May 23 14:06 ib_arch_log_0000000000
-rw-rw----    1 mysql  wheel   5242880 May 23 14:09 ib_logfile0
-rw-rw----    1 mysql  wheel   5242880 May 23 14:06 ib_logfile1
-rw-rw----    1 mysql  wheel  20971520 May 23 14:09 innodbdata
drwxrwxrwx   12 root   wheel       364 Feb 17 20:51 larrys_books
-rw-rw----    1 mysql  wheel       165 May 21 20:32 localhost.001
-rwxr-xr-x    1 mysql  wheel    108250 May 23 14:07 localhost.err
-rw-rw----    1 mysql  wheel         3 May 23 14:06 localhost.pid
drwx------    4 mysql  wheel        92 May 23 14:09 movies_db
drwxrwxrwx   20 mysql  wheel       636 Feb  6 14:18 mysql
drwxrwxrwx   41 mysql  wheel      1350 Mar 10 12:22 stupidwasteoftime_com
drwxrwxrwx    5 mysql  wheel       264 Feb 19 16:12 test
drwxrwxrwx   35 mysql  wheel      1146 Feb  6 19:28 wincs
[localhost:/Users/lullman] root# _
```

Рис. 11.8. Судя по размерам файлов, созданных подсистемой InnoDB (ib_logfile0, innodbdata и др.), можно сделать вывод, что таблицы этого типа занимают довольно много места

Транзакции в MySQL

Самые надежные СУБД, например Oracle, поддерживают транзакции, смысл которых состоит в двухэтапном выполнении SQL-запросов. Сначала запрос выполняется и анализируются его результаты. Если в процессе выполнения запросов не произошло ошибок, то результаты фиксируются, в противном случае откатываются[1]. Для таблиц типа InnoDB транзакции поддерживаются и в MySQL.

Транзакции повышают надежность хранения данных. Если из серии последовательных SQL запросов должны быть выполнены либо все, либо ни одного, то вы можете быть уверены, что не будет выполнена только часть из них. По сути дела, транзакции позволяют отменить результаты выполнения операций в базе данных. Вот почему таблицы с поддержкой транзакций более безопасны, их легче восстановить в случае порчи. С другой стороны, транзакции замедляют выполнение запросов – для поддержки транзакционности требуется более сложный API.

Чтобы начать транзакцию, используйте команду SQL `BEGIN`. После нее следуют обычные запросы. Если результаты транзакции вас устраивают, выполните команду `COMMIT` для их фиксации. В противном случае команда `ROLLBACK` позволит вернуть базу данных в исходное состояние. Давайте начнем добавлять режиссеров и фильмы в базу данных movies_db. Поле director_id в таблице movies зависит от поля director_id в таблице directors; транзакции гарантируют, что обе таблицы будут корректно модифицированы.

---

[1] Это весьма вольная трактовка механизма транзакций. Читателю стоит обратиться к более фундаментальным текстам за разъяснениями по данному вопросу. – *Прим. переводчика.*

Использование транзакций

1. Войдите в монитор mysql:

   ```
   mysql -u root -p
   ```

2. Выберите базу данных с поддержкой транзакций:

   ```
   USE movies_db;
   ```

   Не забывайте, что в текущей версии MySQL транзакции поддерживаются только для таблиц типа InnoDB и BDB.

3. Начните транзакцию (рис. 11.9):

   ```
   BEGIN;
   ```

   Начиная с этого места каждый запрос вплоть до команды `COMMIT` или `ROLLBACK` становится частью транзакции.

4. Вставьте записи о новом режиссере и новом фильме (рис. 11.10):

   ```
   INSERT INTO directors (first_name,
   ⇨ last_name) VALUES('Martin',
   ⇨ 'Scorsese');
   INSERT INTO movies(title,
   ⇨ director_id) VALUES('The Age of
   ⇨ Innocence', LAST_INSERT_ID());
   ```

   Поскольку поле director_id связывает таблицы movies и directors, хотелось бы убедиться, что обе таблицы обновлены корректно. Именно в таких случаях транзакции позволяют гарантировать целостность данных.

5. Если оба запроса были выполнены правильно, зафиксируйте изменения (рис. 11.11):

   ```
   SELECT * FROM movies, directors
   ⇨ WHERE movies.director_id=
   ⇨ directors.director_id;
   COMMIT;
   ```

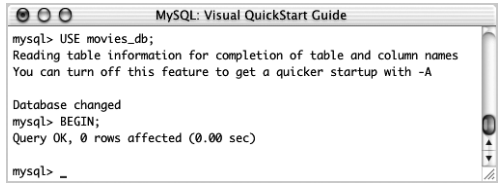

Рис. 11.9. Чтобы начать новую транзакцию, введите `BEGIN;` и нажмите клавишу **Enter** (**Return**)

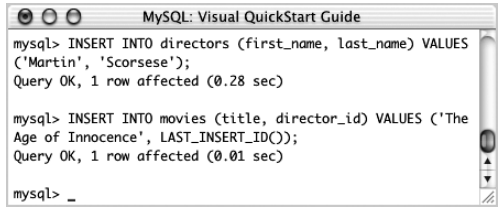

Рис. 11.10. В составе транзакции может быть сколько угодно запросов, в том числе типа `INSERT`, `UPDATE` и `DELETE`

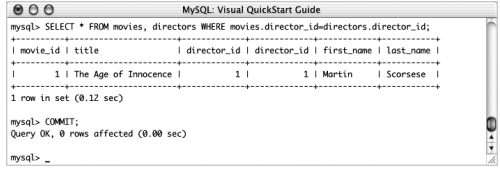

Рис. 11.11. После ввода команды `COMMIT;` результаты транзакции фиксируются

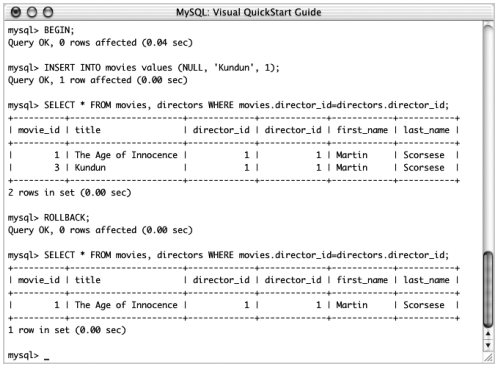

Рис. 11.12. Эта последовательность запросов показывает, насколько легко отменить результаты выполнения запросов

С помощью запроса SELECT вы проверяете, правильно ли информация занесена в таблицы. Внутри транзакции все внесенные изменения видны запросу SELECT даже до фиксации результатов, поэтому данный метод срабатывает.

6. Если вы не хотите фиксировать изменения, откатите их (рис. 11.12):

```
BEGIN;

INSERT INTO movies VALUES(NULL,
⇨ 'Kundun', 1);

SELECT * FROM movies, directors
⇨ WHERE movies.director_id=
⇨ directors.director_id;

ROLLBACK;
```

Чтобы показать, как происходит откат транзакции, я сначала добавил запись о фильме, а затем отменил эту операцию. Предложение SELECT позволяет увидеть, что запись была вставлена, но потом я вместо COMMIT употребил ROLLBACK. Если снова выполнить тот же самый запрос SELECT, вы уже не увидите только что вставленную запись.

П В случае, когда соединение с базой данных прерывается (в вашем сценарии или в мониторе mysql), все начатые транзакции откатываются автоматически.

П Многие команды SQL, в том числе ALTER, DROP и TRUNCATE автоматически начинают и завершают транзакцию.

С Если при работе с таблицами, поддерживающими транзакции, вы забудете про команду BEGIN, все запросы будут выполняться так, как если бы никаких транзакций не было (другими словами, после каждого предложения автоматически выполняется COMMIT).

Блокировка таблиц

Транзакции позволяют защитить целостность данных, но применимы только к отдельным типам таблиц. Для остальных типов, включая и MyISAM, ограничить доступ к таблицам на время выполнения серии запросов можно только путем блокировки. Для этого служит следующее предложение:

```
LOCK TABLES имя_таблицы1 LOCKTYPE,
⇨ имя_таблицы2 LOCKTYPE;
```

При выполнении этой операции над одной или несколькими таблицами можно с помощью параметра `LOCKTYPE` указать вид блокировки: `READ` (на чтение) или `WRITE` (на запись). Блокировка на чтение разрешает всем остальным пользователям только читать данные из таблицы – иначе говоря, нельзя изменять ни ее структуру, ни содержание. Блокировка на запись запрещает всем, кроме того, кто заблокировал таблицу, читать ее и записывать туда данные. Важно отметить, что блокировка ассоциирована с тем соединением, в котором была поставлена: если она используется в одном сеансе работы с клиентом mysql, то для всех остальных сеансов (от имени клиента mysql или любой другой программы) доступ к заблокированным таблицам будет ограничен.

Чтобы разблокировать таблицу, владелец блокировки должен выполнить команду

```
UNLOCK TABLES;
```

для того же соединения, где она была поставлена.

Обычно при работе с MySQL в блокировках таблиц не возникает необходимости, хотя в некоторых ситуациях (например,

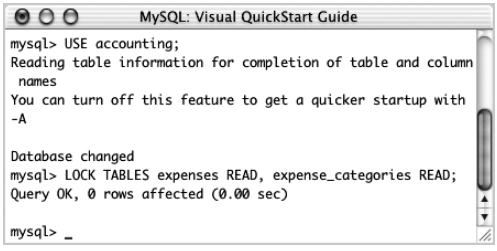

Рис. 11.13. Команда `LOCK TABLES` предохраняет таблицы от редактирования другими пользователями на время выполнения нескольких операций

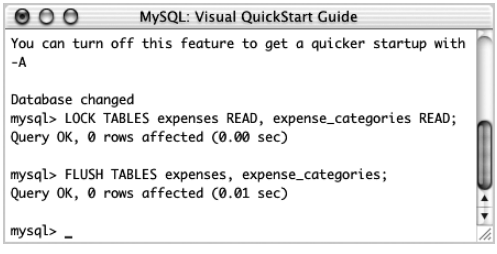

Рис. 11.14. Перед созданием резервной копии таблицы следует выполнить команду `FLUSH TABLES`

при резервном копировании базы данных с помощью команд SQL) такое решение имеет смысл. Другой типичный пример – немедленное выполнение обновления по результатам предложения `SELECT`, когда важно быть уверенным, что в промежутке между этими операциями информация не изменяется.

Блокировка таблицы

1. Войдите в монитор mysql:

   ```
   mysql -u root -p
   ```

2. Выберите базу данных accounting:

   ```
   USE accounting;
   ```

   Поскольку блокировать таблицы в базе, поддерживающей полноценные транзакции (такой как movies_db), нет никакого смысла, я предлагаю поработать с базой accounting.

3. Заблокируйте две таблицы (рис. 11.13):

   ```
   LOCK TABLES expenses READ,
   ⇨ expense_categories READ;
   ```

   На время резервного копирования я заблокирую таблицы, чтобы никакие изменения не вносились (иначе целостность копии будет поставлена под сомнение).

4. Запишите таблицы на диск (рис. 11.14):

   ```
   FLUSH TABLES expenses,
   ⇨ expense_categories;
   ```

   Перед тем как произвести резервное копирование командой `BACKUP TABLE`, вам рекомендуется выполнить команду `FLUSH`, чтобы кэшированные в памяти данные были записаны в файл на диске.

5. Выгрузите таблицы (рис. 11.15):

```
BACKUP TABLE expenses,
⇨ expense_categories TO '/tmp';
```

В результате выполнения этого запроса структура и содержание обеих таблиц будут записаны в каталог /tmp. Команда BACKUP TABLE применима только к таблицам типа MyISAM. Подробнее о резервном копировании баз данных рассказывается в главе 10.

6. Разблокируйте таблицы (рис. 11.16):

```
UNLOCK TABLES;
```

Эта команда разблокирует сразу все заблокированные таблицы. Не забывайте о ней, иначе остальные пользователи не смогут получить доступ к части базы!

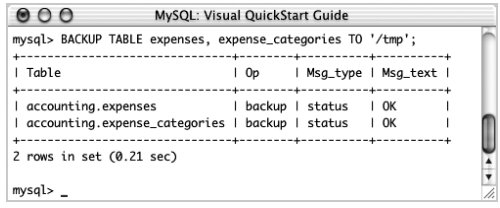

Рис. 11.15. Для резервного копирования таблицы я выполнил команду BACKUP TABLE прямо из монитора mysql

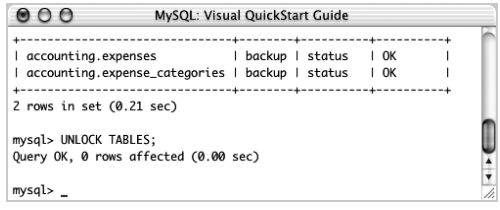

Рис. 11.16. Последний шаг – снятие блокировок со всех таблиц для восстановления нормального режима работы базы данных

П Вторая команда LOCK TABLES, введенная для того же соединения, что и предыдущая, автоматически снимает ранее поставленные блокировки.

С Можно в одном предложении поставить разнотипные блокировки на разные таблицы:

```
LOCK TABLES имя_таблицы1 WRITE,
⇨ имя_таблицы2 READ;
```

Полнотекстовый поиск

Начиная с версии 3.23.23 MySQL поддерживает полнотекстовый поиск. Благодаря этому вы можете находить отдельные слова, являющиеся частью значений колонок. Синтаксис предложения для выполнения полнотекстового поиска таков:

```
SELECT * FROM имя_таблицы WHERE
⇨ MATCH(имя_колонки) AGAINST('строка');
```

В результате будет возвращен набор записей, отсортированный по релевантности в порядке убывания. Проще говоря, первыми будут идти записи, в которых строка string лучше всего соответствует значению в указанной колонке таблицы. Чтобы узнать, какова релевантность записи, сформулируйте запрос так:

```
SELECT имя_колонки1, MATCH(имя_колонки2)
⇨ AGAINST('строка')FROM имя_таблицы;
```

Если вы хотите искать записи по нескольким словам, разделите их в запросе пробелами:

```
SELECT имя_колонки1, MATCH(имя_колонки2)
⇨ AGAINST('слово1 слово2')
⇨ FROM имя_таблицы;
```

В этом случае ранг записи, содержащей оба слова, будет выше, чем записей, содержащих только одно из двух слов (хотя последние тоже считаются удовлетворяющими запросу).

Можно пойти еще дальше, используя булевский режим (начиная с версии MySQL 4.01). При этом вы можете указывать относительные веса каждого из нескольких слов:

```
SELECT имя_колонки1, MATCH(имя_колонки2)
⇨ AGAINST('+слово1 слово2' IN BOOLEAN
⇨ MODE) FROM имя_таблицы;
```

В булевском режиме словам предшествуют специальные символы (табл. 11.1), указывающие их относительные веса при подсчете релевантности.

Если вы решили использовать полнотекстовый поиск, не забудьте построить полнотекстовый индекс по соответствующим колонкам. Вот как это делается:

```
ALTER TABLE имя_таблицы
⇨ ADD FULLTEXT имя_индекса
⇨ (имя_колонки1, имя_колонки2);
```

Полнотекстовый поиск важен в тех случаях, когда пользователи хотят искать отдельные слова в колонках, как, например, в поисковых машинах.

Таблица 11.1. Эти символы используются для указания значимости отдельных слов при полнотекстовом поиске

Символ	Значение	Пример	Интерпретация
+	Слово должно присутствовать	`+punk rock`	Слово `punk` должно присутствовать обязательно, а `rock` – необязательно
-	Слово не должно присутствовать	`+punk -rock`	Слово `punk` обязательно должно присутствовать, а слово `rock` – отсутствовать
' '	Слова должны присутствовать в указанной последовательности	`'punk rock'`	Ищется словосочетание `punk rock`
<	Менее важно	`<punk +rock`	Слово `rock` обязательно, а `punk` менее важно
>	Более важно	`>punk +rock`	Слово `rock` обязательно, а `punk` более важно
()	Группировка	`(>punk roll) +rock`	Слово `rock` обязательно, а `punk` и `roll` не обязательны, причем `punk` более важно, чем `roll`
~	Уменьшает релевантность (то есть степень соответствия запросу)	`+punk ~rock`	Слово `punk` обязательно, а присутствие `rock` уменьшает релевантность (хотя слово `rock` не должно отсутствовать)
*	Неточное соответствие (символ употребляется в конце слова)	`+punk +rock*`	Слова `punk` и `rock` обязательны, причем слова `rocks`, `rocker`, `rocking` и т.п. тоже подходят

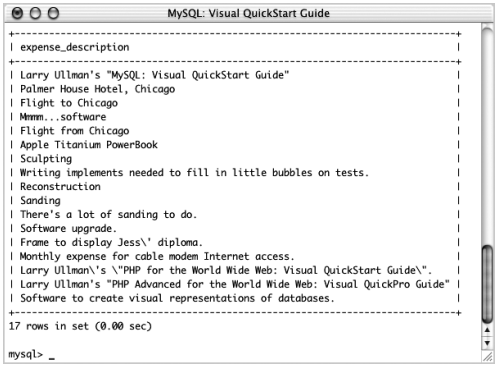

Рис. 11.17. Если в таблице expenses достаточно много записей, то к ней можно применять полнотекстовый поиск

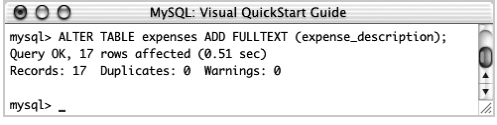

Рис. 11.18. Для создания полнотекстового индекса используется предложение ALTER TABLE

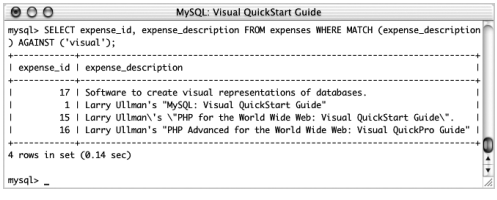

Рис. 11.19. В результате простейшего полнотекстового поиска будут возвращены все записи, содержащие указанное слово

Использование полнотекстового поиска

1. Войдите в монитор mysql:

   ```
   mysql -u root -p
   ```

2. Выберите базу данных accounting:

   ```
   USE accounting;
   ```

3. Убедитесь, что в таблице достаточно много записей (рис. 11.17).

   Добавьте в таблицу (неважно, каким способом) записи с содержательным описанием затрат. Польза от полнотекстового поиска тем ощутимее, чем больше записей в таблице.

4. Создайте полнотекстовый индекс (см. рис. 11.18):

   ```
   ALTER TABLE expenses
   ⇨ ADD FULLTEXT (expense_description);
   ```

   Прежде чем выполнять полнотекстовый поиск, необходимо создать полнотекстовый индекс по соответствующим колонкам.

5. Выполните полнотекстовый поиск по слову «visual» (визуальный) – рис. 11.19.

   ```
   SELECT expense_id, expense_description
   ⇨ FROM expenses WHERE MATCH
   ⇨ (expense_description)
   ⇨ AGAINST('visual');
   ```

   Этот запрос возвращает колонки expense_id и expense_description из записей, для которых релевантность слова «visual» относительно поля expense_description больше нуля. Иными словами, выводятся записи, в которых поле expense_description содержит слово «visual».

6. Выполните полнотекстовый поиск по словам «visual» и «guide» (рис. 11.20).

```
SELECT expense_id, MATCH
⇨ (expense_description) AGAINST
⇨ ('visual guide') AS rel,
⇨ expense_description FROM expenses
⇨ WHERE MATCH(expense_description)
⇨ AGAINST('visual guide');
```

Этот запрос отличается от предыдущего по двум причинам. Во-первых, выбираются не только значения полей, но также и вычисленная релевантность; во-вторых, в поисковом запросе указаны два слова. Записи, содержащие оба слова, имеют приоритет.

При работе с MySQL версии 4.01 и выше пользуйтесь полнотекстовыми запросами в булевском режиме.

При полнотекстовом поиске прописные и строчные буквы не различаются.

Если искомое слово обнаружено более чем в половине записей, оно вообще не учитывается, так как считается недостаточно специфичным.

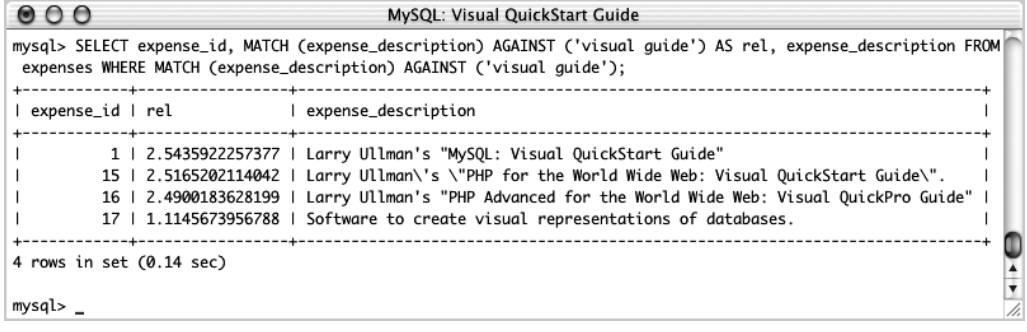

Рис. 11.20. При выполнении полнотекстового запроса с несколькими словами первыми возвращаются записи, содержащие все слова (впрочем, записи, содержащие только одно из них, также удовлетворяют запросу)

Таблица 11.2. При использовании предикатов REGEX и NOT REGEX в регулярные выражения могут входить метасимволы

Символ	Значение
.	Любой символ (один)
q?	Ноль или одно повторение *q*
*q**	Ноль или более повторений *q*
q+	По крайней мере одно *q*
q{*x*}	*x* повторений *q*
q{*x*,}	Не менее чем *x* повторений *q*
q{,*x*}	Не более чем *x* повторений *q*
q{*x*, *y*}	От *x* до *y* повторений *q*
^*q*	Начинается с *q*
q$	Заканчивается *q*
(*pqr*)	Группировка (соответствует *pqr*)
q|*z*	Альтернатива (или *q*, или *z*)
[]	Классы символов (например, [a-z], [0-9])
\	Экранирует метасимвол (\., * и т.д.)

Регулярные выражения

В главе 4 рассматривалось применение предикатов LIKE и NOT LIKE для поиска записей, содержащих в колонке некоторую строку. Для большей гибкости в искомой строке можно использовать метасимволы _ (любой символ) и % (любые символы в неограниченном количестве). Например:

```
SELECT * FROM users WHERE first_name
⇨ LIKE 'John%';
```

Благодаря регулярным выражениям, которые поддерживает MySQL, можно пойти в этом направлении существенно дальше. Регулярные выражения позволяют задать более сложные образцы для сопоставления и использовать их в предикатах REGEXP и NOT REGEXP (или RLIKE и NOT RLIKE):

