Библиотека профессионала

MySQL

Содержит примеры проектирования и **реализации** систем корпоративного уровня и подробные объяснения к ним

Включает главы по вопросам **безопасности**, хранения **данных**, **оптимизации**, распределенных баз **данных** и другим сложным темам

Написана Леоном **Аткинсоном** - ведущим специалистом **в области систем** с **открытым** кодом

Леон Аткинсон

MySQL

БИБЛИОТЕКА ПРОФЕССИОНАЛА

ЛЕОН АТКИНСОН

ВИЛЬЯМС
Москва • Санкт-Петербург • Киев
2002

ББК 32.973.26-018.2.75
А92
УДК 681.3.07

Издательский дом "Вильяме"
Зав. редакцией *С.Н. Тригуб*

Перевод с английс кого и редакция *В.Р. Гинзбурга*

По общим вопросам обращайтесь в Издательский дом "Вильяме" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Аткинсон, Леон.
А92 MySQL. Библиотека профессионала.: Пер. с англ. — М.: Издательский дом "Вильяме", 2002. — 624 с.: ил. — Парал. тит. англ.

ISBN 5-8459-0291-6 (рус.)

В данной книге описана программа MySQL версии 3.23 — самый последний стабильный выпуск, доступный на момент написания книги. Сначала излагаются основы MySQL: запросы, модели баз данных, вопросы нормализации и организации многопользовательской работы, а также транзакции. Затем систематически анализируются все ключевые аспекты программы и демонстрируются эффективные методики взаимодействия с базами данных MySQL посредством языков C, Java, PHP, Perl, Python и др.

Внимательно проверенная ведущим разработчиком MySQL Микаэлем Видениусом, данная книга дает профессионалам именно то, что они ищут: авторитетное, подробное, богатое примерами руководство по разработке приложений MySQL корпоративного уровня.

ББК 32.973.26-018.2.75

ISBN 5-8459-0291-6 (рус.)
ISBN 0-13-066190-2 (англ.)

Оглавление

Содержание

Благодарности

Я бы не смог закончить эту книгу без поддержки моей жены Викки. Помимо того что она прекрасный технический редактор, она постоянно вдохновляла, поддерживала и просто понимала меня.

Кроме Викки я общался с несколькими профессиональными техническими редакторами. Дан Статен (Dan Staten) и Дан Ливингстон (Dan Livingston) дали мне советы с точки зрения общих вопросов редактирования. Микаэль Видениус (Michael Widenius) и Кай Арнё (Kaj Arno), несмотря на свою занятость, внимательно просмотрели текст книги. Благодарю также Синишу Миливоевича (SiniSa MilivojeviC), который обсуждал со мной вопросы библиотеки MySQL++ API.

Я признателен редактору Марку Таубу (Mark Taub) и издательству Prentice Hall. В частности, я благодарен за предоставленную мне свободу в выборе структуры книги. Кроме того, мне было очень приятно, что Марк одобрил мое соглашение с компанией MySQL AB, выразив таким образом свою поддержку сообществу разработчиков открытого программного обеспечения.

Наконец, спасибо всем вам за то, что купили эту книгу!

ВВЕДЕНИЕ

Трудно сделать выбор между теми программными продуктами, которые любишь больше всего. Я много лет с удовольствием занимался Web-программированием на языке PHP с использованием программы MySQL, но сначала решил написать книгу *Core PHP Programming*. Через два года настал черед и книги по MySQL. Оба этих средства — MySQL и PHP — идут рука об руку друг с другом, поэтому я надеюсь, что читатели найдут данную книгу полезным дополнением к упомянутому вышеизданию.

На обложке книги красуется логотип MySQL, который свидетельствует о двух ее особенностях. Во-первых, перед сдачей в печать рукопись была просмотрена Монти Видениусом и Каем Арнё. Это гарантирует достоверность изложенной информации. Во-вторых, компания MySQL AB получает долю доходов от продажи книги. Это означает, что, покупая книгу, вы вносите личный вклад в дальнейшую разработку программы MySQL.

Подобные альтернативные источники финансовой поддержки разработчиков MySQL очень важны, ведь сама программа распространяется на условиях общей лицензии GNU (GNU General Public License, GPL), т.е. плата за нее минимальна. Более того, вы имеете право модифицировать исходные коды программы и делиться этими модификациями с другими людьми при условии соблюдения правил лицензии. Модель распространения программ с открытыми исходными кодами не нова, но популярность она завоевала относительно недавно. Таким образом, в отличие от традиционных разработчиков ПО, компания MySQL AB не может извлекать прибыль из продажи готовых программных пакетов, но она все же остается прибыльной благодаря предоставлению высококлассных услуг по своим продуктам.

В данной книге описаны реляционные базы данных вообще и MySQL в частности. Я предполагаю, что изложенные в книге идеи будут понятны любому человеку, у которого возникла необходимость изучить программу. Предварительный опыт работы с базами данных не требуется, но нужно уметь инсталлировать программы и пользоваться интерпретатором команд. Не помешают также навыки в области программирования.

Я пытался сделать книгу как можно более полезной читателям. По этой причине вторая, справочная, часть расположена посередине, чтобы книга всегда лежала открытой у вас на столе. Кроме того, я старался писать "экономно", не повторяясь и не объясняя многократно одно и то же.

В первой части книги рассматриваются реляционные базы данных как таковые. В первой главе вводится понятие СУБД. Далее следует описание процедур инсталляции сервера MySQL и взаимодействияс ним (главы 2 и 3). Остальные главы этой части посвящены деталям функционирования баз данных, включая знакомство с языком SQL.

Вторая часть представляет собой справочник команд и утилит MySQL. Сюда же входит описание типов данных, переменных и инструкций языка SQL. В последней главе рассмотрена библиотека функций языка C, предназначенных для работы с MySQL.

В третьей части описываются механизмы взаимодействия с сервером MySQL в разных языках программирования, включая C, C++, Java, VBScript, PHP, Perl и Python. От читателей предполагается знакомство с этими языками.

В четвертой части рассматриваются сложные вопросы работы с базами данных. Здесь рассказывается о том, как использовать ресурсы системы и администрировать. сервер MySQL, как осуществлять репликацию базы данных и писать собственные SQL-функции, а также многое другое.

Во всей книге различные ключевые слова и вообще текст, который может появиться на экране компьютера, набраны моноширинным шрифтом. Internet-адреса и адреса электронной почты записываются курсивом, например *caremysql@leonatkinson.com*.

В книге рассмотрена программа MySQL версии 3.23 — ее самый последний стабильный выпуск, доступный на момент написания книги. Возможно, к тому времени, когда вы будете читать книгу, версия 4.0 уже будет объявлена стабильной. Соответственно, я старался помечать те места, где функции версии 3.23 могут подвергнуться изменениям.

Часть I

MYSQL И РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

В этой части книги представлены основные концепции реляционных баз данных. Глава 1, "Введение в MySQL", содержит краткий обзор функциональных возможностей сервера баз данных. В главе 2, "Инсталляция MySQL", рассказывается об инсталляции программы MySQL. В главе 3, "Взаимодействие с MySQL", речь идет о программах, посредством которых осуществляется взаимодействие с сервером. Сюда входят утилиты командной строки, программы с графическим интерфейсом, а также приложения ODBC.

В главе 4, "Концепции баз данных", обсуждается эволюция баз данных и вводятся основные понятия. Из этой главы читатели узнают о том, какие модели баз данных предшествовали реляционной модели, а какие — придут ей на смену. В главе 5, "Реляционная модель", содержится описание реляционной модели и особенностей ее реализации в MySQL. В главе 6, "Язык SQL", рассматривается язык SQL, применяемый для манипулирования данными в MySQL и большинстве других реляционных СУБД.

В главе 7, "Проектирование баз данных", рассказывается о проектировании баз данных, начиная от составления спецификации и заканчивая построением диаграмм отношений между объектами. В главе 8, "Нормализация", обсуждаются особенности процесса нормализации, вследствие которого устраняется ненужная избыточность данных. В главе 9, "Транзакции и параллельные вычисления", рассматриваются вопросы, связанные с одновременным доступом к базе данных множества пользователей. Излагаются методики устранения возникающих при этом проблем за счет транзакций и блокировок.

ВВЕДЕНИЕ В MYSQL

В этой главе…

Принципы использования баз данных
Преимущества баз данных
Недостатки баз данных
Зачем нужна программа MySQL
История MySQL

Глава 1

В настоящей главе рассматриваются общие концепции баз данных и описываются принципы их реализации в MySQL. Будет рассказано о том, каким образом в MySQL решаются основные проблемы управления данными и чем MySQL отличается от других СУБД.

Принципы использования баз данных

В наши дни люди часто говорят о базах данных. Компьютеры составляют неотъемлемую часть современного общества, поэтому нередко можно услышать фразы вроде "Я поищу твою запись в базе данных". И речь идет не о больших ящиках, где хранятся груды папок, а о компьютерных системах, предназначенных для ускоренного поиска информации.

Компьютерные системы хранения

Компьютеры так прочно вошли в нашу жизнь, потому что их можно запрограммировать на выполнение утомительных, повторяющихся операций и решение задач, которые нам самим было бы не под силу решить без их вычислительной скорости и емкости информационных носителей. Помещение информации на бумагу и разработка схемы хранения бумаг в папках и картотеках — достаточно четко отработанный процесс, но многие вздохнули с облегчением, когда задача свелась к перемещению электронных документов в папки на жестком диске.

Одной из функций баз данных является упорядочение и индексация информации. Как и в библиотечной картотеке, не нужно просматривать половину архива, чтобы найти нужную запись. Все выполняется гораздо быстрее.

Не все базы данных создаются на основе одних и тех же принципов, но традиционно в них применяется идея организации данных в виде записей. Каждая запись имеет фиксированный набор полей. Записи помещаются в таблицы, а совокупность таблиц формирует базу данных.

Давайте рассмотрим работу с базами данных на примере автосалона. Кен, торговец автомобилями, владеет более чем 100 машинами. Естественно, Кен не может помнить детальное описание каждой из них, поэтому он решает создать базу данных. В ней будет содержаться таблица с описанием каждого автомобиля, включая производителя, модель, год выпуска и ряд других параметров. В США у каждого автомобиля есть уникальный идентификационный номер (VIN, Vehicle Identification Number). Он также занесен в таблицу, чтобы можно было различать модели с одинаковыми параметрами.

СУБД

Для работы с базой данных необходима СУБД (система управления базами данных), т.е. программа, которая берет на себя все заботы, связанные с доступом к данным. Она содержит команды, позволяющие создавать таблицы, вставлять в них записи, искать и даже удалять записи.

MySQL — это быстрая, надежная и недорогая СУБД. Бизнес Кена невелик, поэтому он не может себе позволить приобрести дорогую корпоративную систему, но точно так же он не может допустить, чтобы с базой данных произошел крах. Поэтому он выбрал открыто распространяемый пакет, з надежной работе которого он больше уверен. Кроме того, Кен обнаружил, что есть много бесплатных ресурсов, посвященных поддержке MySQL. Если же ему понадобится профессиональная поддержка с гарантией, он сможет оплатить соответствующую услугу в компании MySQL AB.

MySQL берет на себя заботу об эффективном хранении записей и таблиц на жестком диске. Кен избавлен от этого. Ему лишь нужно вводить правильные команды.

MySQL, как и многие другие СУБД, функционирует по модели "клиент/сервер". Под этим подразумевается сетевая архитектура, в которой компьютеры играют роли клиентов либо серверов. Серверы обычно обладают более мощными ресурсами и предназначены для предоставления услуг группам клиентов. Именно на серверах концентрируются вычислительные мощности и данные, тогда как на клиентах располагаются интерфейсные программы, посредством которых пользователи получают доступ к ресурсам сервера. На рис. 1.1 изображена схема передачи информации между компьютером Кена и жестким диском сервера.

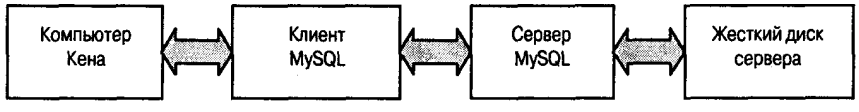


Рис. 1.1. Схема передачи данных в архитектуре "клиент/сервер"

Кен работает с клиентской программой MySQL, которая представляет собой утилиту командной строки. Эта программа подключается к серверу по сети. Команды, выполняемые сервером, обычно связаны с чтением и записью данных на жестком диске.

Клиентские программы могут работать не только в режиме командной строки. Есть и графические клиенты, например MySQL GUI. О них пойдет речь в главе 3, "Взаимодействие с MySQL".

Язык баз данных

MySQL взаимодействует с базой данных на языке, называемом SQL (Structured Query Language — язык структурированных запросов). Одни люди произносят эту аббревиатуру как "сиквел", другие, и я в том числе, — как "эскюэль". Последний вариант, похоже, нравится и разработчикам MySQL. Они заявляют, что название программы должно произноситься так: "май-эскюэль".

Первое, что нужно сделать Кену, разобравшись с полями записей, — это создать таблицу. Данной цели служит команда, показанная в листинге 1.1.

```
CREATE TABLE car (
   VIN VARCHAR(17) NOT NULL,
   Make VARCHAR(16) NOT NULL,
   Model VARCHAR(16) NOT NULL,
   ModelYear INT(16) NOT NULL,
   WholesalePrice FLOAT(6,2) NOT NULL,
   Color VARCHAR(S) NOT NULL,
   Mileage INT(11) NOT NULL,
   Comments TEXT,
   PRIMARY KEY(VIN)
);
```

С этой командой стоит познакомиться поближе. После названия команды CREATE TABLE указано имя таблицы: car. Далее в фигурных скобках идет список разделенных запятыми строк, описывающих поля таблицы.

Первое поле называется VIN и его тип — VARCHAR, т.е. строка символов переменной длины. Максимальное число символов в строке — 17. Спецификатор NOT NULL говорит о том, что строка не может быть пустой, т.е. пользователь обязательно должен ввести в эту строку какие-то данные.

Некоторые поля имеют другой тип. Например, поле WholesalePrice содержит число с плавающей запятой. Формат этого числа таков: шесть цифр до запятой, и две — после. Поле ModelYear содержит целое число, состоящее не более чем из четырех цифр.

Последняя строка списка представляет собой определение не поля, а индекса. В ней сообщается о том, что поле VIN является первичным ключом таблицы. Как уже говорилось выше, идентификационный номер автомобиля должен быть уникальным. А чтобы гарантировать уникальность значений в столбце, его нужно сделать первичным ключом. MySQL не позволит вставить в таблицу две записи с одинаковым значением поля VIN.

Запросы к базам данных

На языке баз данных команды, обращающиеся к базе, называются инструкциями либо запросами. *Инструкция*- это более общий термин. *Запросом* обычно считается такая инструкция, которая возвращает информацию (их еще называют запросами на выборку). Запрос можно рассматривать как вопрос, задаваемый базе данных, напри-

мер: "Существуют ли записи, в которых цвет автомобиля указан белым?" Если такие записи имеются, они будут выданы в виде результатов запроса.

Запрос, показанный в листинге 1.2, представляет собой инструкцию SELECT. Она отбирает записи, соответствующие следующему критерию: поле Color записи содержит строку 'White'. Но возвращается не вся запись целиком, а лишь четыре поля: VIN, Make, Model и ModelYear.

```
SELECT VIN, Make, Model, ModelYear
  FROM car
 WHERE Color = 'White';
```

Обратите внимание на то, что инструкция SELECT напоминает предложение, записанное по-английски. Такова особенность языков четвертого поколения, к которым относится и SQL. К языкам первого поколения относятся платформнозависимые машинные коды, напрямую воспринимаемые центральным процессором. Второе поколение — это ассемблерные языки. Языки третьего поколения считаются высокоуровневыми, и на них работают большинство программистов. В качестве примера можно привести C и PHP. Языки четвертого поколения еще на шаг приближены к естественным человеческим языкам.