```
SELECT * FROM users WHERE first_name
⇨ REGEXP '^(Jo)h?n+.*';
```

По этому запросу будут найдены имена John, Johnathon, Jon, Jonathon и т.д., тогда как запрос с предикатом LIKE выведет только слова John и Johnathon. Если вы раньше не встречались с регулярными выражениями, то показанный выше образец может вызвать недоумение, поэтому я подробнее остановлюсь на том, как записываются регулярные выражения.

В определении образца могут присутствовать литералы (например, 'a' соответствует единственной букве «a») и специальные символы в любом сочетании.

Если вы хотите найти строки, начинающиеся с ab, используйте регулярное выражение '^ab.*', то есть литерал ab плюс ноль или более любых символов (один символ представлен точкой). Для поиска строк, содержащих слово «color» или «colour», подойдет выражение 'col(or)|(our)': литерал col, за которым следует or или our.

Ясно, что самое трудное (в MySQL или любой другой программе) – научиться правильно составлять образцы. Подбор нужного выражения – это зачастую поиск компромисса между слишком размытым и слишком строгим вариантом.

Использование предикатов REGEX и NOT REGEX

1. Войдите в монитор mysql:

   ```
   mysql -u root -p
   ```

2. Выберите базу данных accounting:

   ```
   USE accounting;
   ```

3. Найдите клиентов с бесплатными номерами телефонов (рис. 11.21):

   ```
   SELECT client_name, client_phone
   ⇨ FROM clients WHERE client_phone
   ⇨ REGEXP '(800)|(888)|(877)';
   ```

   Бесплатные номера телефонов (в США и Канаде) имеют код доступа 800, 888 или 877. Приведенное выше регулярное выражение позволяет отобрать все такие телефоны по одному запросу.

4. Выберите все корректные адреса электронной почты контактных лиц (см. рис. 11.22):

   ```
   SELECT client_name, contact_first_
   ⇨ name, contact_email FROM clients
   ⇨ WHERE contact_email REGEXP
   ⇨ '[[:alnum:]]+@.*\.[[:alnum:]]{2,3}';
   ```

   Приведенное регулярное выражение для распознавания синтаксически корректных адресов электронной почты довольно простое и слишком «мягкое», но для экспресс-проверки его вполне достаточно. При желании вы можете составить более или менее строгое выражение, отвечающее вашим потребностям.

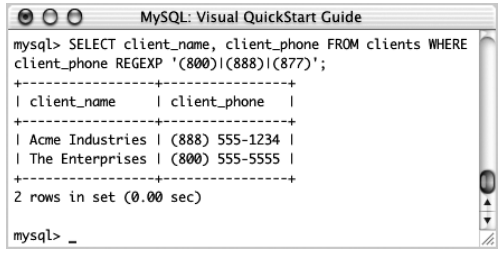

Рис. 11.21. Регулярные выражения повышают гибкость формулирования запросов

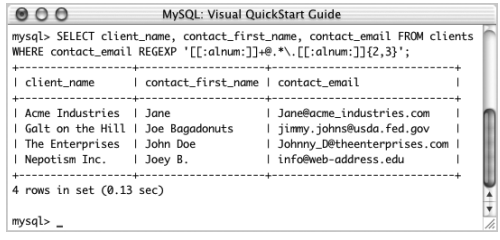

Рис. 11.22. Регулярные выражения также позволяют проконтролировать корректность данных

Приложение 1 Диагностика и устранение ошибок

При работе с MySQL, как, впрочем, и с любой другой программой, бывают моменты, когда хочется биться головой о стенку. Учитывая это, я включил в данный раздел описания наиболее типичных ошибок и способов их исправления. С одной стороны, после того как программа MySQL запущена, она обычно очень долго работает безо всякого вмешательства со стороны администратора. С другой стороны, иногда возникают проблемы при попытке запустить сервер. Прочитав это приложение, вы научитесь преодолевать элементарные трудности. Более полный перечень возможных проблем и путей их решения можно найти в руководстве по MySQL.

Установка MySQL

Не знаю, хорошо это или плохо, но большинство проблем возникает на этапе установки программы. Когда вы инсталлируете и конфигурируете MySQL, пользуйтесь следующими советами:

- при сборке MySQL из исходных текстов не забывайте указать параметр `--prefix=/путь/к/mysql`;
- если MySQL устанавливается впервые, то есть никаких баз данных еще нет, запустите сценарий mysql_install_db (он находится в каталоге mysql/scripts), который создает таблицы users (пользователи) и privileges (полномочия) – рис. П1.1;
- при сборке новой версии MySQL из исходных текстов не забудьте сначала удалить результаты предыдущей сборки, выполнив команды `rm config.cache` и `make clean`;
- на платформе Windows по возможности устанавливайте MySQL в каталог C:\mysql, который выбирается по умолчанию.

С Подробные сведения о конфигурировании вы найдете на странице www.mysql.com/doc/C/o/Compilation_problems.html или выведете по команде `./configure --help` (рис. П1.2).

П Дополнительная информация о сценарии mysql_install_db приводится на странице www.mysql.com/doc/m/y/mysql_install_db.html.

С Удалить информацию о предыдущей сборке можно одной командой `make distclean`.

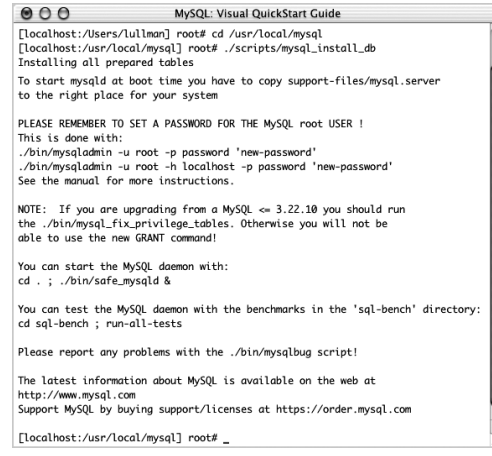

Рис. П1.1. Сценарий mysql_install_db подготавливает базу данных mysql, необходимую для предоставления полномочий пользователям

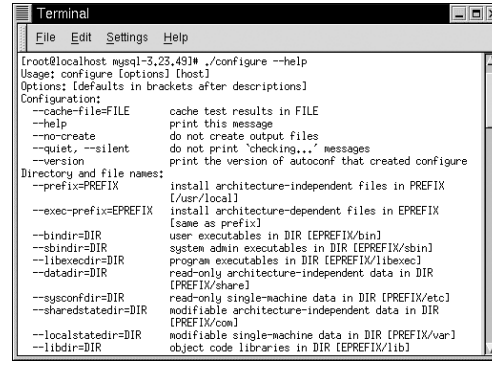

Рис. П1.2. При сборке из исходных кодов на стадии конфигурирования можно задать десятки разных параметров

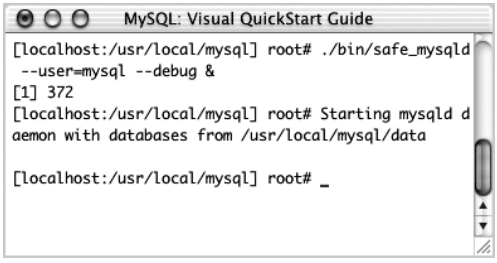

Рис. П1.3. Использование флага `--debug` позволяет выяснить, почему MySQL не запускается

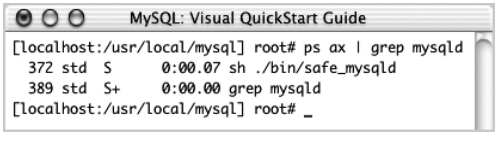

Рис. П1.4. Команда `ps as | grep mysqld` показывает, что safe_mysqld и mysqld работают

Запуск MySQL

Ниже представлены решения типичных проблем, которые могут возникнуть при запуске MySQL:

- если MySQL не может найти каталог с данными, задайте в команде запуска флаг `--basedir=`*`/путь/к/`*`mysql/data`;
- если вы работаете на платформе Windows, старайтесь устанавливать самые последние обновления операционной системы. Так, для работы в системе Windows NT для MySQL требуется Service Pack 3 (третий пакет исправлений и обновлений) или младше;
- сообщение о невозможности найти файл host.frm или mysql.host означает, что MySQL не может прочитать данные из таблиц полномочий в базе данных mysql. Убедитесь, что вы не забыли выполнить сценарий mysql_install_db и что MySQL имеет доступ к файлам, составляющим базу данных mysql.

Для отладки можно запустить MySQL с флагом `--debug` (рис. П1.3). В этом случае будет создан файл mysql.trace, где, в частности, протоколируются ошибки (просмотреть этот файл можно в любом редакторе).

П В Windows возникают сложности с запуском MySQL как сервиса, если в пути к исполняемой программе есть пробелы (например, C:/Program Files/mysql) или какая-то часть пути слишком длинна.

С Чтобы проверить, работает ли MySQL, наберите в UNIX и Mac OS X команду `ps ax | grep mysqld` (рис. П1.4).

П Более подробную информацию, касающуюся запуска MySQL, вы найдете на странице www.mysql.com/doc/P/o/Post-installation.html.

Доступ к MySQL

Итак, СУБД MySQL успешно установлена и запущена. Однако попытка получить доступ к базе данных может окончиться неудачей. Наиболее важные параметры, задаваемые при установлении соединения, – имя хоста, имя пользователя и пароль. Для открытия доступа эти значения должны соответствовать какой-либо записи о пользователе в базе данных mysql.

Далее MySQL проверяет, имеет ли пользователь право выполнять те или иные запросы к указанной базе данных (см. главу 2). Но, как правило, проблема заключается в невозможности установить соединение с сервером, а не в недостатке прав для выполнения запросов.

Вот как решаются вопросы доступа:

- перезагружайте MySQL после изменения полномочий, иначе новые значения не вступят в силу. Воспользуйтесь для этой цели утилитой mysqladmin (рис. П1.5) или выполните в мониторе mysql команду `FLUSH PRIVILEGES`;
- проверьте правильность пароля. Сообщение