Показанный выше запрос можно модифицировать как угодно, но результат пока что будет одним и тем же. СУБД не найдет ни одной записи, так как таблица пуста! Следовательно, нужно что-то добавить в нее. Для этого предназначена инструкция INSERT. Данная инструкция относится к семейству запросов на изменение и официально называется *запросом на добавление записей*.

В листинге 1.3 показана инструкция, которая добавляет запись в таблицу car. Порядок значений соответствует порядку столбцов в таблице.

```
INSERT INTO car VALUES (
    '12345678901234567',
    'Plymouth',
    'Roadrunner',
    1969,
    5500.00,
    'Blue',
    148123,
    'Unrestored'
);
```

Такие инструкции должны быть выполнены для всех автомобилей, находящихся в распоряжении Кена.

Имея заполненную таблицу, Кен может производить в ней более предметный поиск. Например, если клиент спросит, есть ли в продаже автомобили "Форд" по цене меньше $10000, Кен сможет проверить это с помощью показанного ниже запроса (листинг 1.4).

```
SELECT *
  FROM car
 WHERE Make = 'Ford'
   AND WholesalePrice < 9000.00;
```

Звездочка (*) означает выборку всех полей Записи. Кен рассчитывает заработать на продаже каждого автомобиля не менее 1000 долларов, поэтому в запросе фигурирует откорректированная цена.

Просматривая записи, Кен обнаруживает, что у одного из автомобилей неправильно указан цвет. Для исправления записи нужно ввести инструкцию UPDATE, называемую *запросом на обновление записей*. Кен запоминает идентификатор автомобиля и вводит запрос, показанный в листинге 1.5.

```
UPDATE car
   SET Color = 'White'
 WHERE VIN = '10203040506070809';
```

Как и в инструкции SELECT, предложение WHERE ограничивает круг записей, с которыми имеет дело инструкция. В данном случае это одна запись, так как поле VIN является первичным ключом таблицы.

В конце недели Кен продал два автомобиля, поэтому он решает удалить их записи из таблицы. Для этого предназначена инструкция DELETE, представляющая собой *запрос на удаление записей* (листинг 1.6).

```
DELETE
  FROM car
 WHERE VIN IN ('12345678901234567', '10203040506070809');
```

В этом запросе используется идея множества: из таблицы удаляются записи, в которых поле VIN равно одному из значений, входящих в множество.

Абстракция

Работая с таблицей car, Кен вскоре обнаруживает проблему. Дело в Том, что в поле Color хранится текстовое название цвета, вводимое пользователем. Во-первых, утомительно вводить все время одни и те же названия, во-вторых, легко можно допустить ошибку в написании. Нужно придумать лучшую организацию таблицы.

В таблицах баз данных можно хранить не только скалярные значения, но и абстрактные указатели на другие таблицы. Кен находит выход: присвоить каждому цвету

номер. Для этого нужно создать еще одну таблицу с двумя полями: в одном хранится код цвета, во втором — его название. В листинге 1.7 показана инструкция, создающая такую таблицу.

```
CREATE TABLE car_color (
    ColorCode INT(6) NOT NULL AUTO_INCREMENT,
    NameVARCHAR(16),
    PRIMARY KEY(ColorCode)
);
```

Номер цвета является целым числом, а столбец ColorCode — первичным ключом таблицы. Кен не хочет самостоятельно назначать эти номера, поэтому он возлагает данную задачу на СУБД, помечая поле ColorCode ключевым словом AUTO_INCREMENT. При добавлении очередной записи в таблицу car_color MySQL автоматически запишет в это поле номер, на единицу больший предыдущего.

Для начала необходимо сформировать запрос к таблице car, чтобы получить список имеющихся названий цветов (листинг 1.8).

```
SELECT DISTINCT Color
  FROM car;
```

Ключевое слово DISTINCT заставляет MySQL удалить из таблицы результатов дублирующиеся значения. Теперь можно ввести серию запросов, вставляющих записи в таблицу car_color по одной за раз. Но это очень неуклюжий подход. Более эффективное решение — объединить инструкции INSERT и SELECT, создав *подчиненный запрос*. Результаты инструкции SELECT будут непосредственно вставляться в таблицу car_color (листинг 1.9).

```
INSERT INTO car_color (Name)
    SELECT DISTINCT Color
      FROM car;
```

После названия таблицы car_color в круглых скобках указан столбец, куда заносятся данные (Name). Обычно в пропущенные поля вставляется специальная константа NULL, обозначающая отсутствующее значение. Но поле ColorCode помечено спецификатором NOT NULL, т.е. значения NULL в нем недопустимы. С другой стороны, в определении поля стоит ключевое слово AUTO_INCREMENT, а это означает, что в поле будут автоматически вставляться целые числа: в первой записи это будет число 1, во второй — 2 и т.д.

На следующем этапе требуется перестроить таблицу car, модифицировав определение столбца Color (листинг 1.10).

```
ALTER TABLE car
     CHANGE Color
  ColorCode INT(6) NOT NULL;
```

К сожалению, подобная модификация означает удаление всей информации из столбца Color. MySQL поменяет тип столбца, подставив во все поля значение 0. Далее Кену придется ввести последовательность инструкций UPDATE, чтобы создать в таблице car правильные ссылки на новую таблицу car_color.

После обновления таблицы car в ней будут храниться не названия цветов, а их коды (в поле ColorCode). Белый цвет будет иметь, к примеру, код 1, а темно-синий цвет — код 13. Запомнить соответствие между номерами и названиями цветов едва ли возможно. Необходимо, чтобы при выполнении запросов к таблице car отображались именно названия цветов, а не их коды.

Решение проблемы заключается в выполнении операции *объединения* (join) двух таблиц. Это особая разновидность инструкции SELECT, в которой каждая запись одной таблицы сравнивается по определенному критерию с каждой записью другой таблицы. Есть несколько типов объединений, среди которых чаще применяется *левое внешнее объединение (листинг 1.11).*

```
SELECT car.VIN, car_color.Name
  FROM car LEFT JOIN car_color
    ON car.ColorCode = car_color.ColorCode
 WHERE car.Make = 'Plymouth';
```

Возможные результаты такого запроса представлены в листинге 1.12.

```
+-------------------+------+
| VIN               | Name |
+-------------------+------+
| 12345678901234567 | Red  |
+-------------------+------+
```

Синтаксис запроса с объединением сложнее, чем у других запросов, потому что в предложении FROM указывается не обычная, но абстрактная таблица, получаемая в результате объединения исходных таблиц. В данном случае абстрактная таблица формируется следующим образом.

1) Каждая запись таблицы car сравнивается с каждой записью таблицы car_color.
2) Если поле ColorCode таблицы car совпадает с полем ColorCode таблицы car_color *(объединение по равенству),* создается итоговая запись, состоящая из полей первой таблицы, к которым присоединены поля второй таблицы.
3) Для каждой записи таблицы car, полю ColorCode которой не найдено соответствие в таблице car_color, создается итоговая запись, состоящая из полей первой таблицы, к которым вместо полей второй таблицы присоединены значения NULL *(левое объединение).*
4) Из объединенной таблицы отбираются записи, соответствующие условию WHERE.
5) В результаты запроса включаются поля, заданные в предложении SELECT.

Между таблицами car и car_color нет отношения "один к одному", так как может быть несколько автомобилей одинакового цвета. Подробнее о составлении такого рода объединений речь пойдет в последующих главах.

В листинге 1.11 несложно заметить одну особенность многотабличных запросов: при ссылке на поля таблиц следует избегать неоднозначности. Например, обе таблицы содержат поле ColorCode. Чтобы явно указать, к какой таблице относится поле, нужно задать полное имя поля. Синтаксис полного имени таков: *имя_та*блицы.*.имя_поля.*

Преимущества баз данных

Для постоянного хранения информации в компьютерах используются жесткие диски. Данные на них записываются в виде файлов, которые являются аналогами документов, хранящихся в папках и картотеках большинства офисов. Необходимо понять, при каких обстоятельствах выгоднее содержать информацию в базах данных, а не просто в файлах.

Традиционные подходы к управлению данными

Когда объем данных невелик и составлять по ним отчеты практически не требуется, подойдет и бумажная система регистрации документов. Представьте себе картотечный шкаф с тремя ящиками по 10 папок в каждом. Надпись на папке определяет тип содержащихся в ней документов. Если, к примеру, нужно найти прошлогодние счета за электроэнергию, необходимо поискать в первом ящике папку с надписью "Счета за электроэнергию", просмотреть содержащиеся в ней документы и отобрать среди них 12 прошлогодних счетов.

Этот процесс относительно прост, но если он повторяется многократно или появляются сложные формы отчетности, возникает желание его автоматизировать. Бумажные документы превращаются в файлы на диске, а для работы с ними пишутся программы на таком языке, как C или Perl.

Файлы группируются в каталоги на основании определенных логических принципов. Содержимое файлов структурируется таким образом, чтобы программы могли быстро извлекать из них нужные данные. Для каждого типа файлов необходимо написать набор подпрограмм, манипулирующих этими файлами, включая чтение, запись и об-

новление данных. Для каждого отчета тоже требуется специальная подпрограмма, собирающая соответствующие данные и представляющая их в требуемом виде.

Все это приводит к немалым затратам времени и усилий. В домашних условиях можно все оставить в бумажном виде, но с точки зрения ведения делопроизводства компьютерная автоматизация процессов очень важна. Ведь чем больше информации накапливается, тем более обоснованные решения можно принимать.

В бизнесе существует тенденция постоянного усложнения отчетов. Важно знать о своем бизнесе больше, чем конкуренты — о своем. Однако высокая стоимость получения отчетов может свести на нет все их преимущества. Более того, если задержка между определением потребности в отчете и составлением самого отчета будет слишком большой, может оказаться, что к моменту получения отчет устареет и станет бесполезным.

Базы данных были разработаны как раз с целью решения проблем, возникающих при создании специализированных подпрограмм для обработки данных и построения отчетов.

Унификация средств доступа

Первое преимущество баз данных — это наличие унифицированного интерфейса. Не нужно "изобретать колесо" и постоянно создавать новые модули манипулирования данными. Все обращения к базе данных централизуются на уровне СУБД.

В базе данных информация представлена единообразно. Сведения о том, как структурированы данные и как они связаны друг с другом, хранятся в самой базе в виде метаданных. Это позволяет СУБД иметь в своем распоряжении универсальные подпрограммы.

Работа с обычными ("плоскими") файлами, которая ведется на языках третьего поколения, требует профессиональных навыков программирования. Такие файлы не имеют единого интерфейса. Каждая подпрограмма трактует файл по-своему в зависимости от его атрибутов. Следствием подобной адаптации является то, что любое изменение структуры данных вызывает каскадные изменения программ, зависящих от этой структуры. Представьте, что поле, содержавшее пятизначный почтовый код, должно теперь содержать девятизначный код. Во-первых, существующие данные придется преобразовать в новый формат с помощью специальной программы. Во-вторых, потребуется переписать любую программу, работающую с файлом такого формата. Особенностью подобного подхода является то, что в программах делаются явные предположения о структуре данных.

Базы данных более гибки в использовании. Они позволяют приложениям работать с данными на логическом уровне, игнорируя их физическую структуру. Изменение формата данных может не вызвать никаких изменений в интерфейсе взаимодействия с ними, а следовательно — в самих приложениях.

Наконец, современные базы данных поддерживают специальный язык запросов, позволяющий создавать произвольные выборки данных. Наличие языка запросов упрощает рядовым пользователям задачу формулирования инструкций для построения отчетов.

Повышение производительности

Базы данных повышают производительность обработки информации за счет концентрации функций манипулирования данными в одном модуле, который можно оптимизировать. Базы данных берут на себя ответственность за управление информацией наиболее эффективным способом. Детали того, как это происходит на физическом уровне, скрыты от пользователя, который видит лишь логическую модель данных.

Благодаря тому что функции обработки данных сконцентрированы в одном модуле, этот модуль можно применять для доступа ко всем базам данных. А это, в свою очередь, означает, что улучшение работы такого модуля приводит к ускорению всех приложений, обращающихся через него к базам данным.

Другой источник повышения производительности — это возможность многопользовательской работы. Базы данных контролируют все обращения к хранимой информации и могут выполнять несколько запросов одновременно. Сравните с обычными файлами, которые в конкретный момент времени могут быть открыты для записи (а зачастую и для чтения) только одним процессом.

Централизация средств управления данными ведет также к усилению их безопасности. СУБД может назначать аутентифицированным пользователям определенные права. Она, к тому же, способна выполнять резервное копирование данных, не прерывая работу пользователей.

Усиление целостности

Базы данных упрощают задачу структурирования данных, позволяя избегать ненужного дублирования. Например, благодаря операции объединения таблиц появляется возможность сосредоточивать однотипную информацию в одной таблице, ссылаясь на нее в других таблицах по уникальным ключам. Это помогает устранять конфликты, возникающие в результате неполных обновлений. Например, если бы адрес клиента хранился в двух разных местах, его можно было бы изменить в каком-то одном месте и забыть это сделать в другом. В реляционной базе данных можно хранить все адреса в одной таблице, ссылаясь на нее из других таблиц в случае необходимости.

СУБД контролирует целостность данных на основании формальных определений, хранимых в самой базе. Например, если при создании таблицы было указано, что такое-то поле хранит даты, то СУБД откажется записывать в это поле значения других типов. С точки зрения физического хранения данных в поле даты можно записать и целочисленное значение, но СУБД будет следовать логическим правилам, которые были заданы пользователем. MySQL, если это возможно, выполняет операцию приведения типов в случае несовпадения или же просто подставляет значение по умолчанию.

СУБД способна также контролировать отношения между таблицами. Можно, например, задать правило, по которому поля двух таблиц станут связанными. СУБД гарантирует, что значения подчиненного поля всегда будут попадать в диапазон значений главного поля.

Недостатки баз данных

Несмотря на свои многочисленные преимущества, базы данных не решают всех проблем, связанных с управлением данными. Чтобы они приносили реальную пользу, их нужно правильно использовать. На самом деле ничто не мешает пользователю продублировать данные в нескольких местах — ничто, кроме тщательного выполненного проектирования.

Определенную сложность представляют различия, связанные с реализацией конкретных СУБД. Существуют международные стандарты языка SQL, но мало какой разработчик СУБД устоит перед искушением дополнить стандарт собственными расширениями. В результате перенос базы данных из одной СУБД в другую часто оказывается затруднительным. Подробнее об этом рассказывается в главе 28, "Перенос данных в разные СУБД". Если заранее известно о том, что базу данных придется переносить в другую СУБД, постарайтесь не использовать функциональные возможности, специфичные для MySQL. Это существенно упростит процесс переноса.

Зачем нужна программа MySQL

Прежде чем рассматривать особенности использования программы MySQL, необходимо понять, почему есть смысл работать именно с ней, а не cOracle, к примеру, или PostgreSQL, или любой другой известной СУБД. Многие находят программу MySQL особенно привлекательной из-за того, что ее легко изучить. Она также оказывается достаточно гибкой в самых разных ситуациях. Там, где другие СУБД навязывают одну модель данных, MySQL предлагает варианты.

Наиболее важным достоинством MySQL является то, что это — настоящий сервер баз данных. Далее в книге мы рассмотрим характеристики реляционных баз данных и увидим, как в MySQL реализуются требования к реляционным СУБД. В MySQL есть все, что необходимо для изучения концепций базданных.

MySQL — это быстрая СУБД. На Web-узле *www.mysql.com* приведены результаты тестов, в которых сравнивается производительность MySQL и других реляционных СУБД. Во многих случаях разница существенна, а повышенная производительность ценится всегда.