  ```
  Access denied for user 'root@localhost'
  ⇨ (Using password: YES)
  ```

  обычно означает, что указан неверный пароль – это не единственная, но наиболее вероятная причина;
- при назначении полномочий или обновлении пароля пользуйтесь функцией `PASSWORD()`;

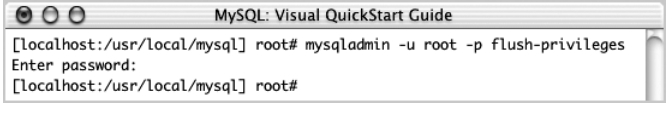

Рис. П1.5. После изменения состава пользователей или полномочий запустите утилиту mysqladmin, чтобы новые значения вступили в силу

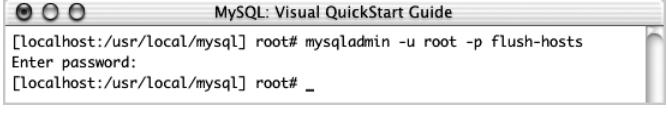

Рис. П1.6. Если проблема связана с именем хоста, для начала сбросьте кэш хостов MySQL при помощи утилиты mysqladmin

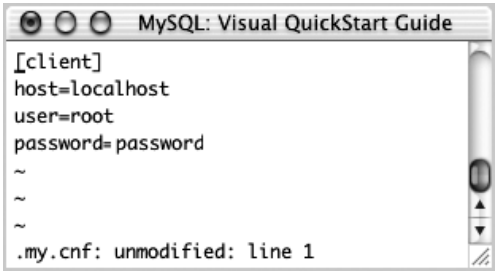

Рис. П1.7. Значения параметров в файле .my.cnf могут повлиять на поведение программ, обращающихся к MySQL, без вашего ведома

- если вы подозреваете, что доступ запрещен с конкретного хоста, попробуйте задать другое имя хоста (например, localhost, если вы работаете на том же компьютере, что и сервер mysqld). Попытайтесь также выполнить команду `mysqladmin flush-hosts` (рис. П1.6), запустить mysqld с флагом `--skip-name-resolve` или изменить ограничения на хосты, с которых может работать данный пользователь;
- если для настройки параметров клиентских программ вы применяете файл .my.cnf, не забудьте просмотреть хранящиеся в нем значения (рис. П1.7). Также проверьте содержимое глобальных файлов my.cnf и my.ini;
- сообщение `Can't connect to ...` (номер ошибки 2002) означает, что СУБД MySQL либо вообще не запущена, либо запущена не на том сокете или порте, с которыми пытается соединиться клиент. Сначала убедитесь, что MySQL работает, а затем попробуйте выбрать другой порт или сокет.

С Чтобы проверить, работает ли MySQL в системе Windows NT, воспользуйтесь приложением Task Manager.

П Подробнее проблемы с доступом описываются на странице www.mysql.com/doc/A/c/Access-denied.html.

Проблемы с mysql.sock

Для справки: mysql.sock – это имя сокета, с которым соединяются клиенты на платформах UNIX и Mac OS X[1] (в Windows используется протокол TCP/IP). Если клиентское приложение, например монитор mysql, не сможет найти этот сокет (обычно он находится в каталоге /tmp), то и соединение с сервером не установится.

Проблему, как и большинство ей подобных, часто решают путем перезагрузки компьютера. Но такой подход, во-первых, не всегда срабатывает, а, во-вторых, не является наилучшим. Есть два других решения: защитить каталог, где находится сокет, так чтобы его нельзя было удалить, или создать символическую ссылку на сокет в другом месте.

Как защитить каталог /tmp

1. Зарегистрируйтесь на сервере как пользователь root в приложении Terminal.
2. Добавьте «sticky bit» к правам доступа[2] на каталог /tmp (рис. П1.8):

   ```
   chmod +t /tmp
   ```

   Добавление этого флага к каталогу означает, что хранящиеся в нем файлы могут быть удалены только владельцем файла или суперпользователем. Одна из причин, по которым клиенту MySQL

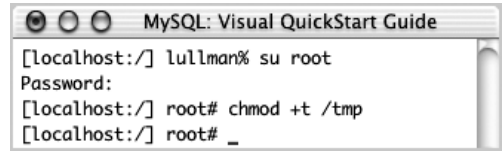

Рис. П1.8. В системах UNIX и Mac OS X для защиты от удаления файлов в каталоге /tmp можно использовать команду `chmod`

[1] Только когда клиент работает на той же машине, что и сервер. В противном случае используется сокет TCP/IP на порте 3306 (по умолчанию). – *Прим. переводчика.*

[2] Установка данного флага в правах доступа к каталогу означает, что пользователи могут удалять файлы, только если имеют право на запись в них. – *Прим. ред.*

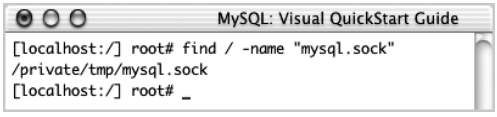

Рис. П1.9. Чтобы клиент мог соединиться с сервером MySQL, он должен найти сокет mysql.sock

не удается найти сокет, заключается в том, что его случайно удалили другой пользователь или другая программа. Данная операция исключает такую вероятность.

3. Перезапустите MySQL.

Как создать символическую ссылку

1. Зарегистрируйтесь на сервере как пользователь root в приложении Terminal.
2. Выясните, где находится сокет mysql.sock (рис. П1.9):

   ```
   find / -name "mysql.sock"
   ```

   По этой команде осуществляется просмотр всей файловой системы сервера в поисках файла mysql.sock. Если он находится в нестандартном месте, то клиент MySQL не сможет его найти, поэтому придется создать символическую ссылку.
3. Создайте символическую ссылку с реального файла сокета в то место, где он должен находиться.

   ```
   ln -s /путь/к/реальному/mysql.sock
   ⇨ /tmp/mysql.sock
   ```

   Наличие такой ссылки убедит клиента MySQL, что сокет mysql.sock находится в каталоге /tmp.

С Чтобы проверить, подходит ли сокет, запустите утилиту mysqladmin с флагом `--socket=/путь/к/сокету`.

П В некоторых конфигурациях сокет расположен в каталоге /var/lib/mysql, а не /tmp.

С Задать положение сокета можно либо в файле my.cnf, либо с помощью флага `--socket` при запуске mysqld.

Восстановление пароля пользователя root

Я знаю – такое бывает, сам испытал: при попытке соединиться с MySQL от имени пользователя root вы получаете отказ в доступе! Вы не можете ни вспомнить старый пароль, ни заставить работать новый. В общем, доступ к MySQL закрыт, и надежды тают с каждым часом. К счастью, выход из такой ситуации есть: нужно восстановить пароль пользователя root.

Для восстановления пароля вы запускаете MySQL без проверки полномочий. Тогда любой человек сможет получить доступ к базе, внести изменения и перезапустить MySQL.

Как восстановить пароль пользователя root

1. Остановите сервер MySQL, выполнив следующую команду от имени суперпользователя UNIX или пользователя mysql:

   ```
   kill `cat /путь/к/mysql/data/
   ⇨ имя_хоста.pid`
   ```

   Не зная пароля пользователя root, трудно остановить MySQL корректным способом. Если стандартные операции не дают результата, примените «жесткий» метод (для UNIX и Mac OS X)[1].

2. Запустите mysqld с флагом `--skip-grant-tables` (рис. П1.10):

   ```
   ./bin/safe_mysqld --user=mysql
   ⇨ --skip-grant-tables &
   ```

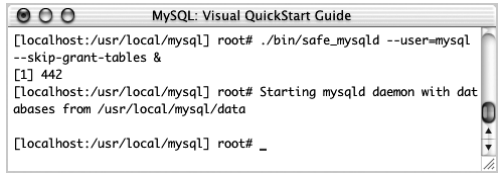

Рис. П1.10. При запуске mysqld с флагом `--skip-grant-tables` система проверки полномочий отключается

[1] Пользователи Windows в тех же целях могут воспользоваться Менеджером задач (Task Manager). – *Прим. ред.*

Рис. П1.11. Если СУБД MySQL была запущена с отключенной проверкой полномочий, то для ее останова не надо задавать имя и пароль пользователя

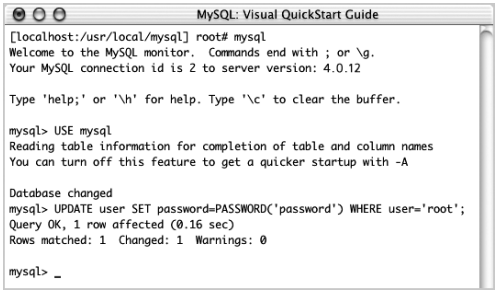

Рис. П1.12. Чтобы вручную изменить пароль, откройте базу mysql и выполните предложение `UPDATE` с функцией `PASSWORD()`

Флаг `--skip-grant-tables` полностью отключает систему проверки полномочий в MySQL. Хотя это очевидная угроза безопасности, другого способа восстановить пароль пользователя root не имеется.

3. Измените пароль пользователя root с помощью утилиты mysqladmin:

```
mysqladmin -u root password
⇨ 'новый_пароль'
```

Эта команда аналогична той, которая используется при назначении пароля пользователя root (см. главу 2). Но старый пароль не запрашивается, так как проверка полномочий отключена.

4. Снова остановите mysqld (рис. П1.11):

```
mysqladmin shutdown
```

Так как сейчас MySQL не проверяет полномочия, то нужно остановить и снова запустить сервер – команды `FLUSH PRIVILEGES` недостаточно.

5. Запустите mysqld обычным образом:

```
./bin/safe_mysqld --user=mysql &
```

6. Войдите в монитор mysql, задав имя и пароль пользователя root:

```
mysql -u root -p
```

С Чтобы восстановить пароль пользователя root с помощью клиента mysql, произведите действия, описанные в пп. 1 и 2, затем войдите в монитор mysql, выберите одноименную базу данных и выполните команду `UPDATE user SET password=PASSWORD ('новый_пароль') WHERE user='root'` (рис. П1.12). После этого перезапустите mysqld стандартным способом.

Восстановление начального значения автоинкрементного поля

Возможность автоматически инкрементировать поля таблицы (прежде всего, первичные ключи) – это очень удобный способ обеспечить уникальность ключей индекса. Обычно MySQL прекрасно справляется с этой задачей, так что вам беспокоиться не о чем. Но если удалить записи из конца таблицы, то механизм автоинкремента «рассинхронизуется». Допустим, есть десять записей, и вы удаляете девятую и десятую. Тогда следующая запись получит идентификатор 11, хотя значение 9 свободно. Чтобы исправить положение, нужно изменить значение счетчика автоинкремента[1].

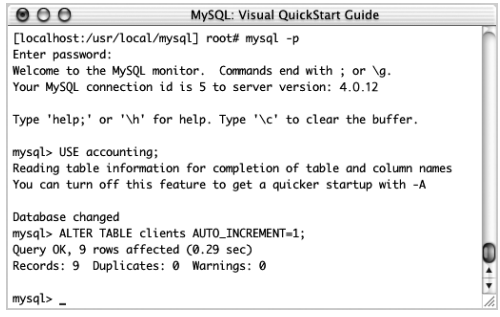

Рис. П1.13. Команду `ALTER` можно использовать для изменения значения счетчика автоинкремента

Восстановление счетчика автоинкремента

1. Войдите в монитор mysql:

   `mysql -u` *`имя_пользователя`* `-p`

2. Выберите базу данных:

   `USE` *`имя_базы_данных`*;

3. Измените соответствующую таблицу (рис. П1.13):

   `ALTER TABLE` *`имя_таблицы`*
   `AUTO_INCREMENT=9;`

   Эта команда устанавливает следующее значение счетчика для автоинкрементного поля по вашему указанию или просто берет очередное значение по порядку. В данном примере следующим значением должно быть 9.

[1] Обычно нет причин исправлять это положение. «Дырки» в нумерации никакого вреда не приносят. – *Прим. переводчика.*

Будьте осторожны

Я завел речь о том, как переустановить счетчик автоинкремента, поскольку начинающие пользователи часто задают такой вопрос. На мой взгляд, бывают ситуации, когда это имеет смысл: например, вы удалили из таблицы все записи и хотите снова начать нумерацию с 1.

Но, с другой стороны, чрезмерное усердие может нарушить целостность данных, если, например, автоинкрементной колонке соответствует внешний ключ в другой таблице. Полезно знать, как изменить счетчик автоинкремента, но следует помнить и о нежелательных побочных эффектах.

П Тот же результат достигается с помощью утилиты myisamchk:

```
myisamchk --set-auto-increment
```

С Как и большинство запросов SQL, предложение `ALTER ... AUTO_INCREMENT` можно выполнить из программы.

С В предыдущем примере вместо 9 вы могли бы использовать любое целое положительное число по своему усмотрению.

Если запрос возвращает странный результат

С этой проблемой сталкиваются многие пользователи MySQL, особенно при доступе к базе из программ на языках PHP, Perl или Java. Непонятно почему запрос вдруг возвращает неожиданные результаты. Начинающие разработчики могут решить, что это ошибка программы, но чаще всего дело в самом запросе.

Для отладки сложных запросов выполните следующие действия:

1. Если SQL-запрос строит программа, распечатайте его, чтобы точно знать, что же передается серверу для исполнения.
2. Если можно, выполните этот запрос с помощью другого инструмента, например монитора mysql.
3. Воспользуйтесь предложением `EXPLAIN`, чтобы посмотреть, как MySQL обрабатывает запрос (рис. П1.14). Подробнее команда `EXPLAIN` рассматривается в документации по MySQL.
4. Измените запрос так, чтобы выбирались лишь колонки, упомянутые в условии `WHERE`, а все остальное временно удалите.
5. Перепишите запрос так, чтобы он стал максимально простым. Постепенно усложняйте его, пока не поймете, что́ именно приводит к ошибке.

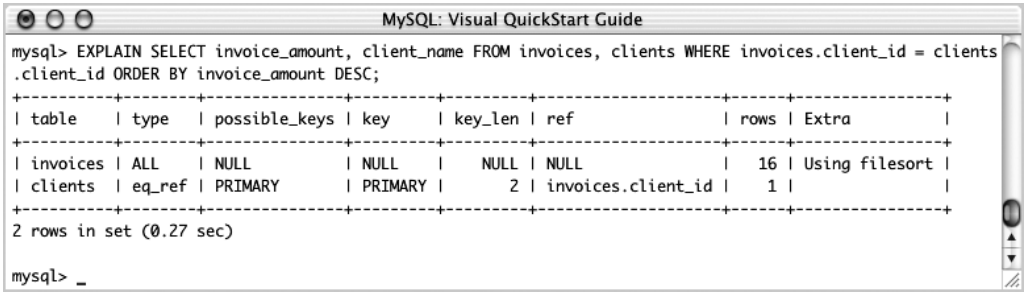

Рис. П1.14. Команда `EXPLAIN` показывает, что́ MySQL делает для выполнения запроса

Приложение 2 Справочник по SQL и MySQL

Это приложение задумано как краткий справочник по SQL и MySQL. Здесь собраны все таблицы, встречающиеся в книге, причем некоторые из них дополнены новым материалом. Здесь же разъясняется терминология и приводятся перечни функций. Материал сгруппирован по темам и в основном организован так же, как во всей книге. Отсутствующую информацию и самые свежие сведения о новых версиях ищите в руководстве по MySQL.

Основы языка SQL

Разработчики MySQL стремились обеспечить полную поддержку стандартов ANSI SQL92 и SQL99, но все же допустили некоторые отступления в интересах производительности. В этом разделе речь пойдет о стандартном языке SQL, а в следующих мы остановимся на некоторых особенностях MySQL. В табл. П2.1 представлены наиболее употребительные команды SQL.

Ниже приведены некоторые примеры запросов:

- CREATE DATABASE *имя_базы_данных*;

  создает новую базу данных;

- CREATE TABLE *имя_таблицы*
  ⇨ (*имя_колонки1 определение_колонки1*,
  ⇨ *имя_колонки2 определение_колонки2*);

  создает новую таблицу, структура которой задана определениями колонок;

- DELETE FROM *имя_таблицы*
  ⇨ WHERE *имя_колонки* = '*x*';

  удаляет из таблицы все строки, где в указанной колонке находится значение *x*;

- DROP TABLE *имя_таблицы*;

  удаляет таблицу вместе со всеми данными;

- DROP DATABASE *имя_базы_данных*;

  удаляет базу данных вместе со всеми таблицами;

- INSERT INTO *имя_таблицы*
  ⇨ VALUES('*x*', '*y*', '*z*');

  вставляет новую строку в указанную таблицу и заполняет колонки значениями *x*, *y*, *z*. Такая запись приемлема, лишь если число значений в точности равно числу колонок в таблице;

Таблица П2.1. Наиболее употребительные команды SQL

Команда	Использование
ALTER	Модификация структуры таблицы
CREATE	Создание таблицы или базы данных
DELETE	Удаление записей из таблицы
DROP	Удаление таблиц и баз данных
INSERT	Добавление записей в таблицу
SELECT	Выборка информации из базы данных
SHOW	Вывод сведений о структуре таблицы или базы данных
UPDATE	Обновление записей в таблице

- `INSERT INTO имя_таблицы (имя_колонки1, имя_колонки3) VALUES('x', 'y');`

  вставляет в указанную таблицу новую строку, записывая в первую колонку значение *x*, а в третью – значение *y*. Такая запись приемлема, если указанные колонки существуют;

- `INSERT INTO имя_таблицы VALUES('x', 'y', 'z'), ('a', 'b', 'c');`

  вставляет в таблицу две строки. Это расширение, введенное в MySQL; в других СУБД оно, скорее всего, работать не будет;

- `SELECT * FROM имя_таблицы;`

  выбирает из таблицы все строки и все колонки;

- `SELECT имя_колонки1, имя_колонки2 FROM имя_таблицы;`

  выбирает две указанные колонки из каждой записи;

- `UPDATE имя_таблицы SET имя_колонки = 'x';`

  записывает в указанную колонку каждой записи таблицы значение *x*.

Команда ALTER

Практически любую команду SQL можно использовать в разных контекстах, но команда ALTER, наверное, самая многоликая. Ее общее назначение – изменять структуру таблицы: переименовывать, добавлять, удалять или модифицировать колонки. В табл. П2.2 перечислены некоторые наиболее употребительные формы команды ALTER. Обратите внимание, что любая форма начинается с фразы ALTER TABLE.

Предложения

Во многих запросах и, прежде всего, в предложениях SELECT, UPDATE и DELETE употребляются фразы, призванные ограничить или модифицировать множество строк, на которые распространяется действие запроса. Чаще всего встречаются фразы WHERE, GROUP BY, ORDER BY и LIMIT. Вот несколько примеров:

- SELECT * FROM *имя_таблицы*
  ⇨ ORDER BY *имя_колонки* DESC;

  возвращает все записи таблицы, отсортированные по указанной колонке в порядке убывания;
- SELECT * FROM *имя_таблицы*
  ⇨ ORDER BY *имя_колонки1* ASC,
  ⇨ *имя_колонки2* DESC;

  возвращает все записи таблицы, отсортированные сначала по первой колонке в порядке возрастания, а затем – по второй в порядке убывания;
- SELECT * FROM *имя_таблицы* LIMIT 10;

  возвращает первые десять записей из таблицы;
- SELECT * FROM *имя_таблицы* LIMIT 100, 50;

  возвращает записи таблицы со 101 по 150.

Один запрос может содержать несколько фраз. Они могут просто ссылаться на некоторую колонку по имени, а могут в сочетании со скобками и операторами образовывать более сложные условия. В табл. П2.3 перечислены операторы и предикаты, которые часто употребляются в предложении WHERE.

Таблица П2.2. Команда ALTER TABLE применяется для изменения структуры существующей таблицы

Фраза	Пример	Назначение
ADD COLUMN	ADD COLUMN *имя_колонки* ⇨ VARCHAR(40)	Добавляет новую колонку после всех остальных
CHANGE COLUMN	CHANGE COLUMN *имя_колонки* ⇨ *имя_новой_колонки* VARCHAR(60)	Позволяет изменить тип данных, атрибуты и имя колонки
DROP COLUMN	DROP COLUMN *имя_колонки*	Удаляет колонку с ее содержимым
ADD INDEX	ADD INDEX *имя_индекса* ⇨ (*имя_колонки*)	Строит новый индекс по указанной колонке
DROP INDEX	DROP INDEX *имя_индекса*	Удаляет существующий индекс
RENAME AS	RENAME AS *новое_имя_таблицы*	Изменяет имя таблицы

Таблица П2.3. Операторы и предикаты, которые в сочетании со скобками могут служить для создания сложных выражений в SQL-запросах

Оператор	Назначение
+	Сложение
-	Вычитание
*	Умножение
/	Деление
%	Остаток от деления
=	Равно
<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно
!=	Не равно
%	Означает любые несколько символов (используется в сочетании с LIKE)
_ (знак подчеркивания)	Означает любой единичный символ (используется в сочетании с LIKE)
IS NOT NULL	Имеет значение (в том числе равное пустой строке или нулю)
IS NULL	Не имеет значения
BETWEEN	Внутри диапазона
NOT BETWEEN	Вне диапазона
IN	Принадлежит списку значений
NOT IN	Не принадлежит списку значений
OR (также ||)	Хотя бы один из операндов равен true
AND (также &&)	Оба операнда равны true
NOT (также !)	Условие ложно
OR NOT (также ^)	Один из операндов равен true
LIKE	Значение сопоставляется с образцом
NOT LIKE	Значение не сопоставляется с образцом
REGEXP	Значение сопоставляется с образцом в виде регулярного выражения
NOT REGEXP	Значение не сопоставляется с образцом в виде регулярного выражения
MATCH AGAINST	Значение содержит указанные слова

Административные команды SQL

Помимо команд для манипулирования данными в SQL есть команды, предназначенные исключительно для управления базой данных в целом (см. главу 10 и табл. П2.4).

Таблица П2.4. Команды SQL, предназначенные для решения административных задач

Команда	Пример	Назначение
ANALYZE	ANALYZE TABLE *имя_таблицы*	Исследует таблицы типа MyISAM и BDB
BACKUP	BACKUP TABLE *имя_таблицы* ⇨ TO '*/путь/к/резервному/каталогу*'	Сохраняет структуру и данные таблицы типа MyISAM в виде текстовых файлов
CHECK	CHECK TABLE *имя_таблицы*	Проверяет таблицы типа MyISAM и InnoDB
FLUSH	FLUSH PRIVILEGES	Актуализирует изменения привилегий, произведенные в базе mysql
OPTIMIZE	OPTIMIZE TABLE *имя_таблицы*	Повышает эффективность доступа к фрагментированной таблице типа MyISAM или BDB
REPAIR	REPAIR TABLE *имя_таблицы*	«Ремонтирует» поврежденные таблицы типа MyISAM
RESTORE	RESTORE TABLE *имя_таблицы* ⇨ FROM '*/путь/к/резервному/каталогу*'	Восстанавливает таблицу типа MyISAM из текстовых файлов, созданных командой BACKUP
SHOW	SHOW GRANTS FOR *имя_пользователя*	Выводит информацию о полномочиях указанного пользователя

Права доступа в MySQL

Встроенная в MySQL система проверки прав доступа определяет, кто и что может делать в конкретной базе данных. База данных mysql, которая создается в процессе установки сценарием mysql_install_db, содержит информацию о пользователях, паролях, хостах и базах данных. Там же хранятся сведения о том, какие действия (то есть какие команды SQL) разрешено выполнять каждому пользователю в каждой базе. В табл. П2.5 приведены варианты прав доступа. Содержимое колонки «Роль» характеризует уровень подготовки и ответственности пользователя, которому предоставляется то или иное право: *читатель* (разрешено только просматривать записи), *менеджер* (разрешено добавлять и обновлять записи), *администратор БД* (разрешено создавать базы данных), *администратор MySQL* (отвечает за сервер и установленную на нем систему MySQL). Это условная классификация, придуманная мной для того, чтобы вы легко определяли, как следует назначать права. Обычно каждому пользователю следует предоставлять лишь те полномочия, которые ему безусловно необходимы для работы.