Одна из причин, по которой MySQL обладает таким преимуществом, как скорость, заключается в тщательно продуманной архитектуре. В ней отражено конкретное практическое стремление создать СУБД в чистом виде. Программа избавлена от разного рода излишеств, в отличие от многих СУБД, выглядящих словно новогодняя елка, обвешанная гирляндами. Все функциональные возможности программы четко обоснованы, так как разработчики стремились обеспечить максимальную простоту и гибкость ее использования.

Команда разработчиков MySQL старается реализовывать все, что предписывается стандартом ANSI. Но если та или иная функция пользователю не нужна, он может отключить ее на этапе компиляции. Это, опять-таки, свидетельствует о гибкости программы.

Основная движущая сила, делающая программу MySQL такой, какая она есть, заложена в особенностях процесса разработки. MySQL — это открыто распространяемая программа, причем один из лидеров в своей области. Любой пользователь может про-

смотреть каждую строку кода программы и исправить ее ошибки. Многие так и поступают. Общее руководство процессом осуществляет команда разработчиков, которые знают направление своего пути и добавляют к программе новые функции так, чтобы они оставались в духе всего проекта. Небольшая группа программистов осуществляет официальную проверку всего кода MySQL, гарантируя высокое качество программы.

Стоимость приобретения и эксплуатации MySQL очень невелика, как и у всякого открытого ПО. Программу можно бесплатно загрузить с нескольких Web-узлов, к тому же она входит во многие дистрибутивы Linux. Существуют многочисленные телеконференции, посвященные вопросам поддержки пользователей MySQL. Сотрудники компании MySQL AB постоянно проверяют сообщения, поступающие в список рассылки *bugs@lists.mysql.com,* и бесплатно устраняют возникающие проблемы. Кроме того, за умеренную плату компания оказывает гарантированную, персональную поддержку всем желающим.

MySQL — надежная СУБД. Так как ее исходные коды доступны для всеобщего обозрения, пользователи регулярно находят и исправляют ошибки по мере их появления.

Другим следствием открытости кода стала доступность MySQL для множества платформ. Эта программа может работать в большинстве версий UNIX, Linux, Windows и даже в менее популярных операционных системах, например в OS/2.

История MySQL

Однажды Микаэлю "Монти" Видениусу пришла в голову мысль добавить SQL-модуль в качестве интерфейса к своей старой базе данных, которую он на протяжении 15 лет вел в компании TCX DataKonsult AB. Он решил использовать для этой цели открытую реляционную СУБД mSQL, но ему не удалось заинтересовать ее автора, Дэвида Хьюза (David Hughes). Тогда Монти начал создавать собственную реляционную СУБД, клиентский интерфейс которой был смоделирован на основе API-функций mSQL, чтобы в новую систему можно было перенести приложения, написанные для mSQL. В планы Монти никогда не входило реализовывать стандарт языка SQL целиком, но после того как он поделился исходными кодами программы с сообществом разработчиков, ответная реакция оказалась ошеломляющей.

Первая версия MySQL, тогда еще интерфейса к старой базе данных Монти, была закончена в мае 1995 года. На ее написание ушло три месяца. С этого момента программа MySQL начала свой путь к тому, чтобы стать самой популярной СУБД, используемой в Internet.

Компанию TCX DataKonsult AB впоследствии переименовали в MySQL AB, и с сегодня Монти является руководителем ее технического отдела. Эта шведская компания целиком посвятила себя разработке и поддержке программы MySQL. С самого начала компания стала прибыльной за счет оказания платной поддержки, предоставления платных консультаций и продажи лицензий на встроенную версию MySQL. Это прекрасный пример того, что модель распространения программ с открытыми кодами является вполне жизнеспособной.

В июне 2000 г. программа MySQL стала доступна на условиях общей лицензии GNU (GNU General Public License, GPL). Это дает возможность каждому пользователю улучшать программу и передавать ее своим коллегам без каких-либо лицензионных отчислений.

ИНСТАЛЛЯЦИЯ MYSQL

В этой главе.

Загрузка дистрибутива
Инсталляция с помощью менеджера пакетов RedHat Linux
Инсталляция в Windows
Инсталляция вручную
Компиляция программы
Предоставление привилегий

Глава 2

Пользователю базы данных необязательно знать, как инсталлировать MySQL. В крупных организациях есть системные администраторы, которые этим занимаются. Что касается разработчиков, то им нужно понимать особенности данного процесса. Именно здесь у них появляется доступ к различным конфигурационным установкам, с помощью которых можно настроить производительность программы. Естественно, необходимо обладать правами администратора на том компьютере, где MySQL устанавливается в виде сервиса, запускаемого автоматически. Программу можно запускать также из персональных учетных записей.

Загрузка дистрибутива

MySQL можно инсталлировать двумя способами: скомпилировав исходные коды программы или воспользовавшись предварительно скомпилированными двоичными файлами. Первый вариант допускает больше возможностей в плане конфигурации, но более продолжителен. Второй вариант удобнее, так как есть готовые дистрибутивы для многих операционных систем. На момент написания книги существовали версии MySQL для FreeBSD, HP-UX, IBM AIX, Linux, MacOS X, SCO, SGI Irix, Solaris и многих вариантов Microsoft Windows.

Информацию обо всех дистрибутивах можно получить на Web-узле *www.mysql.com*. Там же публикуются последние новости о программе.

Проверка исходных требований

Если инсталлируются двоичные файлы, следует убедиться в том, что система соответствует исходным требованиям программы. Важным моментом является поддержка потоков. При инсталляции в старых версиях некоторых операционных систем MySQL требует наличия отдельной библиотеки потоковых функций POSIX. POSIX — это международный стандарт, определяющий работу системных сервисов, а потоки — это механизм, позволяющий программам выполнять несколько задач одновременно.

Современные операционные системы поддерживают стандарт POSIX. Если вы все же не уверены, проверьте в интерактивной документации, будет ли программа MySQL работать в данной версии операционной системы.

Выбор версии

Команда разработчиков MySQL публикует тестовые и стабильные версии дистрибутивов отдельно. Информацию о статусе той или иной версии программы можно найти на Web-узле. Эти же сведения закодированы в названии дистрибутива. На момент написания книги последняя стабильная версия имела номер 3.23.39.

В целом лучше выбирать самую новую версию. Исключение составляют лишь альфа-версии, в которых впервые вводятся новые функциональные возможности. Программа MySQL отвечает самым высоким критериям качества и надежности, поэтому ее бета-версии зачастую вполне сопоставимы с финальными версиями других программ.

Инсталляция с помощью менеджера пакетов RedHat Linux

Если программа MySQL инсталлируется в Linux, то лучше всего воспользоваться модулем RPM (RedHat Packet Manager— менеджер пакетов RedHat). MySQL работает в Linux версий 2.0 и выше. Тестирование программы выполнялось в RedHat 6.2. В программе используется библиотека `glibc`, подключаемая статически. Если в системе установлена более старая версия библиотеки, программу придется скомпилировать заново.

Ниже приведено описание доступных модулей RPM.

- `MySQL-3.23.39-1.1386.rpm`

  Содержит все файлы, необходимые для запуска сервера MySQL, включая клиентские программы.

- `MySQL-3.23.39-1.src.rpm`

  Содержит все исходные коды MySQL.

- `MySQL-bench-3.23.39-1.1386.rpm`

  Содержит программы, предназначенные для тестирования производительности MySQL. Для запуска тестов необходим основной дистрибутив, а также интерпретатор Perl.

- `MySQL-client-3.23.39-1.1386.rpm`

  Содержит лишь клиентские программы.

- `MySQL-devel-3.23.39-1.1386.rpm`

  Содержит библиотеки и файлы заголовков, необходимые для компиляции клиентских программ.

- `MySQL-shared-3.23.39-1.1386.rpm`

  Содержит совместно используемые библиотеки для клиентских программ.

Опытные пользователи Linux знают, что флаг `-i` служит программе `rpm` указанием инсталлировать пакет. Таким образом, основной модуль MySQL инсталлируется следующей командой:

```
rpm -i MySQL-3.23.39-1.1386.rpm
```

В результате инсталляции в каталог /etc/rc.d добавляется файл сценария, содержащий команду запуска сервера MySQL после перезагрузки компьютера. Однако сам серверный демон запускается немедленно.

По окончании инсталляции потребуется изменить стандартные привилегии доступа к базам данных, о чем пойдет речь в конце главы.

Можно также инсталлировать модуль RPM с исходными кодами программы. В этом случае воспользуйтесь опцией --rebuild, чтобы подготовить бинарный модуль.

Обычно пользователи инсталлируют лишь модули MySQL-3.23.39-1.i386.rpm и MySQL-client-3.23.39-1.i386.rpm. Для тех, ктособираются писать собственные клиентские программы, потребуется также модуль MySQL-devel-3.23.39-1.i386.rpm.

Инсталляция в Windows

Программа MySQL распространяется и в виде ZIP-архива, содержащего набор инсталляционных файлов. Перед извлечением файлов из архива создайте отдельный каталог, например c:\windows\Desktop\mysql, так как в архиве нет информации о путевых именах файлов.

Чтобы приступить к инсталляции, выполните двойной щелчок на файле setup.exe, после чего начнут появляться различные диалоговые окна. Первый вопрос, на который предстоит ответить, касается папки, куда должна быть помещена программа. По умолчанию предлагается папка c:\mysql. Можно выбрать любую другую папку, но в таком случае придется отредактировать конфигурационный файл.

Следующий вопрос касается инсталлируемых компонентов. Если выбрать "типичную" инсталляцию, будут инсталлированы серверный модуль, справочные файлы, а также набор файлов, содержащих описание стандартных привилегий доступа. В случае инсталляции "на выбор" можно будет дополнительно установить утилиты тестирования и библиотеки функций разработки.

Далее начнется собственно установка программы. Если инсталляционный каталог называется не c:\mysql, то поокончании инсталляции нужно будет дополнительно установить файл my. ini. Для этого перейдите в каталог программы и найдите файл my-example.cnf. Скопируйте его в системный каталог (c:\windows или c:\winnt) и переименуйте в my. ini. Можно поступить и по-другому: скопировать файл в корневой раздел диска C: и назвать егоmy.cnf.

Теперь нужно отредактировать этот файл, чтобы переменная basedir указывала на инсталляционный каталог. Если соответствующая строка присутствует в виде комментария, удалите символы комментария. В противном случае добавьте эту строку самостоятельно, например:

```
basedir = d:\mysql
```

Если программа MySQL инсталлируется в Windows NT или Windows 2000, то, возможно, ее нужно запустить в виде сервиса. Для этого требуется перейти в режим командной строки и ввести следующую команду:

```
c:\mysql\bin\mysqld-nt --install
```

Название сервиса появится в окне сервисов панели управления, где можно будет настроить программу на автоматический запуск. Утилита winmysqladmin, входящая в Windows-дистрибутив, позволяет автоматизировать множество задач, включая конфигурацию.

Инсталляция вручную

Если программа MySQL инсталлируется не в Linux или Windows либо еслиуслуги менеджера пакетов не нужны, можно инсталлировать двоичные файлы вручную. Соответствующий дистрибутив распространяется в виде tar-архива, сжатого с помощьюпрограммыgzip.

Первый этап заключается в добавлении нового пользователя, от имени которого будет работать демон MySQL. Естественно, это не должен быть пользователь root. Программе MySQL нельзя предоставлять права суперпользователя, и никакие компромиссы здесь недопустимы. Можно, например, создать группу mysql и одноименного пользователя с помощью команд addgroup и adduser либо groupadd и useradd, в зависимости от версии UNIX. Ниже показан пример для RedHat Linux:

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

Обычно начальным каталогом MySQL выбирают /usr/local/mysql. Послераспаковки архива будет создан каталог, имя которого совпадает с именем дистрибутива, поэтому удобнее всего просто создать символическую ссылку mysql. Вот как этоделается:

```
cd /usr/local
tar xvfz mysql-3.23.35-pc-linux-gnu-i686.tar.gz
ln -s mysql-3.23.35-pc-linux-gnu-i686 mysql
cd mysql
```

Далее необходимо запустить сценарий mysql_install_db, находящийся в каталоге scripts. Он создаст базу данных с описанием существующих привилегий и тестовую базу данных.

Как правило, программа MySQL инсталлируется от имени пользователя root, поэтому следующий шаг заключается в изменении владельца всех файлов программы:

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

Теперь можно запустить демон MySQL с помощью сценария safe_mysqld. Следующая команда запускает демон от имени пользователя mysql:

```
/usr/local/mysql/bin/safe_mysqld --user=mysql &
```

Если нужно, чтобы сервер MySQL запускался всякий раз после перезагрузки компьютера, добавьте соответствующую строку в файл /etc/rc.d/rc. local или же скопируйте сценарий mysql. server в каталог /etc/init.d и создайте правильные символические ссылки на него. В комментариях к файлу support-files/mysql. server рекомендуются такие ссылки: /etc/rc3.d/S99mysqlи/etc/rcO.d/SOlmysql.

Компиляция программы

Если в вашем распоряжении имеются исходные коды программы, создайте из них двоичные файлы и следуйте приведенным выше инструкциям. Поскольку исходные тексты были подготовлены с помощью утилиты autoconf, для компиляции программы нужно будет ввести последовательность команд configure, make и make install.

По возможности следует избегать перекомпиляции программы. Разработчики MySQL досконально знают все тонкости процесса компиляции, поэтому они умеют создавать максимально оптимизированные исполняемые файлы.

При компиляции исходных кодов появляется возможность подключить компоненты, не встроенные в стандартные инсталляционные пакеты. Для некоторых из них требуются библиотеки функций разработки. Инсталлируйте их до начала компиляции программы.

В главе 26, "Оптимизация", описан процесс перекомпиляции MySQL в целях повышения производительности программы, поэтому мы не будем здесь вдаваться в тонкости компиляции.

Чтобы получить список всех опций конфигурирования, введите команду configure --help.

Предоставление привилегий

Сценарий mysql_install_db предоставляет любому пользователю локального компьютера привилегии, позволяющие регистрироваться на сервере базданных. Сетевые соединения не допускаются. По умолчанию любой пользователь имеет доступ к базе test, а пользователь root имеет полный доступ ко всем базам данных. Если в какой-то из баз хранится важная информация, нужно назначить суперпользователю пароль.

Программа MySQL не работает со списком пользователей, который есть у операционной системы. У нее своя таблица пользователей. Тем не менее если при работе с имеющимися клиентскими программами не ввести имя пользователя в процессе регистрации на сервере, будет подставлено системное имя пользователя.

Чтобы поменять пароль пользователя root, нужно запустить интерпретатор команд MySQL от имени суперпользователя. Данный интерпретатор представляет собой программу mysql, путь к которой должен быть указан в переменной среды PATH. Пользователям Windows придется вводить путевое имя целиком, например c:\mysql\bin\mysql. С помощью опции --user задается имя для регистрации. В нашем случае интерпретатор запускается с помощью такой команды:

```
mysql --user=root mysql
```

Вызвав интерпретатор, необходимо обновить две строки в таблице user, касающиеся пользователя root. Этоделает следующая инструкция:

```
UPDATE user SET Password = PASSWORD('secret')
 WHERE User = 'root';
```

В ответ на эту инструкцию интерпретатор отобразит две модифицируемые записи. Естественно, вместо строки 'secret' следует выбрать более надежный пароль. Этот пароль должен применяться лишь в административных целях.