Таблица П2.5. Список прав доступа, которые можно назначать пользователям MySQL

Право	Роль	Применимо к...
SELECT	Читатель	...таблицам
ALTER	Менеджер	...таблицам
DELETE	Менеджер	...таблицам
INSERT	Менеджер	...таблицам
UPDATE	Менеджер	...таблицам
CREATE	Администратор БД	...базам данных, таблицам и индексам
DROP	Администратор БД	...базам данных, таблицам
FILE	Администратор БД	...файлам на сервере
INDEX	Администратор БД	...таблицам
GRANT	Администратор MySQL	...базам данных, таблицам
PROCESS	Администратор MySQL	...серверу MySQL
RELOAD	Администратор MySQL	...серверу MySQL
SHUTDOWN	Администратор MySQL	...серверу MySQL

Типы данных в MySQL

Выбор правильного типа колонок – одно из важных условий построения качественной и быстрой базы данных. В табл. П2.6–П2.8 перечислены различные строковые, числовые и иные типы данных вместе с указанием места, необходимого каждому типу на диске. В описании колонки следует всегда указывать наиболее эффективный (то есть занимающий меньше места) тип, принимая во внимание наибольшее значение, которое может храниться в колонке.

Я не стал включать тип BLOB (Binary Large OBject – большой двоичный объект) в состав строковых, хотя технически он почти не отличается от типа TEXT; единственная разница в том, что BLOB подразумевает различение прописных и строчных букв, а TEXT – нет. Тип BLOB отнесен к другой категории, так как он обычно используется для хранения двоичных данных.

Кроме того, следует отметить, что типы MEDIUMINT, SET, ENUM, а также типы BLOB и TEXT разных размеров специфичны для MySQL и не описаны в стандарте SQL. То же самое относится к атрибутам AUTO_INCREMENT, UNSIGNED и ZEROFILL.

И наконец, определяя тип колонки, не забывайте, что любая колонка может иметь атрибут NULL или NOT NULL, что целочисленные колонки могут иметь атрибут UNSIGNED (БЕЗ ЗНАКА), а любые числовые колонки – атрибут ZEROFILL (С ЗАПОЛНЕНИЕМ НУЛЯМИ). Начиная с версии MySQL 4.0.2 атрибут UNSIGNED применим и к колонкам с плавающей точкой.

Таблица П2.6. Типы данных, применяемые для колонок, в которых хранятся строковые данные

Тип	Размер	Описание
CHAR[*длина*]	Число байтов равное *длина*	Поле фиксированной длины от 0 до 255 символов
VARCHAR[*длина*]	*длина* + 1 байт	Поле переменной длины от 0 до 255 символов
TINYTEXT	*длина* + 1 байт	Строка длиной не более 255 символов
TEXT	*длина* + 2 байта	Строка длиной не более 65 535 символов
MEDIUMTEXT	*длина* + 3 байта	Строка длиной не более 16 777 215 символов
LONGTEXT	*длина* + 4 байта	Строка длиной не более 4 294 967 295 символов

Таблица П2.7. При выборе типа числовой колонки нужно прежде всего понять, будут ли в ней храниться только целые числа или также и вещественные

Тип	Размер	Описание
TINYINT[*длина*]	1 байт	Целое число в диапазоне от –128 до 127 или от 0 до 255, если указан модификатор UNSIGNED
SMALLINT[*длина*]	2 байта	Целое число в диапазоне от –32 768 до 32 767 или от 0 до 65 535, если указан модификатор UNSIGNED
MEDIUMINT[*длина*]	3 байта	Целое число в диапазоне от –8 388 608 до 8 388 607 или от 0 до 16 777 215, если указан модификатор UNSIGNED
INT[*длина*]	4 байта	Целое число в диапазоне от –2 147 483 648 до 2 147 483 647 или от 0 до 4 294 967 295, если указан модификатор UNSIGNED
BIGINT[*длина*]	8 байт	Целое число в диапазоне от –9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 или от 0 до 18 446 744 073 709 551 615, если указан модификатор UNSIGNED
FLOAT[*длина*, *десятичные знаки*]	4 байта	Небольшое число с плавающей точкой
DOUBLE[*длина*, *десятичные знаки*]	8 байт	Большое число с плавающей точкой
DECIMAL[*длина*, *десятичные знаки*]	*длина* + 1 байт или *длина* + 2 байта	Число типа DOUBLE, хранящееся в виде с фиксированной десятичной точкой

Таблица П2.8. Если в колонке хранятся нечисловые и нетекстовые данные, то следует выбрать один из прочих типов, предоставляемых MySQL

Тип	Размер	Описание
DATE	3 байта	Дата в формате *ГГГГ-ММ-ДД*
DATETIME	8 байт	Дата и время в формате *ГГГГ-ММ-ДД ЧЧ:ММ:СС (год, месяц, день, часы, минуты, секунды)*
TIMESTAMP	4 байта	Временной штамп в формате *ГГГГММДДЧЧММСС*; допустимый диапазон – до 2037 года
TIME	3 байта	Время в формате *ЧЧ:ММ:СС*
YEAR	1 байт	Год в формате *ГГ* или *ГГГГ*
ENUM	1 байт или 2 байта	Перечисление, означающее, что в колонке может находиться одно из нескольких допустимых значений
SET	1, 2, 3, 4 или 8 байт	Аналогично ENUM, но в колонке может храниться более одного из нескольких допустимых значений
TINYBLOB	*длина* + 1 байт	Двоичные данные длиной до 255 символов
BLOB	*длина* + 2 байта	Двоичные данные длиной до 65 535 символов
MEDIUMBLOB	*длина* + 3 байта	Двоичные данные длиной до 16 777 215 символов
LONGBLOB	*длина* + 4 байта	Двоичные данные длиной до 4 294 967 295 символов

Функции MySQL

Форматирование результатов запроса на стороне сервера увеличивает ценность возвращаемых данных и позволяет сократить объем программирования на стороне клиента. Для этой цели в MySQL предназначены встроенные функции, о которых шла речь в главе 5. В табл. П2.9 приведены функции для работы со строковыми данными. В табл. П2.10 перечислено большинство функций для работы с числами, в табл. П2.11 – функции для работы с датами, а в табл. П2.12 – параметры форматирования для функций `DATE_FORMAT()` и `TIME_FORMAT()`. В табл. П2.13 собраны агрегатные функции, а в табл. П2.14 – все прочие, не вошедшие ни в одну из вышеупомянутых категорий. Почти все функции применимы как к значениям колонок, так и к литералам (например, `SELECT ROUND(3.142857, 2)`).

Таблица П2.9. Функции для работы с текстовыми данными

Функция	Пример использования	Назначение
`LEFT()`	`LEFT(`*имя_колонки*`, `*x*`)`	Возвращает *x* символов, начиная с первого
`LENGTH()`	`LENGTH(`*имя_колонки*`)`	Возвращает длину строки, хранящейся в указанной колонке
`LOCATE()`	`LOCATE(`*подстрока*`, `*строка*`)`	Возвращает позицию первого вхождения подстроки в строку
`LOWER()`	`LOWER(`*имя_колонки*`)`	Преобразует строку к нижнему регистру
`LTRIM()`	`LTRIM(`*имя_колонки*`)`	Исключает лишние пробелы в начале хранимой строки
`REPLACE()`	`REPLACE(`*имя_колонки*`, ⇨ `*найти*`, `*заменить*`)`	Возвращает значение колонки после замены всех вхождений строки *найти* на строку *заменить*
`RIGHT()`	`RIGHT(`*имя_колонки*`, `*x*`)`	Возвращает *x* символов, начиная с конца
`RTRIM()`	`RTRIM(`*имя_колонки*`)`	Исключает лишние пробелы в конце хранимой строки
`STRCMP()`	`STRCMP(`*имя_колонки1*`, ⇨ `*имя_колонки2*`)`	Возвращает 0, если строки совпадают, -1 – если строка в первой из указанных колонок меньше строки во второй колонке, и 1 – в противном случае
`SUBSTRING()`	`SUBSTRING(`*имя_колонки*`, ⇨ `*начало*`, `*число*`)`	Возвращает *число* символов из указанной колонки от позиции *начало* (позиции нумеруются с 0)
`TRIM()`	`TRIM(`*имя_колонки*`)`	Исключает лишние пробелы в начале и в конце хранимой строки
`UPPER()`	`UPPER(`*имя_колонки*`)`	Преобразует строку к верхнему регистру

Таблица П2.10. Некоторые из функций MySQL для работы с числами, не считая тригонометрических и еще более экзотических

Функция	Пример использования	Назначение
ABS()	ABS(*имя_колонки*)	Возвращает абсолютную величину указанной колонки
CEILING()	CEILING(*имя_колонки*)	Возвращает наименьшее целое число, большее либо равное значению в колонке
FLOOR()	FLOOR(*имя_колонки*)	Возвращает наибольшее целое число, меньшее либо равное значению в колонке
FORMAT()	FORMAT(*имя_колонки*, *y*)	Возвращает значение в колонке, отформатированное в виде числа с *y* десятичных знаков, причем группы из трех цифр разделены запятыми
LEAST()	LEAST(*x*, *y*, *z*...)	Возвращает наименьшее число в списке аргументов
GREATEST()	GREATEST(*x*, *y*, *z*...)	Возвращает наибольшее число в списке аргументов
MOD()	MOD(*x*, *y*)	Возвращает остаток от деления *x* на *y* (один или оба аргумента могут быть колонками)
POWER()	POWER(*x*, *y*)	Возвращает результат возведения *x* в степень *y*
RAND()	RAND()	Возвращает случайное число между 0 и 1,0
ROUND()	ROUND(*x*, *y*)	Возвращает число *x*, округленное до *y* десятичных знаков
SIGN()	SIGN(*имя_колонки*)	Возвращает индикатор знака аргумента: -1, если аргумент отрицателен, 0 – если равен 0, и 1 – если положителен
SQRT()	SQRT(*имя_колонки*)	Вычисляет квадратный корень из значения колонки

Таблица П2.11. В MySQL много функций для выполнения арифметических действий с датами и форматирования даты и времени

Функция	Пример использования	Назначение
HOUR()	HOUR(*имя_колонки*)	Возвращает только час
MINUTE()	MINUTE(*имя_колонки*)	Возвращает только минуту
SECOND()	SECONS(*имя_колонки*)	Возвращает только секунду
DAYNAME()	DAYNAME(*имя_колонки*)	Возвращает название дня
DAYOFMONTH()	DAYOFMONTH(*имя_колонки*)	Возвращает номер дня в месяце
MONTHNAME()	MONTHNAME(*имя_колонки*)	Возвращает название месяца
MONTH()	MONTH(*имя_колонки*)	Возвращает номер месяца
YEAR()	YEAR(*имя_колонки*)	Возвращает год
ADDDATE()	ADDDATE(*имя_колонки*, ⇨ INTERVAL *x тип*)	Возвращает дату, полученную прибавлением *x* единиц к значению колонки (см. врезку «Функции ADDDATE() и SUBDATE()» в главе 5)