Далее нужно сообщить серверу об изменении привилегий. Для этого предназначена такая инструкция:

```
FLUSH PRIVILEGES;
```

Любой пользователь может захотеть создать персональную базу данных для собственных экспериментов, но делать это разрешено только пользователю root. Он же может создавать учетные записи новых пользователей и предоставлять им необходимые привилегии. Рассмотрим пример:

```
CREATE DATABASE leon;
 GRANT ALL
        ON leon.*
        TO leon@'%' IDENTIFIED BY PASSWORD('secret');
```

Первая инструкция создает базу данных leon. Вторая инструкция создает учетную запись пользователя leon и предоставляет ему доступ к одноименной базе данных. Пароль для доступа — 'secret'. Пользователь leon может подключаться к базе данных с любого компьютера, даже если он расположен в сети Internet.

ВЗАИМОДЕЙСТВИЕ С MYSQL

В этой главе...

Клиент-серверное взаимодействие средствами TCP/IP

Утилиты командной строки

Графические клиенты

ODBC

Web-интерфейсы

Глава 3

В этой главе рассматриваются различные способы взаимодействия с сервером MySQL. Допускаются клиентские подключения по протоколу IP, а также через сокеты UNIX и именованные каналы. Имеется ряд готовых клиентов. Простейшие из них— это утилиты командной строки, являющиеся частью проекта MySQL. Кроме них есть графические клиенты для различных операционных систем. Те, кто работают в Windows, могут пользоваться специальным драйвером ODBC, а любой, у кого есть Web-броузер, сможет получить в свое распоряжение Web-клиенты.

Различные утилиты командной строки, входящие в состав MySQL, описаны в главе 14, "Утилиты командной строки".

Клиент-серверное взаимодействие средствами TCP/IP

Программа MySQL работает по протоколам TCP/IP, как и другие Internet-сервисы. Соединения различаются по имени узла и номеру порта. По умолчанию используется порт 3306, но это конфигурируемый параметр.

В MySQL применяется архитектура "клиент/сервер". Выделенный сервер способен принимать запросы от множества клиентов. На сервер ложится достаточно большая нагрузка, тогда как клиенты представляют собой всего лишь программные оболочки.

Имя клиентского компьютера, с которого устанавливается соединение, преобразуется в IP-адрес: четыре числа, разделенных точками. Например, специальному имени `localhost` соответствует адрес 127.0.0.1. Иногда такое преобразование выполняется на основании записей локальных таблиц ядра, но чаще всего — с помощью сервера DNS (Domain Name System — система доменных имен).

По умолчанию запрос на подключение поступает через порт 3306. Этот порт ПОСТОЯННО прослушивается сервером MySQL. При ответе на запрос сервер создает сеанс связи с клиентом. За сеансом закрепляются два порта: один будет использоваться для отправки данных, а другой — для их приема.

В ходе сеанса клиент посылает серверу команды, имеющие вид инструкций SQL. В ответ на некоторые инструкции сервер возвращает данные, а клиент форматирует их для отображения на экране.

Утилиты командной строки

В проект MySQL входит много специальных клиентов, а также один клиент общего назначения, называемый mysql. Именно с ним мы познакомимся в данной главе, а рассмотрение остальных клиентов отложим до главы 14, "Утилиты командной строки".

Если программа MySQL инсталлирована правильно, то путь к утилите mysql будет указан в переменной среды PATH. Напомним, что по умолчанию в UNIX утилита будет записана в каталог /usr/local/bin, а в Windows — в каталог c:\mysql\bin.

Если не заданы другие установки, утилитаmysql пытается подключиться к серверу MySQL на узле localhost (порт 3306), используя для доступа к базе данных регистрационное имя текущего пользователя. Как уже говорилось в предыдущей главе, стандартные привилегии доступа программы MySQL разрешают любому пользователю локального узла обращаться к серверу.

В листинге 3.1 показан сеанс работы с утилитой mysql. Строка приглашения ~> выдается интерпретатором bash.

```
-> mysql test
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 6 to server version: 3.23.39

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql> create table book (
    -> title varchar(32),
    -> author varchar(64)
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> insert into book values ('Core MySQL', 'Leon Atkinson');
Query OK, 1 row affected (0.07 sec)

mysql> select * from book;
+            +                +
I title      |  author        |
+            +                +
I Core MySQL | Leon Atkinson I
+            +                +
1 row in set  (0.02 sec)

mysql> exit
Bye
~>
```

Для подключения к удаленному серверу необходимо указать его адрес с помощью опции `host`. Альтернативный порт задается опцией `port`, имя пользователя — опцией `user`, пароль — опцией `password`. Значение опции указывается после знака равенства (листинг 3.2).

```
~> mysql user=leon password host=localhost port=3306
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 3.23.36

Type 'help; or '\h' for help. Type '\c' to clear the buffer

mysql >
```

Последним параметром утилиты mysql является имя базы данных. В листинге 3.1 утилита сразу загрузила базу `test`. Это позволило не вводить команду use.

Другая важная утилита командной строки называется `mysqladmin`. Среди прочего она позволяет создавать новые базы данных. По умолчанию с этой утилитой может работать только пользователь root, но дело втом, что первоначально ему не назначен пароль, поэтому сразу после инсталляции любой пользователь вполне может вводить команды наподобие той, которая показана в листинге 3.3.

```
~> mysqladmin -u root create coremysql
```

Графическиеклиенты

Команда разработчиков MySQL ведет список программного обеспечения, написанного для MySQL. Его можно найти по адресу *www.mysql.com/dffwnloads/contrib.htmL* Многие из приложений являются **специализированными**, но некоторые вполне подходят для общего взаимодействия с базами данных.

Уже упоминавшийся **Синиша Миливоевич,** чле н команды разработчиков, отвечает за сопровождение программы MySQL GUI. Существуют версии для Linux, Win32, **FreeBSD, OpenBSD** и Solaris, которые можно найти по адресу *www.mysql.com/downloads/gui-clients.htm* На момент написания книги программа проходила стадию **бета-тестирования.**

Рис. 3.1 иллюстрирует работу программы, подключенной к тестовой базе данных. Поверх окна запросов открыто окно администратора. В этой программе продублировано большинство функций, которые можно выполнять с помощью утилит командной строки.

Программа MySQL-Maker (рис. 3.2) является условно-бесплатным приложением, в котором база данных представлена в виде дерева, что упрощает навигацию по ней. Здесь можно редактировать определения таблиц и, самое важное, вводить произвольные запросы. Результаты запросов отображаются в табличном виде.

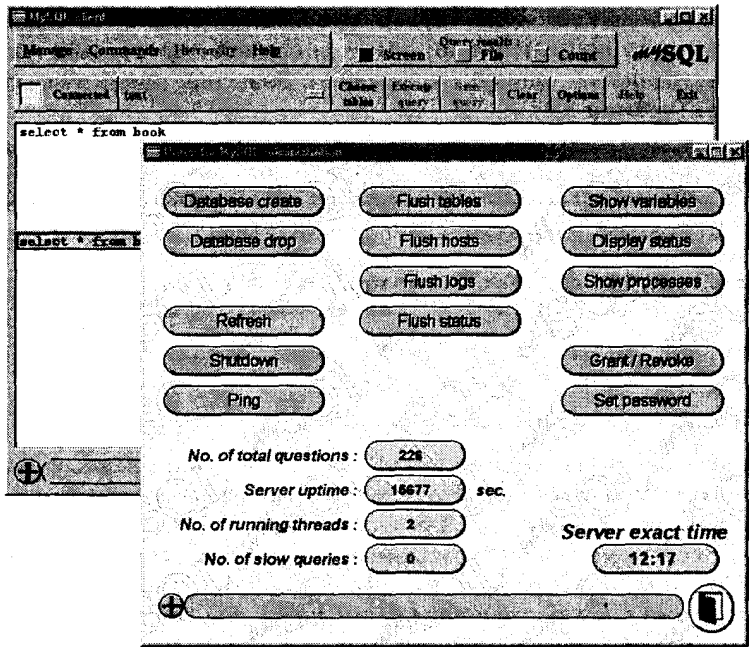


Рис. ***3.1.*** Программа MySQL GUI в Windows 98

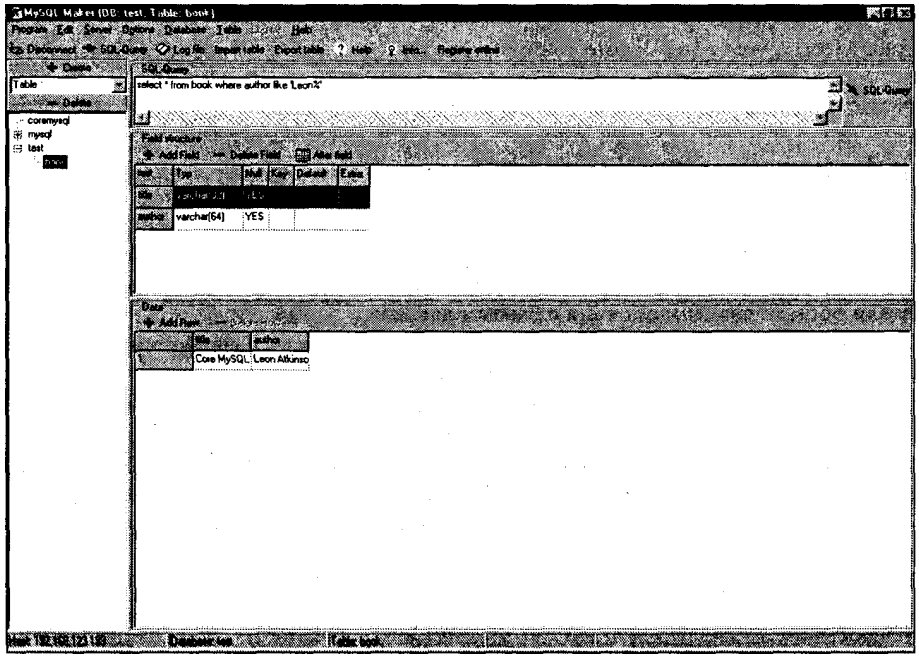


Рис. 3.2. Программа MySQL-Maker

ODBC

Драйверы ODBC доступны не только в Windows, но также в Linux и прочих разновидностях UNIX. Конечно, производительность будет выше при работе с утилитами командной строки, использующими "родные" функции MySQL, но многие приложения Windows ориентируются на более универсальный интерфейс ODBC.

В некоторые версии пакета Microsoft Office входит программа Microsoft Query, которая позволяет взаимодействовать с любым источником данных ODBC (рис. 3.3). Это не самое удобное средство для ввода запросов, но и оно может оказаться полезным. Драйвер ODBC для программы MySQL доступен по адресу *www.mysql.com/downloads/api-myodbc.htm*

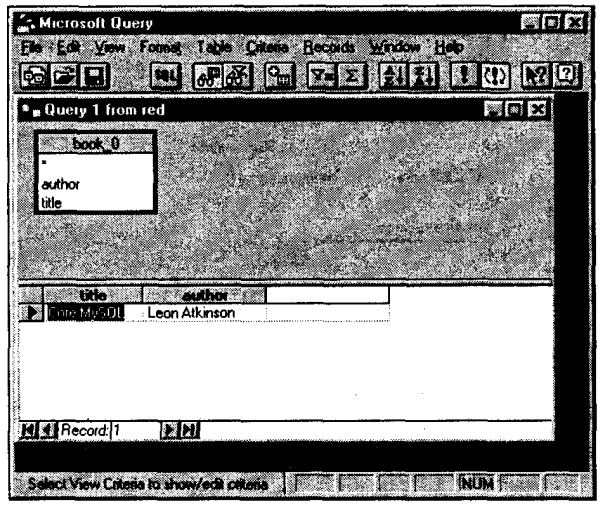


Рис. 3.3. Программа MS Query

Web-интерфейсы

К серверу MySQL можно обращаться и через **Web-броузер**. Существует несколько Web-клиентов, среди которых один из **лучших – phpMyAdmin** (рис. 3.4). Эта программа мне особенно нравится, так как она написана на **PHP**, самом популярном языке создания **Web-приложений**. Клиент phpMyAdmin доступен по адресу *http://phpwizard.net/projects/phpMyAdmin/ index.html*.

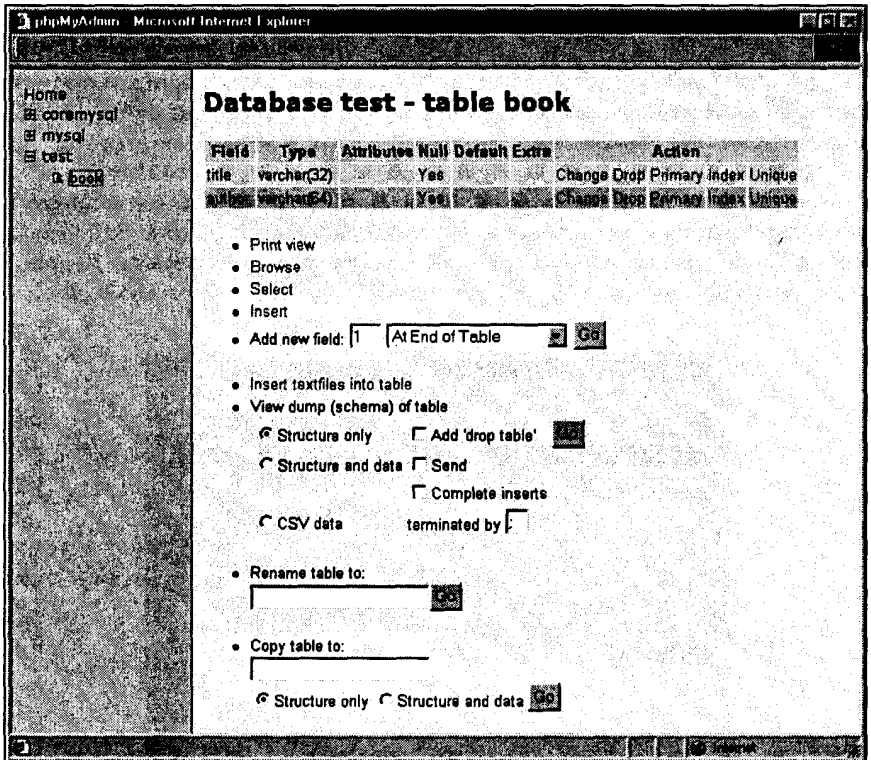


Рис. 3.4. Программа **phpMyAdmin**

КОНЦЕПЦИИ БАЗ ДАННЫХ

В этой главе…

Глава 4

В этой главе рассказывается об истории баз данных и их основных концепциях. Реляционные базы данных уже давно доминируют в своей области, но они представляют собой всего лишь одну из ступеней эволюционной лестницы. За ними стоит богатая история изобретений, и будущее сулит не менее разнообразные перспективы.

История

Базы данных — это одна из наиболее важных современных компьютерных технологий. Сегодня они во многом ассоциируются с банковскими транзакциями, хотя так было не всегда. История баз данных начинается с одного из самых значительных инженерных подвигов прошлого столетия: полета на Луну.

Североамериканская компания Rockwell заключила контракт с правительством США на участие в проекте Apollo. Построение космического корабля включает в себя сборку нескольких миллионов деталей, поэтому была создана система управления файлами, отслеживавшая информацию о каждой детали. Однако в ходе последующей проверки обнаружилась огромная избыточность. Выяснилось, что почти все данные повторяются в двух и более файлах.

Столкнувшись с задачей координации заказов на миллионы деталей, компания Rockwell в сотрудничестве с IBM в 1968 г. разработала автоматизированную систему заказов. Названная IMS (Information Management System — система управления информацией), она заложила основу концепции СУБД.

Ключевым новшеством IMS было разделение данных и функций деловой логики. Прикладные программисты получили возможность работать с информацией на логическом уровне, а база данных брала на себя задачу физического хранения. Подобное разделение труда привело к резкому скачку производительности.

Еще одним изобретением стал язык DL/I (Data Language/I). Это был специализированный язык составления нерегламентированных запросов к базе данных. Его по-

явление сделало ненужным дорогостоящее программирование на таких языках, как COBOL и FORTRAN, популярных в то время.

В СУБД IMS, применяемой до сих пор, реализована *иерархическая модель данных,* в которой существует один-единственный путь от корня иерархии к каждой записи. Такая модель стала основой для систем управления данными, она же дала толчок к последующим изобретениям из-за своей ограниченности. Полная история IMS была опубликована в 1998 г. (*www.research.ibm.com/journal/sj/374/blackman.html*).

В 1971 г. состоялась конференция по языкам обработки данных (Conference on Data Systems Languages, CODASYL), в задачу которой входила разработка стандартов баз данных. Ранее эта конференция уже стандартизировала язык COBOL. Новый стандарт был расширен на иерархическую модель данных, применяемую в IMS. Результатом стало появление *сетевой модели данных.*

В сетевой модели любая запись может участвовать в нескольких отношениях предок/потомок. Это позволяло обходить целый ряд ограничений иерархической модели. Разработкой сетевой модели занимался Чарльз Бейчман (Charles Bachman), в то время руководитель проекта IDS (Integrated Data System — интегрированная система обработки данных) в компании General Electric. Он же изобрел "диаграммы Бейчмана", описывающие сетевые базы данных. За свой труд в 1973 г. Бейчман получил награду Тьюринга.

Тем временем научный сотрудник компании IBM доктор Эдгар Кодд (Edgar Codd) работал над эпохальным документом для Ассоциации производителей вычислительной техники (Association for Computing Machinery, ACM). В июне 1970 г. этот документ был опубликован в *ACM Journal* под названием "Реляционная модель для больших банков совместно используемых данных" ("A Relational Model of Data for Large Shared Data Banks"). Этот документ в корне изменил теорию баз данных и принес доктору Кодду награду Тьюринга в 1981 году.

Доктор Кодд придумал реляционную модель, в которой данные можно было свободно описывать в их естественном виде без каких-либо ограничений, накладываемых средой физического хранения. Следовательно, это позволяло создать язык высокого уровня, способный работать с данными независимо от того, как они хранятся в компьютере.

В результате появились две СУБД: System R компании IBM и Ingres Калифорнийского университета в Беркли. В обеих был реализован реляционный модуль и язык запросов. Последний в СУБД System R первоначально назывался SEQUEL (Structured English Query Language — структурированный английский язык запросов). Позднее появилось название SQL (Structured Query Language). В 1986 г. организация ANSI опубликовала официальный стандарт языка SQL.

Терминология

СУБД управляет одной или несколькими базами данных. *База данных* представляет собой совокупность информации, организованной в виде множеств. Каждое множество содержит *записи* унифицированного вида. Сами записи состоят из *полей.* Обычно множества называют *таблицами,* а записи — строками таблиц.

Такова логическая модель данных. На жестком диске вся база данных может находиться в одном файле. В MySQL для каждой базы данных создается отдельный ката-

лог, а каждой таблице соответствуют три файла. В других СУБД могут использоваться иные принципы физического хранения данных.

Строки таблиц могут быть связаны друг с другом одним из трех способов. Простейшее отношение — "один к одному". В этом случае строка первой таблицы соответствует одной-единственной строке второй таблицы. На диаграммах такое отношение выражается записью 1:1.

Отношение "один ко многим" означает ситуацию, когда строка одной таблицы соответствует нескольким строкам другой таблицы. Это наиболее распространенный тип отношений. На диаграммах он выражается записью 1:N.

Наконец, при отношении "многие ко многим" строки первой таблицы могут быть связаны с произвольным числом строк во второй таблице. Такое отношение записывается как N:M.

СУБД

Программист, работающий с базой данных, не заботится о том, как эти данные хранятся, и приложения, взаимодействующие с СУБД, не знают о способе записи данных на диск. "Снаружи" виден лишь логический образ данных, и это позволяет менять код СУБД, не затрагивая код самих приложений.

Подобная обработка данных осуществляется посредством языка четвертого поколения (4GL), который поддерживает запросы, записываемые и исполняемые немедленно. Данные быстро утрачивают свою актуальность, поэтому скорость доступа к ним важна. Кроме того, программист должен иметь возможность формулировать новые запросы. Они называются *нерегламентированными* (ad hoc), поскольку не хранятся в самой базе данных и служат узкоспециализированным целям.

Язык четвертого поколения позволяет создавать *схемы*— точные определения данных и отношений между ними. Схема хранится как часть базы Данных и может быть изменена без ущерба для данных.

Схема предназначена для контроля целостности данных. Если, к примеру, объявлено, что поле содержит целочисленные значения, то СУБД откажется записывать в него числа с плавающей запятой или строки. Отношения между записями тоже четко контролируются, и несогласованные данные не допускаются. Операции можно группировать в транзакции, выполняемые по принципу "все или ничего".

СУБД обеспечивает безопасность данных. Пользователям предоставляются определенные права доступа к информации. Некоторым пользователям разрешено лишь просматривать данные, тогда как другие пользователи могут менять содержимое таблиц.

СУБД поддерживает параллельный доступ к базе данных. Приложения могут обращаться к базе данных одновременно, что повышает общую производительность системы. Кроме того, отдельные операции могут "распараллеливаться" для еще большего улучшения производительности.

Наконец, СУБД помогает восстанавливать информацию в случае непредвиденного сбоя, незаметно для пользователей создавая резервные копии данных. Все изменения, вносимые в базу данных, регистрируются, поэтому многие операции можно отменять и выполнять повторно.

Системы управления файлами

Простейшая база данных организована в виде набора обычных файлов. Эта модель напоминает картотечную организацию документов, при которой папки хранятся в ящиках, а в каждой папке подшито некоторое число страниц.

Системы управления файлами нельзя классифицировать как СУБД, так как обычно они являются частью операционной систем и ничего не знают о внутреннем содержимом файлов. Это знание заложено в прикладных программах, работающих с файлами. В качестве примера можно привести таблицу пользователей UNIX, хранящуюся в файле /etc/passwd. Программы, обращающиеся к этомуфайлу, знают, что в его первом поле находится имя пользователя, оканчивающееся двоеточием. Если приложению нужно отредактировать эту информацию, оно должно непосредственно открыть файл и позаботиться о правильном форматировании полей.

Такая модель базы данных очень неудобна, поскольку она требует использовать язык третьего поколения (3GL). В результате время программирования запросов увеличивается, а программист должен обладать более высокой квалификацией, так как ему нужно продумать не только логическую, но и физическую структуру хранения данных. Это приводит к тому, что между приложением и файлом образуется тесная связь. Вся информация о полях таблиц закодирована в приложении. Другое приложение, обращающееся к тому же файлу, вынуждено дублировать существующий код.

По мере увеличения числа приложений растет сложность управления базой данных. Изменения схемы данных приводят к необходимости изменения каждого программного компонента, для которого это актуально. Формирование новых запросов занимает столько времени, что зачастую теряет всякий смысл.

Системы управления файлами не могут помешать дублированию информации. Хуже того, не существует механизмов, предотвращающих несогласованность данных. Представьте себе файл, содержащий сведения обо всех служащих компании. В каждой строке есть поле, где записано имя начальника. Под руководством одного начальника работает много служащих, поэтому его имя будет неизбежно повторяться. Если где-то это имя будет записано неправильно, формально получится, что у служащего другой начальник. При замене начальника его имя придется "вылавливать" по всей базе данных.

Безопасность обычных файлов контролируется операционной системой. Отдельный файл может быть заблокирован для просмотра или модификации со стороны того или иного пользователя, но это выполняется только на уровне операционной системы. В конкретный момент времени лишь одно приложение может осуществлять запись в файл, что снижает общую производительность.

Иерархические базы данных

Иерархические базы данных поддерживают древовидную организацию информации. Связи между записями выражаются в виде отношений предок/потомок, а у каждой записи есть ровно одна родительская запись. Это помогает поддерживать ссылочную целостность. Когда запись удаляется из дерева, все ее потомки также должны быть удалены.

На рис. 4.1 изображена простая иерархическая база данных, в которой фиксируется деятельность независимого подрядчика. Корень дерева представляет собой запись о клиенте. Ее потомками являются две записи о счет-фактурах и три записи об оплатах счетов. Структура счета номер 17 уточняется в трех дочерних записях, у счета номер 23 одна такаязапись.

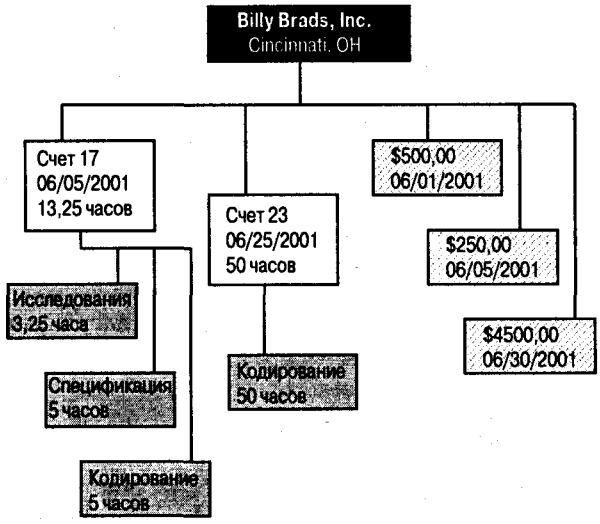


Рис. *4.1.* Иерархическая база данных

Иерархические базы данных имеют централизованную структуру, т.е. безопасность данных легко контролировать. К сожалению, определенные знания о физическом порядке хранения записей все же необходимы, так как отношения предок/потомок реализуются в виде физических указателей из одной записи на другую. Это означает, что поиск записи осуществляется методом прямого обхода дерева. Записи, расположенные в одной половине дерева, ищутся быстрее, чем в другой.

Отсюда следует необходимость правильно упорядочивать записи, чтобы время их поиска было минимальным. Это трудно, так как не все отношения, существующие в реальном мире, можно выразить в иерархической базе данных. Отношения "один ко многим" являются естественными, но практически невозможно описать отношения "многие ко многим" или ситуации, когда запись имеет несколько предков. До тех пор пока в приложениях будут кодироваться сведения о физической структуре данных, любые изменения этой структуры будут грозить перекомпиляцией.

Сетевые базы данных

Сетевая модель расширяет иерархическую модель, позволяя группировать связи между записями в множества. С логической точки зрения связь — это не сама запись. Связи лишь выражают отношения между записями. Как и в иерархической модели, связи ведут от родительской записи к дочерней, но на этот раз поддерживаетсямножественное наследование.

Следуя спецификации CODASYL, сетевая модель поддерживает DDL (Data Definition Language— язык определения данных) и DML (Data Manipulation Language — язык обработки данных). Это специальные языки, предназначенные для определения структуры базы данных и составления запросов. Несмотря на их наличие программист по-прежнему должен знать структуру базы данных.

В сетевой модели допускаются отношения "многие ко многим", а записи не зависят друг от друга. При удалении записи удаляются и все ее связи, но не сами связанные записи.

В сетевой модели требуется, чтобы связи устанавливались между существующими записями во избежание дублирования и искажения целостности. Данные можно изолировать в соответствующих таблицах и связать с записями в других таблицах.

Программисту не нужно заботиться о том, как организуется физическое хранение данных на диске. Это ослабляет зависимость приложений и данных. Но в сетевой модели требуется, чтобы программист помнил структуру данных при формировании запросов.

Оптимальную структуру базы данных сложно сформировать, а готовую структуру трудно менять. Если вид таблицы претерпевает изменения, все отношения с другими таблицами должны быть установлены заново, чтобы не нарушилась целостность данных. Сложность подобной задачи приводит к тому, что программисты зачастую отменяют некоторые ограничения целостности ради упрощения приложений.

Реляционные базы данных

В сравнении с рассмотренными выше моделями реляционная модель требует от СУБД гораздо более высокого уровня сложности. В ней делается попытка избавить программиста от выполнения рутинных операций по управлению данными, столь характерных для иерархической и сетевой моделей.

В реляционной модели база данных представляет собой централизованное хранилище таблиц, обеспечивающее безопасный одновременный доступ к информации со стороны многих пользователей. В строках таблиц часть полей содержит данные, относящиеся непосредственно к записи, а часть — ссылки на записи других таблиц. Таким образом, связи между записями являются неотъемлемым свойством реляционной модели.

Каждая запись таблицы имеет одинаковую структуру. Например, в таблице, содержащей описания автомобилей, у всех записей будет один и тот же набор полей: производитель, модель, год выпуска, пробег и т.д. Такие таблицы легко изображать в графическом виде.

В реляционной модели достигается информационная и структурная независимость. Записи не связаны между собой настолько, чтобы изменение одной из них затронуло остальные, а изменение структуры базы данных не обязательно приводит к перекомпиляции работающих с ней приложений.

В реляционных СУБД применяется язык SQL, позволяющий формулировать произвольные, нерегламентированные запросы. Это язык четвертого поколения, поэтому любой пользователь может быстро научиться составлять запросы. К тому же, существует множество приложений, позволяющих строить логические схемы запросов в графическом виде. Все это происходит за счет ужесточения требований к производительности компьютеров. К счастью, современные вычислительные мощности более чем адекватны.

Реляционные базы данных страдают от различий в реализации языка SQL, хотя это и не проблема реляционной модели. Каждая реляционная СУБД реализует какое-то подмножество стандарта SQL плюс набор уникальных команд, что усложняет задачу программистам, пытающимся перейти от одной СУБД к другой. Приходится делать нелегкий выбор между максимальной переносимостью и максимальной производительностью. В первом случае нужно придерживаться минимального общего набора команд, поддерживаемых в каждой СУБД. Во втором случае программист просто сосредоточивается на работе в данной конкретной СУБД, используя преимущества ее уникальных команд и функций.

MySQL — это реляционная СУБД, и настоящая книга посвящена изучению именно реляционной модели. Но теория баз данных не стоит на месте. Появляются новые технологии, которые расширяют реляционную модель.

Объектно-ориентированные базы данных

Объектно-ориентированная база данных (ООБД) позволяет программистам, которые работают с языками третьего поколения, интерпретировать все свои информационные сущности как объекты, хранящиеся в оперативной памяти. Дополнительный интерфейсный уровень абстракции обеспечивает перехват запросов, обращающихся к тем частям базы данных, которые находятся в постоянном хранилище на диске. Изменения, вносимые в объекты, оптимальным образом переносятся из памяти на диск.

Преимуществом ООБД является упрощенный код. Приложения получают возможность интерпретировать данные в контексте того языка программирования, на котором они написаны. Реляционная база данных возвращает значения всех полей в текстовом виде, а затем они приводятся к локальным типам данных. В ООБД этот этап ликвидирован. Методы манипулирования данными всегда остаются одинаковыми независимо от того, находятся данные на диске или в памяти.

Данные в ООБД способны принять вид любой структуры, которую можно выразить на используемом языке программирования. Отношения между сущностями также могут быть произвольно сложными. ООБД управляет кэш-буфером объектов, перемещая объекты между буфером и дисковым хранилищем по мере необходимости.

С помощью ООБД решаются две проблемы. Во-первых, сложные информационные структуры выражаются в них лучше, чем в реляционных базах данных, а во-вторых, устраняется необходимость транслировать данные из того формата, который поддерживается в СУБД. Например, в реляционной СУБД размерность целых чисел может составлять 11 цифр, а в используемом языке программирования — 16. Программисту придется учитывать эту ситуацию.

Объектно-ориентированные СУБД выполняют много дополнительных функций. Это окупается сполна, если отношения между данными очень сложны. В таком случае производительность ООБД оказывается выше, чем у реляционных СУБД. Если же данные менее сложны, дополнительные функции оказываются избыточными.