SUBDATE()	SUBDATE(*имя_колонки*, ⇨ INTERVAL *x тип*)	Возвращает дату, полученную вычитанием *x* единиц из значения колонки (см. врезку «Функции ADDDATE() и SUBDATE()» в главе 5)

Таблица П2.11. В MySQL много функций для выполнения арифметических действий с датами и форматирования даты и времени (окончание)

Функция	Пример использования	Назначение
CURDATE()	CURDATE()	Возвращает текущую дату
CURTIME()	CURTIME()	Возвращает текущее время
NOW()	NOW()	Возвращает текущие дату и время
UNIX_ ⇨ TIMESTAMP()	UNIX_TIMESTAMP(*дата*)	Возвращает число секунд, прошедших с 1 января 1970 года или от указанной даты

Таблица П2.12. Специальные символы форматирования даты и времени, применяемые в функциях DATE_FORMAT() и TIME_FORMAT()

Символ	Значение	Пример
%e	Порядковый номер дня месяца	1-31
%d	Порядковый номер дня месяца с двумя цифрами	01-31
%D	Порядковый номер дня с суффиксом	1st-31st (1-е – 31-е)
%W	День недели	Sunday-Saturday (воскресенье–суббота)
%a	Сокращенное название дня недели	Sun-Sat
%c	Порядковый номер месяца	1-12
%m	Порядковый номер месяца с двумя цифрами	01-12
%M	Название месяца	January-December (январь–декабрь)
%b	Сокращенное название месяца	Jan-Dec
%Y	Год	2002
%y	Последние две цифры года	02
%l	Час	1-12
%h	Час с двумя цифрами	01-12
%k	Час в 24-часовом измерении	0-23
%H	Час в 24-часовом измерении с двумя цифрами	00-23
%i	Минуты	00-59
%S	Секунды	00-59
%r	Время	8:17:02 PM
%T	Время в 24-часовом измерении	20:17:02
%p	Сокращение AM (до полудня) или PM (пополудни)	AM или PM

Таблица П2.13. Агрегатные функции обычно используются в сочетании с фразой GROUP BY

Функция	Пример использования	Назначение
AVG()	AVG(*имя_колонки*)	Возвращает среднее значение в колонке
COUNT()	COUNT(*имя_колонки*)	Подсчитывает число строк
COUNT(DISTINCT)	COUNT(DISTINCT *имя_колонки*)	Подсчитывает число различных значений в колонке
MIN()	MIN(*имя_колонки*)	Возвращает наименьшее значение в колонке
MAX()	MAX(*имя_колонки*)	Возвращает наибольшее значение в колонке
SUM()	SUM(*имя_колонки*)	Возвращает сумму всех значений в колонке

Таблица П2.14. Функции для выполнения различных операций – от шифрования до конкатенации

Функция	Пример использования	Назначение
CONCAT()	CONCAT(*имя_колонки1*, ⇨ '-', *имя_колонки2*)	Сводит все аргументы в одну строку
CONCAT_WS()	CONCAT_WS(*символ*, ⇨ *имя_колонки1*, *имя_колонки2*)	Сводит все аргументы в одну строку, разделяя их указанным символом
DATABASE()	DATABASE()	Возвращает имя текущей рабочей базы данных
ENCODE()	ENCODE('*строка*', '*аргумент*')	Возвращает зашифрованную строку, которую можно восстановить в исходном виде
ENCRYPT()	ENCRYPT('*строка*', '*аргумент*')	Возвращает строку, зашифрованную с помощью аргумента (требует наличия функции crypt из стандартной библиотеки UNIX)
DECODE()	DECODE('*строка*', '*аргумент*')	Возвращает расшифрованную строку
LAST_INSERT_ID()	LAST_INSERT_ID()	Возвращает последнее значение автоинкрементной колонки
PASSWORD()	PASSWORD('*строка*')	Возвращает зашифрованную строку
USER()	USER()	Возвращает имя пользователя в текущем сеансе

Таблица П2.15. Для вставки некоторых специальных символов в базу данных их необходимо экранировать

Символ	Описание
\'	Одиночная кавычка
\"	Двойная кавычка
\\	Обратная косая черта
\n	Символ новой строки
\r	Перевод каретки
\t	Табуляция
\%	Символ процента
_	Символ подчеркивания

Другие справочные материалы

В табл. П2.15 показано, как представлены в коде специальные символы. Обратившись к табл. П2.16, вы научитесь задавать степень важности слов в булевском режиме полнотекстового поиска. В табл. П2.17 приведены некоторые метасимволы, употребляемые в регулярных выражениях.

Таблица П2.16. Начиная с версии MySQL 4 можно выполнять полнотекстовый поиск в булевском режиме, применяя специальные символы

Символ	Значение	Пример	Интерпретация
+	Слово должно присутствовать	+*punk rock*	Слово *punk* должно присутствовать обязательно, а *rock* – необязательно
-	Слово не должно присутствовать	+*punk -rock*	Слово *punk* обязательно должно присутствовать, а слово *rock* – отсутствовать
''	Слова должны встречаться в указанной последовательности	*'punk rock'*	Ищется фраза *punk rock*
<	Менее важно	<*punk* +*rock*	Слово *rock* обязательно, а *punk* менее важно
>	Более важно	>*punk* +*rock*	Слово *rock* обязательно, а *punk* более важно
()	Группировка	*(>punk roll)* +*rock*	Слово *rock* обязательно, а *punk* и *roll* необязательны, но *punk* более важно, чем *roll*
~	Уменьшает релевантность (то есть степень соответствия запросу)	+*punk ~rock*	Слово *punk* обязательно, а присутствие *rock* уменьшает релевантность (хотя слово *rock* не должно отсутствовать)
*	Неточное соответствие (символ употребляется в конце слова)	+*punk* +*rock**	Слова *punk* и *rock* обязательны, причем слова *rocks*, *rocker*, *rocking* и т.п. тоже подходят

Таблица П2.17. При использовании предикатов REGEX и NOT REGEX в регулярные выражения могут входить метасимволы

Символ	Значение
.	Любой символ (один)
q?	Ноль или одно повторение *q*
*q**	Ноль или более повторений *q*
q+	По крайней мере одно *q*
q{*x*}	*x* повторений *q*
q{*x*,}	Не менее чем *x* повторений *q*
q{,*x*}	Не более чем *x* повторений *q*
q{*x*,*y*}	От *x* до *y* повторений *q*
^*q*	Начинается с *q*
q$	Заканчивается *q*
(*pqr*)	Группировка (соответствует *pqr*)
q|*z*	Альтернатива (или *q*, или *z*)
[]	Классы символов (например, [a-z], [0-9])
\	Экранирует метасимвол (\., * и т.д.)

ПРИЛОЖЕНИЕ РЕСУРСЫ

3

Поскольку ни одна книга не может рассказать обо всем на свете и со временем утрачивает актуальность, я предпочитаю включать в свои работы перечень полезных ресурсов. Здесь вы найдете главным образом ссылки на Web-сайты, которые, на мой взгляд, заслуживают внимания, а также на другие книги и списки рассылки.

По всем вопросам, касающимся MySQL, обращайтесь прежде всего на официальный сайт www.mysql.com. Возможно, вторым источником информации станет для вас домашняя страница этой книги – www.DMCinsignts.com/mysql. Сайт создавался специально для поддержки книги, там вы найдете:

- дополнительные ссылки на Web-ресурсы (при последнем подсчете их было около 200);
- примеры сценариев, не вошедшие в книгу;
- дополнительные учебные руководства (по мере их написания);
- форум для обсуждения этой книги;
- перечень опечаток (если таковые со временем обнаружатся).

Все ресурсы, упомянутые в этой главе, как мне кажется, будут полезными среднему читателю. Упоминание того или иного ресурса не следует расценивать как рекламу; не нужно также считать, что это лучший источник информации по данной теме.

MySQL

Первейший и самый авторитетный ресурс по MySQL – это руководство, опубликованное на сайте компании-разработчика (www.mysql.com) в различных форматах. В основной онлайновой версии возможен контекстный поиск, тогда как другой вариант содержит комментарии читателей, к которым иногда не мешает обратиться. Я также храню у себя на диске копии руководств для тех версий MySQL, которыми пользуюсь (поскольку материалы, опубликованные на сайте, всегда относятся к текущей версии, небесполезно сохранять и старые варианты).

Если поиск в руководстве по MySQL ничего не дал, стоит обратиться к нескольким спискам рассылки, посвященным MySQL (официальной конференции в USENET не существует). Списки посвящены разным темам, в числе которых:

- объявления;
- общие вопросы;
- Java;
- Windows;
- ODBC;
- C++;
- Perl.

Все эти списки, за исключением объявлений, доступны в виде дайджестов (коротких сводок). Подписавшись на такую рассылку, вы будете ежедневно получать два больших письма вместо полусотни маленьких. Подобные списки ведутся на разных языках (см. www.mysql.com/doc/M/a/Mailing-list.html).

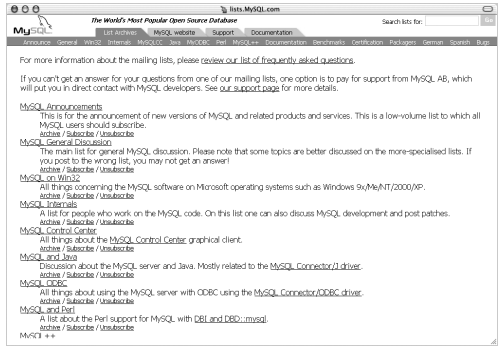

Рис. П3.1. Чтобы получить информацию по MySQL из списков рассылки, оформите подписку или изучите архивы

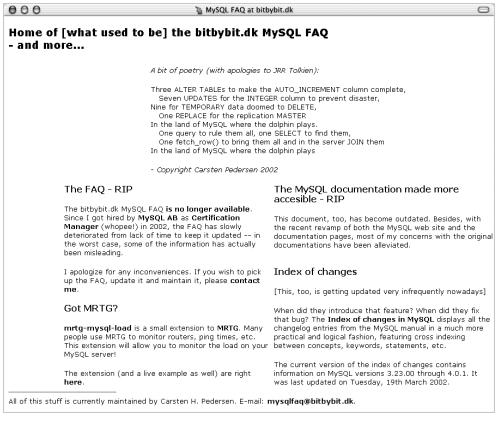

Рис. П3.2. На сайте Bit By Bit много информации, полезной для начинающего разработчика на MySQL

Еще один сайт, посвященный MySQL, – http://lists.mysql.com (рис. П3.1). Здесь можно проводить поиск по архивам списков рассылки. Прежде чем отправлять свой вопрос, имеет смысл поискать ответ на него в архивах (и, конечно же, в руководстве) – иначе вы рискуете получить отповедь от постоянных подписчиков.

На сайте датской компании Bit By Bit (www.bitbybit.dk/mysqlfaq) – рис. П3.2 – есть несколько хороших ресурсов по MySQL, включая список часто задаваемых вопросов (FAQ), который поддерживает Карстен Педерсен (Carsten Pedersen).

Приложения сторонних фирм для MySQL

На сайте MySQL приведен список различных приложений и инструментальных средств для MySQL – от простых сценариев до развитых интерфейсов (см. www.mysql.com/downloads/contrib.html).

В главе 3 я упоминал о специализированных приложениях для разработки схем баз данных. К их числу относятся, например, программы DeZign (www.datanamic.com/dezign/index.html) – рис. П3.3 – и Tabledesigner (www.tabledesigner.com). Хотя такого рода приложения могут быть написаны в расчете на конкретные СУБД, это не мешает применять их и для MySQL.

Простой и в то же время мощный инструмент администрирования – приложение phpMyAdmin, созданное талантливыми программистами из компании phpWizard.net (www.phpwizard.net). Для работы вам потребуется наличие Web-сервера с PHP, зато вы получите возможность взаимодействовать с MySQL из Web-браузера, а не из монитора mysql (рис. П3.4).

Наконец, программа MySQL Manager от компании EMS (http://ems-hitech.com/mymanager/) – недорогой и всеобъемлющий инструмент администрирования MySQL.

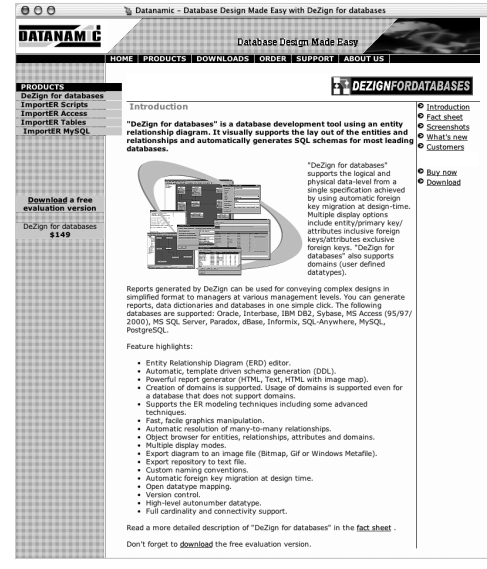

Рис. П3.3. Программа DeZign от компании Datanamic помогает проектировать схемы баз данных (отношения «сущность–связь»)

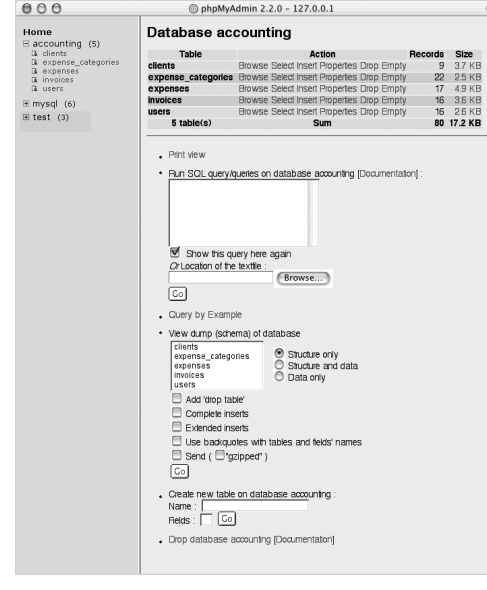

Рис. П3.4. Программа phpMyAdmin от фирмы phpWizard.net – прекрасное средство работы с MySQL

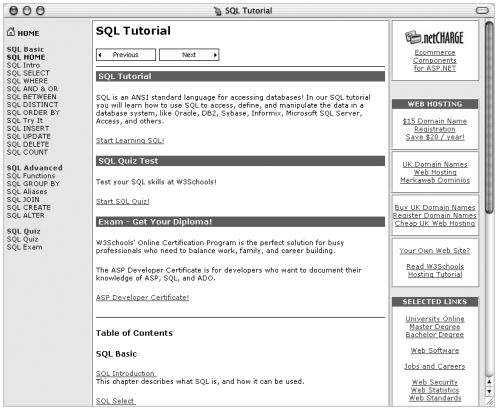

Рис. П3.5. На сайте W3Schools размещено много онлайновых руководств, в том числе и по MySQL

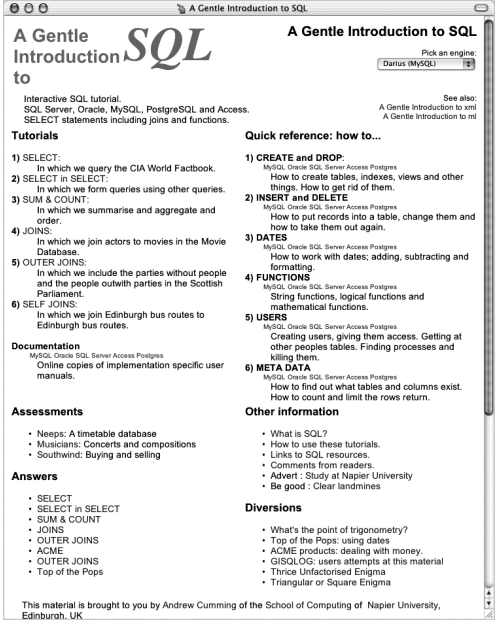

Рис. П3.6. На сайте Эндрю Камминга (Andrew Cumming) «A Gentle Introduction to SQL» язык обсуждается на уровне, доступном начинающему разработчику

Язык SQL

Поскольку язык SQL используется как в MySQL, так и в других СУБД, ему посвящено огромное количество ресурсов. Хотя руководства по стандартному SQL не научат вас, как реализовать максимум возможностей, предоставляемых MySQL, представление об основах технологии у вас все-таки сложится. Вот лишь несколько онлайновых руководств по языку SQL:

- курс SQL – www.sqlcourse.com;
- W3Schools.com – www.w3schools.com/sql/default.asp (рис. П3.5);
- «A Gentle Introduction to SQL», введение в SQL – www.sqlzoo.net (рис. П3.6);
- курс SQL, часть 2 – www.sqlcourse2.com.

Во многих книжных магазинах продаются книги по языку SQL. Я рекомендую две из них: Judith S. Bowman, «The Practical SQL Handbook» и Martin Gruber, «Mastering SQL»[1]. Впрочем, книги, целиком посвященные SQL, изобилуют информацией, необходимой разве что профессиональным разработчикам приложений для баз данных. В довершение всего, вы обнаружите, что MySQL не поддерживает многое из того, что описано в книгах по стандартному языку SQL.

[1] Эти книги вышли на русском языке: Грабер М. SQL. Описание SQL92, SQL99 и SQLJ. – М.: Лори, 2001; Боуман Дж. С., Эмерсон С. Л., Дарновски М. Практическое руководство по SQL. Изд. 4-е. – М.: Вильямс, 2002.

Общие вопросы теории баз данных

Множество сайтов и книг, посвященных общим вопросам баз данных, пригодится и при работе с MySQL. В частности, стоит упомянуть SearchDatabase (www.searchdatabase. com) и виртуальную библиотеку Web-разработчика (http://wdvl.internet.com/Authoring/DB/Relational).

Среди других сайтов, имеющих отношение к MySQL, отмечу следующие:

- ANSI – www.ansi.org. Здесь можно найти стандарт SQL;
- InnoDB – www.innodb.com. Сайт фирмы, разработавшей таблицы типа InnoDB;
- Sleepycat – www.sleepycat.com (рис. П3.7). Сайт фирмы, разработавшей таблицы типа Berkeley DB (BDB).

Есть книга, где рассматриваются все вопросы проектирования баз данных – начиная с того, нужна ли вам база данных вообще, и заканчивая определением типов колонок (Michael J. Hernandez, «Database Design for Mere Mortals»).

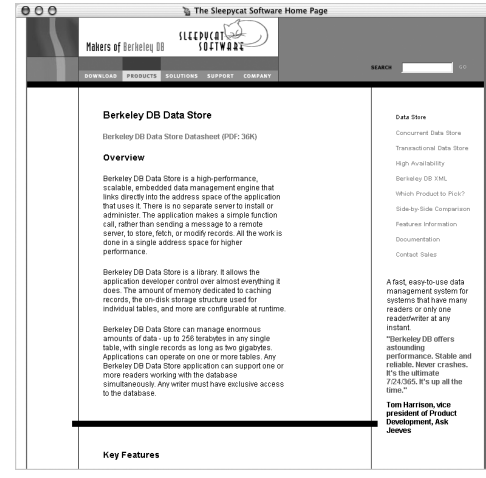

Рис. П3.7. Компания Sleepycat разработала и сопровождает таблицы типа BDB, которые можно использовать в MySQL

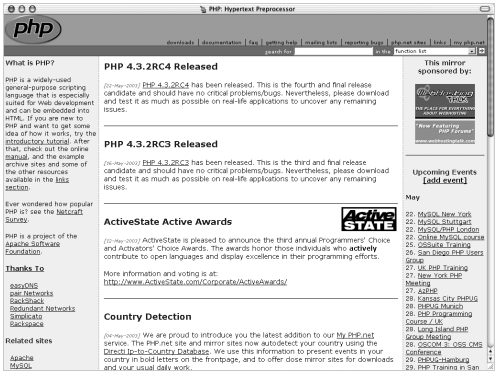

Рис. П3.8. PHP.net – официальная страница языка PHP

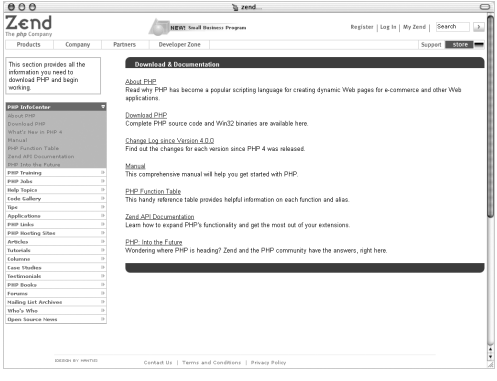

Рис. П3.9. На сайте Zend.com предоставлена разносторонняя поддержка технологии PHP

Язык PHP

В главе 6 было показано, как обращаться к базе данных MySQL из PHP-сценария. Технология PHP настолько популярна, что ей посвящены десятки сайтов. Вот некоторые наиболее известные и полезные:

- домашняя страница PHP (рис. П3.8) – www.php.net. Здесь вы найдете официальное руководство и многое другое;
- Zend – www.zend.com (рис. П3.9). Сайт содержит множество статей и примеров кода;
- PHPBuilder – www.phpbuilder.com. Это замечательный сайт, где есть учебные руководства, форумы и примеры программ.

Язык Perl

В главе 7 был продемонстрирован доступ к MySQL из Perl-сценариев. Ниже перечислено несколько сайтов, где можно найти дополнительную информацию:

- www.perl.com – официальная страница Perl;
- www.cpan.org – сетевой архив материалов по Perl (рис. П3.10). Здесь представлены, в частности, модули для работы с MySQL;
- ActiveState (www.activestate.com) – отсюда можно загрузить продукт ActivePerl, представляющий собой дистрибутив Perl для Windows;
- http://dbi.perl.org – документация по технологии доступа к базам данных DBI (рис. П3.11). Там же есть ссылки на различные ресурсы, посвященные Perl DBI.

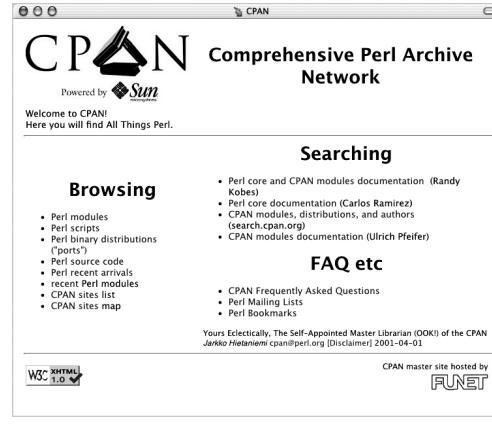

Рис. П3.10. Для доступа к базам данных MySQL из Perl понадобятся некоторые модули из архива CPAN

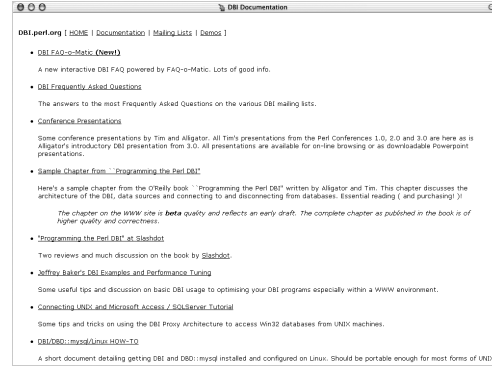

Рис. П3.11. В документации по DBI есть материалы, касающиеся как Perl, так и общих вопросов по базам данных

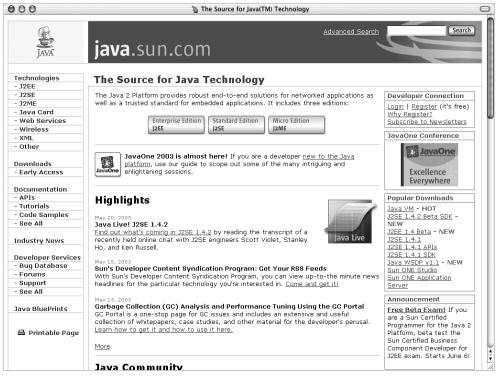

Рис. П3.12. Компания Sun, разработавшая и поддерживающая язык Java, публикует документацию по нему на своем сайте

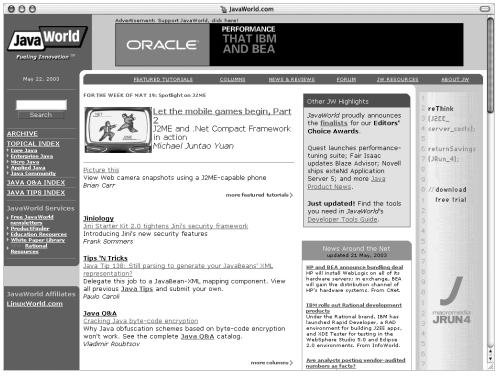

Рис. П3.13. Сайт JavaWorld – прекрасный ресурс для программистов на Java любого уровня

Язык Java

В главе 8 речь шла об использовании драйвера JDBC для доступа к MySQL из программ на языке Java. Вот несколько ссылок на ресурсы по этой теме:

- http://java.sun.com – домашняя страница языка Java (рис. П3.12);
- http://java.sun.com/products/jdbc – домашняя страница технологии JDBC;
- http://mmmysql.sourceforge.net – страница, посвященная драйверу JDBC для MySQL;
- http://javaboutique.internet.com – различные статьи и примеры программ на Java;
- http://www.javaworld.com (рис. П3.13) – разнообразные руководства и примеры кода;
- страница JDBC-драйвера Caucho, www.caucho.com/projects/jdbc-mysql – альтернативы драйверу mm.mysql.

Безопасность

Тема безопасности баз данных заслуживает отдельной книги. Когда речь идет об их защите, устаревшую информацию можно считать просто-напросто непригодной. Лучший способ не отстать от жизни – постоянно следить за статьями, появляющимися на нижеперечисленных сайтах:

- www.mysql.com/doc/G/e/General_security.html (рис. П3.14) – специальный раздел документации по MySQL, посвященный безопасности;
- CERT, www.cert.org. Организация, связанная с Университетом Карнеги Мелон, публикует отчеты о различных вопросах безопасности в Internet;
- сайт SecurityFocus – www.securityfocus.com. Представлены материалы по безопасности, касающиеся главным образом систем Windows и UNIX. Здесь же появляются извещения об ошибках и вирусах, о которых разработчики должны знать;
- сайт Insecure.org – www.insecure.org. Вам предлагается трактовка вопросов безопасности с точки зрения хакера. Акцент сделан главным образом на ОС UNIX. Кроме того, здесь вы найдете различные инструментальные средства;
- www.tripwire.org – программа с открытым исходным кодом Tripwire работает на вашем компьютере и следит за критически важными файлами. В настоящее время существует только версия для систем UNIX;
- OpenSSH – www.openssh.org: бесплатное приложение, обеспечивающее безопасную работу по протоколам telnet и FTP;
- PuTTY – www.chiark.greenend.org.uk/~sgtatham/putty: бесплатный популярный клиент SSH для Windows.

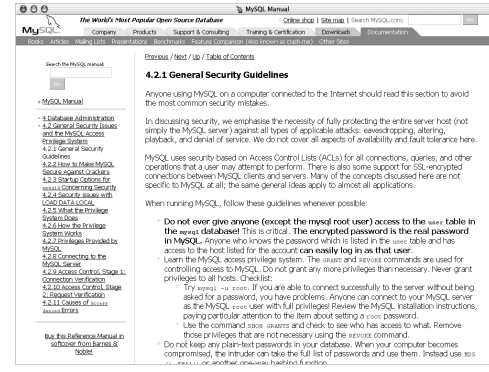

Рис. П3.14. Раздел документации по MySQL, касающийся безопасности, должен прочесть каждый пользователь

Остается добавить, что в MySQL версии 4.0 предусмотрена возможность работы по протоколу SSL (Secure Socket Layer – слой защищенных сокетов) для установления безопасного соединения с базой данных. В руководстве описано, как добиться того же результата с помощью SSH (Secure SHell – защищенная оболочка). Оба подхода заслуживают внимания в случаях, когда важна безопасная передача данных.

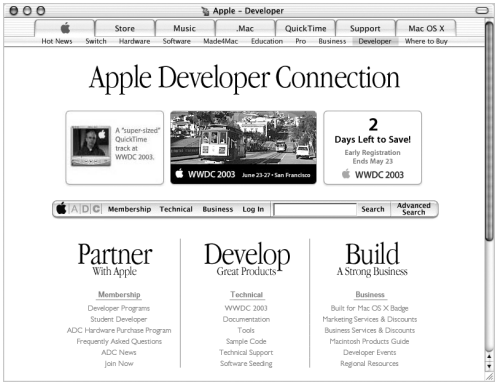

Рис. П3.15. Сайт Apple Developer Connection – необходимый ресурс для более опытных пользователей системы Mac OS X

Прочие ресурсы

Напоследок я предложу вашему вниманию несколько ресурсов, не попадающих ни в одну из вышеперечисленных категорий. Разработчики, которые используют систему Mac OS X (а я ее большой поклонник), должны знать о сайтах:

- http://developer.apple.com предоставляет разработчикам поддержку для систем компании Apple (рис. П3.15);
- Entropy – www.entropy.ch.

Сайт E-gineer (www.e-gineer.com) – это прекрасный источник информации об установке MySQL и других программ.

Уже неоднократно упоминавшийся сайт SourceForge.net (www.sourceforge.net) рекламируют как самое большое в мире хранилище программ с открытым исходным кодом – и тому есть веские основания. На этом сайте разрабатываются и размещаются тысячи самых разных технологий, в том числе JDBC-драйвер mm.mysql.

Developer Sched (www.devsched.com) – ресурс, посвященный общим вопросам разработки программ. Здесь публикуются статьи о различных языках программирования и технологиях.

Сайт WebMonkey (www.webmonkey.com) очень похож на Developer Sched, но шире по тематике.

Если вас заинтересовали движение в поддержку открытых исходных кодов и проекты типа MySQL, загляните на сайт www.opensource.org.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

M

H

O

П

P

C

T

У

Ф

X

A

B

C

D

E

F

G

I

J

K

L

M

O

P

R

S

T

U

W

X

Z

Ларри Ульман

MySQL

Главный редактор	*Мовчан Д. А.*
	dm@dmk-press.ru
Перевод с английского	*Слинкин А. А.*
Научный редактор	*Нилов М. В.*
Выпускающий редактор	*Готлиб О. В.*
Верстка	*Трубачев М. П.*
Графика	*Салимонов Р. В.*
Дизайн обложки	*Дудатий А. М.*

Подписано в печать 3.06.2003. Формат 70×100 $^{1}/_{16}$.
Гарнитура «Миниатюра». Печать офсетная.
Усл. печ. л. 28,6. Тираж 3000 экз. Зак. №

Издательство «ДМК Пресс»
Web-сайт издательства: www.dmk-press.ru
Internet-магазин: www.alians-kniga.ru