В объектной модели данных поддерживаются нерегламентированные запросы, но языком их составления не обязательно является SQL. Логическое представление данных может не соответствовать реляционной модели, поэтому применение языка SQL станет бессмысленным. Зачастую удобнее обрабатывать объекты в памяти, выполняя соответствующие виды поиска.

Большим недостатком объектно-ориентированных базданных является их тесная связь с применяемым языком программирования. К данным, хранящимся в реляционной СУБД, могут обращаться любые приложения, тогда как, к примеру, Java-объект, помещенный в ООБД, будет представлять интерес лишь для приложений, написанных на Java.

Объектно-реляционные базы данных

Объектно-реляционные СУБД объединяют в себе черты реляционной и объектной моделей. Их возникновение объясняется тем, что реляционные базы данных хорошо работают со встроенными типами данных и гораздо хуже — с пользовательскими, нестандартными. Когда появляется новый важный тип данных, приходится либо включать его поддержку в СУБД, либо заставлять программиста самостоятельно управлять данными в приложении.

Не всякую информацию имеет смысл интерпретировать в виде цепочек символов или цифр. Представим себе музыкальную базу данных. Песню, закодированную в виде аудиофайла, можно поместить в текстовое поле большого размера, но как в таком случае будет осуществляться текстовый поиск?

Перестройка СУБД с целью включения в нее поддержки нового типа данных — не лучший выход из положения. Вместо этого объектно-реляционная СУБД позволяет загружать код, предназначенный для обработки "нетипичных" данных. Таким образом, база данных сохраняет свою табличную структуру, но способ обработки некоторых полей таблиц определяется извне, т.е. программистом.

РЕЛЯЦИОННАЯ МОДЕЛЬ

В этой главе…

Реляционная алгебра
Таблицы, строки и столбцы
Ключи
Отношения
Реляционные операции
Является ли MySQL настоящей реляционной СУБД

Глава 5

В этой главе будет подробно проанализирована реляционная модель. Мы рассмотрим специфическую терминологию данной модели, а также поговорим о требованиях, предъявляемых к настоящей реляционной СУБД.

Читая представленный ниже материал, не забывайте о том, что теория не всегда идет рука об руку с практикой. Реляционная теория налагает ряд ограничений, которые оказываются непрактичными в реальных приложениях. Реляционная модель пытается максимально избавить программиста от заботы о физической реализации базы данных. Но на практике трудно игнорировать информацию о том, как СУБД использует жесткие диски и системную память.

Реляционная алгебра

Доктор Кодд первоначально описал реляционную модель как область применения реляционной алгебры. Для тех, кто подзабыл курс высшей математики, напомним, что *алгеброй* называется система определения множеств и операций над ними. Под *множеством* понимается совокупность уникальных элементов, объединенных по какому-то признаку. *Оператор*— это символическая запись правила преобразования, выполняемого над одним или несколькими элементами множества. В традиционной алгебре элементами множеств являются числа, над которыми выполняются такие операции, как сложение, вычитание, умножение, деление и др. Реляционная алгебра — это система манипулирования отношениями, которые являются элементами, группируемыми во множество.

Таблицы, строки и столбцы

Реляционная модель скрывает детали физического хранения данных. Вся работа ведется на логическом уровне. Так легче выявлять отношения, существующие между элементами данных.

Реляционная база данных состоит из таблиц. Можно провести аналогию между базой данных и ящиком картотеки. Таблица в этом случае будет папкой, лежащей в ящике. В MySQL таблице соответствуют три файла на диске, но это имеет значение только тогда, когда приходится заниматься администрированием системы на низком уровне. Доктор Кодд подчеркивал важность освобождения пользователей базы данных от знания особенностей ее физической реализации.

Таблица состоит из строк (записей) и столбцов (полей). В столбце хранятся соответствующие значения каждой строки; каких либо "пропусков" или коротких столбцов быть не может. Запись является отдельной сущностью, а поля представляют собой атрибуты записей.

Доктор Кодд называл таблицы *отношениями,* поскольку каждая строка таблицы неявно связана со всеми остальными строками, разделяя с ними общую форму записи. Собственно запись называется *кортежем,* т.е. набором взаимосвязанных атрибутов. Рассмотрим таблицу, состоящую из четырех строк и трех столбцов (табл. 5.1). Все записи этой таблицы связаны друг с другом, так как они описывают бейсболистов.

Фамилия	***Дата рождения***	***Позиция***
Tejada	1976-05-25	Шорт-стоп
Giambi	1971-01-08	Первая база
Hudson	1975-07-14	Питчер
Seanz	1970-10-08	Лучший отбивающий

У каждого столбца есть название и тип. Типы данных будутрассматриваться в главе 11, "Типы столбцов и индексов", а пока лишь скажем, что базовыми типами считаются строковый, числовой и дата/время. В табл. 5.1 столбцы "Фамилия" и "Позиция" имеют строковый тип, а в столбце "Дата рождения" находятся значения даты.

Определение атрибута является строгим. Все значения атрибута должны иметь один и тот же тип. Строковым атрибутам дополнительно назначается максимальная длина (в табл. 5.1 это не показано). Строки, длина которых превышает заданный предел, будут усекаться.

Порядок строк в таблице произволен и не имеет никакого значения. По сути, он отражает очередность записи строк на диск и может быть разным. Как будет показано в главе 6, "Язык SQL", есть способы указания порядка записей, возвращаемых в результате запроса.

Таблицы не содержат дубликатов. Каждая запись уникальным образом идентифицируется совокупностью значений своих полей. На практике большинство СУБД допускает создание таблиц без ключевых столбцов, но это не такая большая проблема, как может показаться.

В табл. 5.1 строки можно идентифицировать по столбцу "Фамилия". Но что будет, если команда включит в свой состав игрока-однофамильца? Придется взять за первичный ключ комбинацию фамилии и даты рождения. О концепции ключей рассказывается в следующем разделе.

Предположим, в команде появляется новый игрок, позиция которого на поле еще не определена. В этом случае в поле "Позиция" следует занести значение NULL. Это специальная константа, не равная пустой строке. Она означает отсутствие значения.

Ключи

Связи между записями реализуются в виде *ключей*. Ключ — это определенным образом помеченный столбец таблицы. Для того чтобы его значения были связаны со столбцом другой таблицы, необходимо установить между столбцами виртуальную связь. Это делается не на табличном уровне: информация о связях хранится отдельно от данных.

Давайте расширим таблицу бейсболистов, добавив в нее столбец "Команда" (табл. 5.2). В этом столбце хранится идентификатор команды, а не ее название. Соответствия между идентификаторами и названиями находятся в другой таблице (табл. 5.3).

Таблица 5.2. Таблица бейсболистов

Фамилия	*Дата рождения*	*Позиция*	*Команда*
Tejada	1976-05-25	Шорт-стоп	1
Giambi	1971-01-08	Первая база	1
Hudson	1975-07-14	Питчер	1
Seanz	1970-10-08	Лучший отбивающий	1
Bonds	1964-07-24	Внешнее поле	2
Giambi	1974-09-30	Внешнее поле	1
Snow	1968-02-26	Первая база	2

Столбец "Идентификатор" в табл. 5.3 является *первичным ключом*. Значение первичного ключа уникальным образом идентифицирует каждую строку. Только значения первичного ключа могут встречаться в столбце "Команда" табл. 5.2. Последний, в свою очередь, называется *внешним ключом,* так как его значения берутся из внешней таблицы.

Идентификатор	*Название*
1	Oakland Athletics
2	San Francisco Giants

Повторять названия команд в первой таблице нежелательно, поскольку это прямой путь к нарушению целостности. Ничто не мешает пользователю добавить строку, в которой имя команды будет записано неправильно. Как следствие, в последующих запросах на выборку будет отображаться неверное число команд.

В большинстве реляционных СУБД есть средства контроля внешних столбцов, позволяющие гарантировать занесение в них "правильных" значений. В MySQL 3.23.39 таких средств еще нет. Столбец можно назначить внешним ключом, но MySQL не будет проверять, совпадают ли его значения со значениями указанного первичного ключа. Подобная задача возлагается на приложения.

Если СУБД обеспечивает проверку внешних ключей, то она не позволит создавать записи с неправильными значениями внешнего ключа. Возможно также каскадное удаление записей. Например, удаление строки из таблицы может привести к удалению всех записей других таблиц, в которых внешний ключ равен ее первичному ключу. Планируется, что такая функция в скором времени появится и в MySQL.

Пока что речь шла только о первичных и внешних ключах. Но помимо них есть еще несколько типов ключей.

Суперключ— это совокупность атрибутов, уникальным образом идентифицирующих каждую запись. Например, в табл. 5.2 в качестве суперключа можно использовать объединение всех атрибутов или же, к примеру, столбцов "Фамилия" и "Дата рождения". А вот сочетание столбцов "Фамилия" и "Команда" не подойдет, так как в команде "Oakland Athletics" есть два игрока-однофамильца.

Ключ-кандидат— это минимальный суперключ. Например, ключ, объединяющий столбцы "Фамилия", "Дата рождения" и "Команда", не является кандидатом, поскольку первых двух столбцов достаточно, чтобы идентифицировать каждую запись.

Таким образом, первичный ключ представляет собой ключ-кандидат, выбранный для идентификации записей таблицы. У каждой таблицы есть концептуальный набор суперключей. Их подмножеством являются ключи-кандидаты, и только один из кандидатов может стать первичным ключом. Реляционная модель не допускает, чтобы какой-либо атрибут первичного ключа был пустым, поэтому в нашем случае наилучший первичный ключ — столбцы "Фамилия" и "Дата рождения". Игрок может в данный момент не принадлежать никакой команде, но всегда известны его фамилия и дата рождения. Предположим для простоты, что в лиге нет игроков, родившихся в один день и носящих одинаковую фамилию.

Первичный ключ является важным средством обеспечения целостности данных. MySQL не допустит, чтобы две записи имели одинаковые значения в столбцах первичного ключа. Предположим, один из игроков меняет команду. Если попытаться вставить в таблицу новую строку, которая будет отличаться от существующей лишь значением поля "Команда", то MySQL выдаст сообщение об ошибке. Нужно либо изменить существующую строку, либо предварительно удалить ее, а затем вставить в обновленном виде.

Когда столбец или группа столбцов помечается как ключ, дополнительно создается *индекс*. Индекс хранит список значений ключа и указатели на записи, содержащие эти значения. Наличие индекса позволяет СУБД быстро находить нужные записи. В целом индексы ускоряют выполнение операций чтения, но замедляют выполнение операций записи. Это объясняется тем, что при добавлении строк в таблицу или обновлении ключевых столбцов необходимо обновлять индекс.

В реляционной модели все ключи, кроме первичного, считаются вторичными. Они используются для ускорения запросов. Внешний ключ является вторичным ключом, который соответствует первичному ключу другой таблицы.

Отношения

Между таблицами могут существовать отношения трех типов: "один ко многим", "один к одному" и "многие ко многим". Только первый из них необходим для реляционной модели. Для реализации двух других типов требуются определенные табличные преобразования.

Отношение *"один ко многим"* (1:N) является естественным типом отношений в реляционной базе данных. Оно реализуется с помощью внешних ключей, рассмотренных выше. При отношении 1:N любой строке первой таблицы может соответствовать несколько записей второй таблицы. Если проанализировать связь в противоположном отношении, то окажется, что строке второй таблицы соответствует всего одна запись первой таблицы.

В идеально спроектированной реляционной базе данных отношение *"один к одному"* (1:1) не нужно. Если каждой строке одной таблицы соответствует одна строка другой таблицы, то это обычно свидетельствует о том, что обе таблицы нужно объединить в единое целое. Исключение из правила— необычный случай, когда число столбцов таблицы превышает предел, установленный в СУБД. В MySQL этот предел равен 3000, так что маловероятно, чтобы кому-то пришло в голову его превысить. Есть СУБД, где предельное число столбцов гораздо меньше, например 250, но даже этого числа вполне достаточно для большинства приложений. В будущих версиях MySQL жесткий предел будет вообще снят, и создавать столбцы можно будет до тех пор, пока не закончится место на диске.

Еще одна причина существования отношения "один к одному" —это случай, когда определенный набор атрибутов применим лишь к небольшому подмножеству записей. Например, в таблице может храниться статистика по каждому из игроков, но для питчеров (подающих) статистические показатели будут несколько отличаться, поэтому для них лучше создать отдельную таблицу.

В реляционной базе данных нельзя напрямую создать отношение *"многие ко многим"* (M:N). Его необходимо преобразовать в два отношения 1:N, устанавливаемых с промежуточной таблицей. Например, бейсболист, особенно игрок внешнего поля ("аутфилд"), может занимать на поле более одной позиции. Если информацию обо всех занимаемых позициях хранить в общей таблице, то получится, что есть группа игроков с несколькими позициями и есть позиции, занимаемые несколькими игроками.

Выход из положения заключается в декомпозиции, т.е. разбивке отношения M:N на два отношения 1:N. Это означает, что ссылки между двумя таблицами будут вынесены в третью таблицу, содержащую всего два столбца. В них будут сопоставляться первичные ключи основных таблиц.

На рис. 5.1 изображена схема распределения позиций между игроками. Игроку 6 соответствуют три позиции на внешнем поле, а также позиция на первой базе. Промежуточная таблица (в центре) связывает строки таблицы игроков со строками таблицы позиций. Точно так же можно было бы изобразить и обратную связь, например показать список игроков, способных играть на первой базе.

Отношение 1:N, соединяющее столбцы одной и той же таблицы, называется *самообъединением*. Оно используется для отображения иерархических структур. Подразумевается, что внешний ключ ссылается на родительскую строку собственной таблицы.

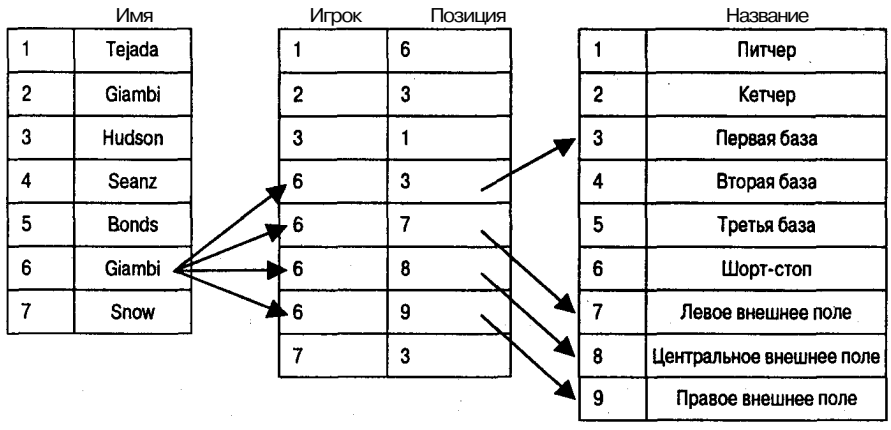


Рис. 5.1. Формирование отношения M:N посредством промежуточной таблицы

Реляционные операции

Как уже упоминалось, реляционная модель основана на реляционной алгебре доктора Кодда. В свою очередь, за реляционной алгеброй стоит теория множеств, в которой определен целый ряд операций над множествами. Как писал сам доктор Кодд, для реляционной модели представляют интерес только те операции, результатом которых являются множества. Отсюда имеем восемь простейших операций: выборка, проекция, пересечение, сложение, вычитание, умножение, деление и переименование. На их основе строятся более сложные операции, называемые объединениями. В языке SQL большинство перечисленных операций реализуется с помощью инструкции SELECT.

Результатом операции над таблицей будет временная таблица. Она не сохраняется в базе данных, но в течение некоторого времени к ней можно обращаться.

Операция *выборки* выполняется над одной таблицей. Результирующая таблица будет содержать все исходные столбцы, но, возможно, не все строки. С логической точки зрения это фильтрация строк. Предположим, нам нужно получить список бейсболистов, родившихся до 1-го января 1975 г. Результат подобной выборки показан в табл. 5.4.

Фамилия	*Дата рождения*	*Позиция*	*Команда*
Giambi	1971-01-08	Первая база	1
Seanz	1970-10-08	Лучший отбивающий	1
Bonds	1964-07-24	Внешнее поле	2
Giambi	1974-09-30	Внешнее поле	1
Snow	1968-02-26	Первая база	2

Операция *проекции* возвращает все записи исходной таблицы, но, возможно, не все столбцы. Это, по сути, фильтрация столбцов. Предположим, требуется получить список фамилий бейсболистов. На рис. 5.2 изображена исходная таблица и результат ее проекции. Результирующая таблица содержит шесть фамилий, а не семь, так как дубликаты недопустимы.

Имя	День рождения	Позиция	Команда
Tejada	1976-05-25	Шорт-стоп	1
Giambi	1971-01-08	Первая база	1
Hudson	1975-07-14	Питчер	1
Seanz	1970-10-08	Лучший отбивающий	1
Bonds	1964-07-24	Внешнее поле	2
Giambi	1974-09-30	Внешнее поле	1
Snow	1968-02-26	Первая база	2

Проекция по столбцу "Имя"

Имя
Tejada
Giambi
Hudson
Seanz
Bonds
Snow

Рис. 5.2. Проекция фамилий игроков

Операция *пересечения* выполняется над двумя таблицами идентичной структуры. В результате возвращаются только те записи, которые встречаются в обеих исходных таблицах. Предположим, нам необходимо узнать, какие игроки принимали участие в матче "Всех звезд" как в 1999, так и в 2000 году. Результат соответствующей операции пересечения представлен на рис. 5.3.

Матч "Всех звезд" 1999 г.
Cal Ripken
Harold Baines
Ivan Rodriguez
Jim Thome
Ken Griffey Jr.
Kenny Lofton
Manny Ramirez
Nomar Garciaparra
Roberto Alomar

∩

Матч "Всех звезд" 2000 г.
Bernie Williams
Carl Everett
David Wells
Derek Jeter
Ivan Rodriguez
Jason Giambi
Jermaine Dye
Roberto Alomar
Travis Fryman

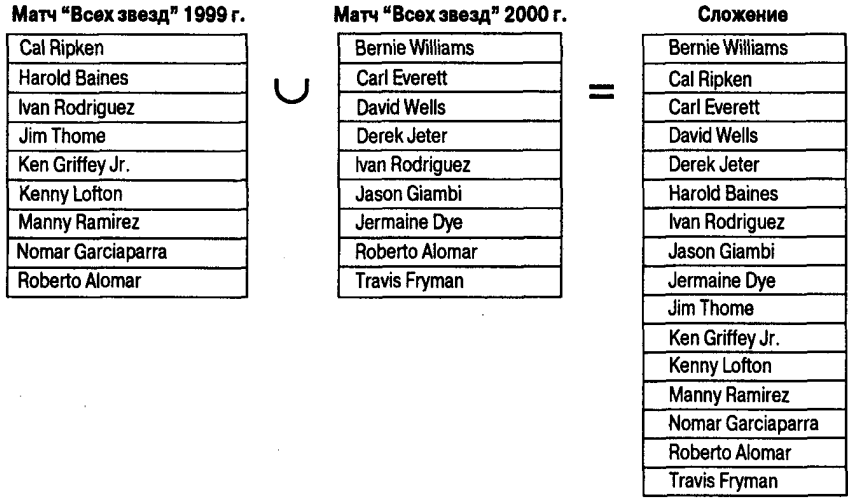
=

Пересечение
Ivan Rodriguez
Roberto Alomar

Рис. 5.3. Операция пересечения

Операция *сложения* также выполняется над двумя таблицами идентичной структуры. При этом в результирующую таблицу попадают все записи исходных таблиц. Например, с помощью данной операции можно получить объединенный список участников матчей "Всех звезд" в 1999 и 2000 году (рис. 5.4). Результирующая таблица будет содержать 16 строк, а не 18, поскольку два игрока принимали участие в обоих матчах и их имена повторяются.

Операция *вычитания* возвращает строки первой таблицы, отсутствующие во второй таблице. Эта операция не является обратной по отношению к операции пересечения. С ее помощью можно определить, к примеру, кто из участников матча "Всех звезд" в 1999 г. не попал на аналогичный матч в 2000 г. (рис. 5.5).



Матч "Всех звезд" 1999 г.
Cal Ripken
Harold Baines
Ivan Rodriguez
Jim Thome
Ken Griffey Jr.
Kenny Lofton
Manny Ramirez
Nomar Garciaparra
Roberto Alomar

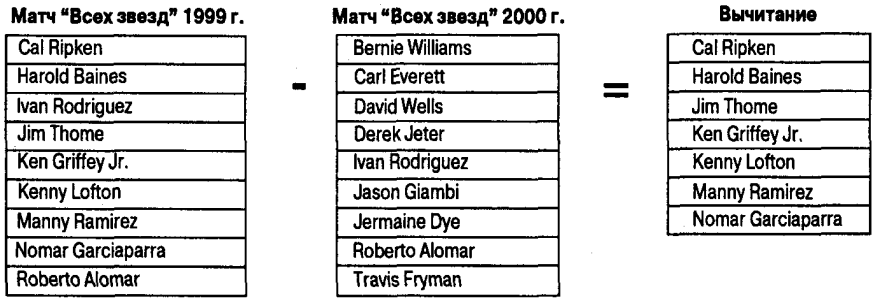
∪

Матч "Всех звезд" 2000 г.
Bernie Williams
Carl Everett
David Wells
Derek Jeter
Ivan Rodriguez
Jason Giambi
Jermaine Dye
Roberto Alomar
Travis Fryman

=

Сложение
Bernie Williams
Cal Ripken
Carl Everett
David Wells
Derek Jeter
Harold Baines
Ivan Rodriguez
Jason Giambi
Jermaine Dye
Jim Thome
Ken Griffey Jr.
Kenny Lofton
Manny Ramirez
Nomar Garciaparra
Roberto Alomar
Travis Fryman

Рис. 5.4. Операция сложения



Матч "Всех звезд" 1999 г.
Cal Ripken
Harold Baines
Ivan Rodriguez
Jim Thome
Ken Griffey Jr.
Kenny Lofton
Manny Ramirez
Nomar Garciaparra
Roberto Alomar

-

Матч "Всех звезд" 2000 г.
Bernie Williams
Carl Everett
David Wells
Derek Jeter
Ivan Rodriguez
Jason Giambi
Jermaine Dye
Roberto Alomar
Travis Fryman

=

Вычитание
Cal Ripken
Harold Baines
Jim Thome
Ken Griffey Jr.
Kenny Lofton
Manny Ramirez
Nomar Garciaparra

Рис. 5.5. Операция вычитания

Операция *умножения* объединяет каждую строку первой таблицы с каждой строкой второй таблицы. Эта операцию еще называют *декартовым произведением*. Количество строк результирующей таблицы равно произведению числа строк исходных таблиц. Столбцами результирующей таблицы являются все столбцы первой таблицы, за которыми следуют все столбцы второй таблицы.

Вернемся опять к бейсболу. После подачи мяч может оказаться у питчера (подающего) или кетчера (принимающего). Им необходимо бросить мяч на одну из трех баз, чтобы ее не "украли" соперники. На рис. 5.6 изображена схема табличного произведения, сопоставляющего питчера и кетчера с игроками на базах. Есть два игрока, способных бросить мяч, и три игрока, которые могут его поймать. В результирующей таблице будут два столбца "Идентификатор" и два столбца "Игрок". В реляционной модели это допустимо. СУБД понимает, какие столбцы к какой исходной таблице относятся.

ID	Игрок
1	Hudson
2	Hernandez

×

ID	Игрок	База
1	Giambi	Первая база
2	Tejada	Вторая база
3	Chavez	Третья база

=

ID	Игрок	ID	Игрок	База
1	Hudson	1	Giambi	Первая база
1	Hudson	2	Tejada	Вторая база
1	Hudson	3	Chavez	Третья база
2	Hernandez	1	Giambi	Первая база
2	Hernandez	2	Tejada	Вторая база
2	Hernandez	3	Chavez	Третья база

Рис. 5.6. Операция умножения

MySQL, как и большинство других СУБД, не разрешает, чтобы в одной таблице существовали два столбца с одинаковыми названиями. Но таблица результатов запроса— это другое дело. Незаметно для пользователя СУБД помечает результирующие столбцы именами исходных таблиц, поэтому совпадение имен в нашем случае — кажущееся. В главе 6, "Язык SQL", будет рассказано о том, как можно явно задавать имена столбцов в подобных ситуациях.

Операция *деления* выполняется над двумя таблицами, первая из которых состоит из двух столбцов, а вторая — из одного. Значения второй таблицы сравниваются со значениями первого столбца первой таблицы, и если обнаруживаются совпадения, то соответствующие значения второго столбца первой таблицы включаются в результаты запроса (по сути, происходит фильтрация этого столбца). Таким образом, результирующая таблица состоит из одного столбца.

Пример операции деления представлен на рис. 5.7. В первой таблице приведен список игроков и позиций, которые они могут занимать на поле. Во второй таблице указываются игроки, чьи позиции необходимо определить. Результирующая таблица содержит две записи (еще одна запись-дубликат была удалена).

Игрок	Позиция
Tejada	Вторая база
Tejada	Шорт-стоп
Seanz	Лучший отбивающий
Seanz	Первая база
Giambi	Первая база

/

Игрок
Giambi
Seanz

=

Позиция
Первая база
Лучший отбивающий

Рис. 5.7. Операция деления

Операция *переименования* назначает столбцу другое имя. Это тривиальная операция, поэтому зачастую о ней не упоминают при обсуждении реляционной алгебры. Но именно с ее помощью решается упомянутая выше проблема с именами столбцов в таблице произведения. Обычно для ясности двум одноименным столбцам назначаются разные имена, выбираемые произвольно. Такие имена называются *псевдонимами*.

Сами по себе все вышеперечисленные реляционные операции не слишком полезны. Настоящая сила заключена в их комбинациях. Например, выполняя одновременно операции выборки и проекции, можно добиться того, что в результате запроса будет получена одна ячейка таблицы.

Одним из классов более сложных операций являются *объединения.* Объединение — это частный случай произведения, при котором к таблице произведения применяются дополнительные критерии фильтрации *(условия отбора).*

В операции *внутреннего объединения* сравниваются связанные столбцы двух таблиц (столбцы, упомянутые в условии отбора). Строки, которые не удовлетворяют условию сравнения связанных столбцов, удаляются из результирующей таблицы. Обычно сравниваемые столбцы называются одинаково. Такое объединение называется *естественным.*

Рассмотрим внутреннее объединение, схема которого представлена на рис. 5.8. Объединение выполняется по столбцу "Номер команды", присутствующему в обеих таблицах и включаемому в результаты запроса только один раз. Обратите внимание на третью строку правой таблицы. Для нее нет эквивалента в левой таблице, поэтому она не попадет в результаты запроса.

Игрок	Позиция	Номер команды
Tejada	Шорт-стоп	1
Giambi	Первая база	1
Hudson	Питчер	1
Bonds	Внешнее поле	2
Snow	Первая база	2

Внутреннее объединение

Номер команды	Название команды
1	Oakland Athletics
2	San Francisco Giants
3	Colorado Rockies

по столбцу "Номер команды"

Игрок	Позиция	Номер команды	Название команды
Tejada	Шорт-стоп	1	Oakland Athletics
Giambi	Первая база	1	Oakland Athletics
Hudson	Питчер	1	Oakland Athletics
Bonds	Внешнее поле	2	San Francisco Giants
Snow	Первая база	2	San Francisco Giants

Рис. 5.8. Внутреннее объединение

Внешнее объединение является расширением внутреннего. Существуют три типа внешних объединений: левое, правое и полное. В *левое внешнее объединение* включаются все строки первой таблицы, в *правое внешнее объединение*— все строки второй таблицы, а в *полное внешнее объединение*— все строки обеих исходных таблиц. Но внешнее объединение — это не то жесамое, что и операция произведения. Строки, которым не находится соответствия в левой или правой таблице, дополняются значениями NULL.

В таблицах левого внешнего объединения на рис. 5.9 содержатся имена бейсболистов и соответствующие им прозвища. Не у всех игроков есть клички, но нам нужно получить полный список игроков, отсюда очевиден выбор типа объединения. В результаты запроса попадают все четыре строки первой таблицы. У игрока под номером 3 нет официального прозвища, поэтому в его столбец помещается значение NULL.

Номер игрока	Имя игрока
1	Hank Aaron
2	Joe DiMaggio
3	Jose Canseco
4	Reggie Jackson

Левое внешнее объединение

Номер игрока	Прозвище
1	Hammerin' Hank
2	Joltin' Joe
4	Mister October

=

Номер игрока	Имя игрока	Прозвище
1	Hank Aaron	Hammerin' Hank
2	Joe DiMaggio	Joltin' Joe
3	Jose Canseco	[null]
4	Reggie Jackson	Mister October

Рис. 5.9. Левое внешнее объединение

Правое внешнее объединение строится аналогично левому, только по отношению к строкам правой таблицы. Некоторые СУБД не поддерживают данный тип объединения, поскольку его всегда можно преобразовать в левое внешнее объединение. Тем не менее в MySQL такая поддержка имеется.

Полное внешнее объединение реализуется так, как если бы правое и левое объединения выполнялись одновременно. Поддержка данного типа объединений отсутствует в MySQL версии 3.23, но должна появиться в версии 4.1.

Является ли MySQL настоящей реляционной СУБД

Ни одна реляционная СУБД не реализует реляционную модель целиком. Разработчики часто руководствуются принципом *разумной достаточности:* гораздо меньше усилий затрачивается на разработку минимального решения, чем полной системы, обладающей интересными, но зачастую совершенно ненужными возможностями. Системы, создаваемые по принципу разумной достаточности, проще в использовании и потребляют меньше системных ресурсов.

Чтобы быть полностью реляционной, СУБД должна поддерживать все реляционные операции и обеспечивать целостность первичных и внешних ключей. Первичные ключи не могут содержать значения NULL, а внешний ключ должен либо быть равен NULL, либо содержать указатель на существующее значение первичного ключа другой таблицы.

MySQL 3.23 не выполняет проверку внешних ключей. Не поддерживаются также некоторые реляционные операции, например полное внешнее объединение и деление. Тем не менее MySQL является настоящей реляционной СУБД.

ЯЗЫК SQL

В этой главе…

SQL — язык четвертого поколения
Определение данных
Вставка записей
Обновление записей
Удаление записей
Запросы
Объединения
Упорядочение результатов запроса
Группировка результатов запроса
Ограничение числа возвращаемых записей
Изменение определения таблицы

Глава 6

SQL— это язык взаимодействия с базами данных, применяемый в большинстве реляционных СУБД, включая MySQL. Основной стандарт языка был принят в 1992 г. и называется SQL2 или SQL-92. Продолжается также работа над наиболее современным стандартом SQL3.

В MySQL команды SQL вводятся с помощью одного из клиентов, описанных в главе 3, "Взаимодействие с MySQL". В настоящей же главе будут рассмотрены основные возможности языка. Подробное знакомство со всеми поддерживаемыми инструкциями SQL произойдет в главе 13, "Инструкции SQL".

SQL — язык четвертого поколения

SQL больше напоминает человеческий язык, чем C, PHP или Java, так как это язык четвертого поколения. К языкам первого поколения относятся платформно-зависимые машинные коды, напрямую воспринимаемые центральным процессором. Второе поколение — это ассемблерные языки. Языки третьего поколения считаются высокоуровневыми, и на них работают большинство программистов.

Одна из задач SQL — быть понятным непрограммистам. Поэтому запросы читаются как обычные предложения. Словарь SQL относительно невелик, а его команды являются словами английского языка. Таким образом, SQL несложно изучить.

Обычно ключевые слова SQL записывают прописными буквами, чтобы отличать их от названий таблиц и столбцов. Но для MySQL это не имеет никакого значения.

Часто для удобства восприятия инструкции SQL записывают в нескольких строках, что допускается синтаксическим анализатором. В программе mysql конец инструкции помечается специальной командой. Проще всего ввести точку с запятой (;), но можно также набрать \д или до. Все эти команды не являются частью языка, они необходимы лишь интерпретатору mysql.

Определение данных

Для определения структуры базы данных предназначена команда CREATE. Если база данных еще не была создана посредством утилиты mysqladmin, то это можно сделать с помощью инструкции CREATE DATABASE. Единственным ее аргументом является имя базы данных (она создается пустой). Естественно, если база с таким именем уже существует, будет выдано сообщение об ошибке. В листинге 6.1 демонстрируется создание базы store.

```
CREATE DATABASE store
```

Инструкция DROP DATABASE удаляет базу данных вместе со всеми таблицами. Это необратимое действие, так что будьте осторожны.

Таблица создается с помощью инструкции CREATE TABLE. Ей нужно указать не только имя таблицы, но и ее полное определение, состоящее из определений отдельных полей.

Имя таблицы может быть полным либо неполным. К полному имени добавляется имя базы данных, например store.item. Точно так же полное имя столбца включает в себя имена базы данных и таблицы: store.item.price. Но гораздо удобнее назначить стандартную базу данных, тогда отпадет необходимость в полных именах. Для этого предназначена инструкция USE. Имя базы данных можно задать и при вызове программы mysql. Подробнее об этом рассказывается в главе 14, "Утилиты командной строки".

Общий формат инструкции CREATE TABLE таков:

```
CREATE TABLE имя_таблицы
    (определение столбца, ...)
```

Определение столбца включает в себя имя столбца и спецификацию его типа. Таблица, создаваемая в листинге 6.2, содержит четыре столбца: ID, Name, Price и Description.

```
CREATE TABLE item (
     ID INT(6) NOT NULL AUTO_INCREMENT,
     Name CHAR(32) NOT NULL,
     Price DECIMAL(4,2) NOT NULL,
     Description CHAR(255) DEFAULT 'No Description',

     PRIMARY KEY(ID),
     KEY (Name)
```

Определение существующей таблицы можно узнать с помощью инструкции DESCRIBE (сокращенный вариант— DESC). В листинге 6.3 показан результат, выдаваемый этой инструкцией в случае таблицы item.

```
mysql> DESCRIBE item;
+-------------+--------------+------+-----+-------------+----------------+
| Field       | Type         | Null | Key |Default      |Extra           |
+-------------+--------------+------+-----+-------------+----------------+
| ID          | int(6)       |      | PRI |NULL         |auto_increment  |
| Name        | char(32)     |      | MUL |             |                |
| Price       | decimal(4,2) |      |     |0.00         |                |
| Description | char(255)    | YES  |     |No Description|               |
+-------------+--------------+------+-----+-------------+----------------+
4 rows in set (0.00 sec)
```

Аналогичную информацию выдает инструкция SHOW COLUMNS, общий формат которой таков:

```
SHOW COLUMNS
        FROM имя_таблицы
```

Существует много разных инструкций семейства SHOW. Например, инструкция SHOW DATABASES выдает список всех баз данных, доступных текущему пользователю. С помощью инструкции SHOW TABLES можно получить список таблиц, существующих в стандартной базе данных.

Столбец ID созданной выше таблицы item содержит целые числа в диапазоне от О до 999999. Значения NULL в нем недопустимы, на что указывает спецификатор NOT NULL. Кроме того, столбец работает в режиме счетчика (флаг AUTO_INCREMENT). Это означает, что при вставке в таблицу новой строки значение ее столбца ID будет вычислено автоматически на основании значения в предыдущей строке. Нумерация начинается с единицы. Это хороший способ присвоения записям уникальных идентификаторов.

У читателей может возникнуть вопрос: почему MySQL сообщает, что по умолчанию значение столбца ID равно NULL? Ведь в инструкции CREATE TABLE стандартное значение не было задано, более того, явно указано, что столбец не может содержать значения NULL. На самом деле значение по умолчанию есть у каждого столбца. Если оно не указано, MySQL сделает выбор самостоятельно. В случае столбцов-счетчиков NULL — это хороший выбор, так как при вставке самой первой строки в столбец будет записано значение 1.

Столбец Name содержит строку из 32-х символов. У каждого элемента таблицы должно быть имя, поэтому столбец помечен спецификатором NOT NULL.

Столбец Price хранит значение стоимости в десятичном формате: четыре цифры до запятой и две — после. Тип с плавающей запятой в данном случае не подходит, потому что он не обеспечивает точность вычислений.

Столбец Description представляет собой текстовое поле максимальной длины: 255 символов. Он может принимать значения NULL, но благодаря наличию предложения DEFAULT они будут преобразовываться в строку 'No Description'.

В конце инструкции CREATE TABLE определяются первичный и вторичный ключи. MySQL проследит, чтобы в столбец ID помещались уникальные значения, а кроме того, для столбцов ID и Name будут созданы индексы, что ускорит поиск в них.

На имена баз данных, таблиц и столбцов накладывается ряд ограничений. Если заглянуть "за кулисы", то окажется, что базы данных представляются в виде каталогов файловой системы сервера, а таблицы являются файлами этих каталогов. Из практических соображений длина имен ограничена 64-мя символами. Если в операционной системе, где установлена СУБД MySQL, в именах файлов учитывается регистр, то имена баз данных и таблиц в MySQL тоже будут чувствительными к регистру. Имена столбцов всегда не зависят от регистра.

Все имена могут состоять из букв, чисел, знаков подчеркивания (_) и символов доллара ($). Из соображений удобочитаемости желательно, чтобы имена начинались с буквы.

Вставка записей

Инструкция INSERT добавляет строку в таблицу. Общий формат ее таков:

```
INSERT INTO имя_таблицы
    VALUES (значение, ...)
```

Необходимо задать значения всех столбцов, причем в том порядке, в котором определения столбцов указывались в инструкции CREATE TABLE. В главе 13, "Инструкции SQL", будут описаны разновидности инструкции INSERT, позволяющие указывать подмножество столбцов, а также вставлять в таблицу записи, возвращаемые подчиненной инструкцией SELECT.

Инструкция, показанная в листинге 6.4, создает новую строку в таблице item. В столбец ID будет записано ближайшее доступное число. Обратите внимание на то, что строковые значения берутся в одинарные кавычки.

```
INSERT INTO item VALUES(
    NULL,
    'toothbrush',
    1.25,
    'A deluxe toothbrush'
)
```

Обновление записей

Для изменения полей существующих записей предназначена инструкция UPDATE, общий формат которой следующий:

```
UPDATE имя_таблицы
   SET выражение, ...
 WHERE условие_отбора
```

В выражении стоит оператор присваивания, по левую сторону от которого указывается имя столбца, а по правую — записываемое значение. Если требуется обновить более одного столбца, следует задать соответствующее число выражений, разделенных запятыми.

Предложение WHERE задает правило отбора обновляемых строк. Если оно отсутствует, изменению подвергнутся все строки таблицы. Инструкция, показанная в листинге 6.5, обновляет значения столбцов Name и Price элемента с идентификатором 1. Другие строки остаются нетронутыми.

```
UPDATE item
   SET Name='Toothbrush', Price=1.15
 WHERE ID=1
```

Условия отбора строк необходимо тщательно проверять, чтобы по ошибке не удалить тысячу строк вместо одной. На всякий случай можно воспользоваться опцией --safe-updates программы mysql, указав в ней максимальное число удаляемых строк. Но еще лучше перед выполнением инструкции UPDATE ввести эквивалентную инструкцию SELECT, чтобы проверить, какие записи соответствуют условию отбора.

Удаление записей

Инструкция DELETE удаляет строку из таблицы. Общий формат инструкции таков:

```
DELETE FROM имя_таблицы
       WHERE условие_отбора
```

Если предложение WHERE **отсутствует, будут удалены все строки таблицы. В листинге 6.6 демонстрируется удаление отдельной строки.**

```
DELETE FROM item
 WHERE ID=16
```

Запросы

Инструкция SELECT извлекает строки из одной или нескольких таблиц. С ее помощью можно реализовать большинство реляционных операций, описанных в главе 5, "Реляционная модель". Формат этой инструкции сложнее, чем у других инструкций:

```
SELECT имя_столбца,...
  FROM имя_таблицы, ...
 WHERE условие_отбора
 GROUP BY имя_столбца,...
 ORDER BY имя_столбца, ...
 LIMIT лимит
```

Правда, многие предложения инструкции являются необязательными.

```
SELECT * FROM item
```

В листинге 6.7 представлен простейший вариант инструкции SELECT. В данном случае отображаются все столбцы всех строк таблицы item. Символ звездочки (*) заменяет собой имена всех столбцов. Запрос, показанный в листинге 6.8, возвращает тот же результат, но имена столбцов на этот раз перечислены явно. Порядок имен важен, так как именно в этом порядке столбцы войдут в результирующую таблицу.

```
SELECT ID, Name, Price, Description
  FROM item
```

Если указаны не все столбцы, операция выборки превратится в операцию проекции (см. главу 5, "Реляционная модель"). Дополнительный фильтр записей задается с помощью предложения WHERE **(листинг 6.9).**

```
SELECT ID, Name
  FROM item
 WHERE ID > 2 AND ID < 10
```

Показанное в листинге 6.9 предложение WHERE состоит из двух операций сравнения. В условиях отбора могут присутствовать многие операторы языков третьего поколения, а также целый ряд встроенных функций. Обо всех них пойдет речь в главах 10, "Типы данных, переменные и выражения", и 12, "Встроенные функции".

Можно также вычислять выражения, не ссылаясь на таблицы. Например, в листинге 6.10 показано, как узнать текущие дату и время, "извлекая" значение из встроенной функции NOW().

```
mysql> SELECT NOW();
  NOW()
  2001-04-13  11:54:23
1 row in set (0.00 sec)
```

Объединения

Объединение создается путем указания нескольких таблиц в предложении FROM. Существуют два варианта записи объединений. В первом из них в предложении FROM указывается выражение объединения. Этим способом чаще всего создают внешние объединения. Второй вариант — применение условия отбора, заданного в предложении WHERE, к произведению таблиц, перечисленных в предложении FROM. Так получаются внутренние объединения.

Для примера создадим вторую таблицу it em_option, которая будет участвовать в операциях объединения с таблицей item. В листинге 6.11 показаны инструкции, создающие обе таблицы и заносящие в них данные. В таблице item_option хранятся описания возможных вариантов изготовления товаров, перечисленных в таблице item.

```
CREATE TABLE item (
    ID INT(6) NOT NULL AUTO_INCREMENT,
    Name CHAR(32) NOT NULL,
    Price DECIMAL(4,2) NOT NULL,
    Description CHAR(255) DEFAULT 'No Description',

    PRIMARY KEY(ID),
    KEY (Name)
);

INSERT INTO item VALUES (1, 'Toothbrush', 1.25, NULL);
INSERT INTO item VALUES (2, 'Comb', 2.50, NULL);
INSERT INTO item VALUES (3, 'Brush', 3.00, NULL);
INSERT INTO item VALUES (4, 'Toothpaste', 0.75, NULL);

CREATE TABLE item_option (
    ID INT(6) NOT NULL AUTO_INCREMENT,
    Item INT(6) NOT NULL,
    Name CHAR(32) NOT NULL,

    PRIMARY KEY(ID),
    FOREIGN KEY(Item) REFERENCES Item (ID)
);

INSERT INTO item_pption VALUES (1, 2, 'Red Plastic');
INSERT INTO item_option VALUES (2, 2, 'Blue Plastic');
```

Чтобы создать внутреннее объединение таблиц, достаточно указать их имена в предложении FROM инструкции SELECT. При отсутствии предложения WHERE получится самый общий вариант такого объединения — декартово произведение, в котором каждая строка первой таблицы объединена с каждой строкой второй таблицы. Если же задано условие отбора, то в результаты запроса попадут лишь те строки таблицы произведения, в которых значения связанных столбцов удовлетворяют требуемому критерию *{связанным* называется столбец, присутствующий в условии отбора).

В листинге 6.12 выполняется объединение двух таблиц по равенству столбцов item. ID и item_option. Item, т.е. для каждой строки первой таблицы находится соответствующий элемент второй таблицы. Из объединенной таблицы отбираются три столбца: с названием и ценой товара, а также с названием варианта его изготовления.

```
mysql> SELECT i.Name, i.Price, o.Name AS 'Option Name'
    ->   FROM item i, item_option o
    ->  WHERE i.ID = o.Item;
+------+-------+--------------+
| Name | Price | Option Name  |
+------+-------+--------------+
| Comb |  2.50 | Red Plastic  |
| Comb |  2.50 | Blue Plastic |
+------+-------+--------------+
2 rows in set (0.00 sec)
```

В листинге 6.12 содержатся примеры псевдонимов таблиц и столбцов. Третий столбец результатов запроса называется так же, как и первый, поэтому во избежание путаницы он переименуется в Option Name. На псевдонимы столбцов можно ссылаться в предложениях GROUP BY и ORDER BY, рассматриваемых ниже.

Псевдонимы таблиц необходимы для того, чтобы не возникала путаница с одноименными столбцами. Например, в обеих таблицах есть столбец Name, поэтому MySQL выдаст сообщениеоб ошибке, если не уточнить имя таблицы, к которой относится столбец. Псевдоним таблицы обычно выбирается максимально коротким, чтобы полные имена столбцов было удобно набирать. Длина псевдонима не может превышать 255 символов.

В листинге 6.13 показано то же самое внутреннее объединение, что и в листинге 6.12, но на этот раз *в* предложении FROM присутствует операция INNER JOIN. В подобном случае условие отбора переносится из предложения WHERE в предложение ON.

```
SELECT i.Name, i.Price, o.Name AS 'Option Name'
  FROM item i INNER JOIN item_option o
    ON i.ID = o.Item
```

Иногда внутреннее объединение не дает нужных результатов. Оно формирует пары только тех строк, между которыми найдено соответствие. Все остальные строки исключаются. Чтобы включить в полученные выше результаты названия все товаров, нам потребуется выполнить внешнее объединение, в данном случае левое (листинг 6.14).

```
mysql> SELECT i.Name, i.Price, o.Name AS 'Option Name'
    ->   FROM item i LEFT JOIN item_option o
    ->     ON i.ID = o.Item;
+------------+-------+--------------+
| Name       | Price | Option Name  |
+------------+-------+--------------+
| Toothbrush |  1.25 | NULL         |
| Comb       |  2.50 | Red Plastic  |
| Comb       |  2.50 | Blue Plastic |
| Brush      |  3.00 | NULL         |
| Toothpaste |  0.75 | NULL         |
+------------+-------+--------------+
5 rows in set (0.00sec)
```

Как видите, отображаются пять строк. В объединенную таблицу вошли все строки таблицы item, даже те, которым не найдено соответствие. Строка, не имеющая "пары", дополняется значениями NULL.

Упорядочение результатов запроса

Предложение ORDER BY содержит имена столбцов, по которым осуществляется сортировка результатов запроса. На столбец можно сослаться по имени (краткому или полному), псевдониму или порядковому номеру в предложении SELECT (нумерация столбцов начинается с единицы). В листинге 6.15 таблица результатов сортируется по названию товара.

```
SELECT i.Name, i.Price, o.Name AS 'Option Name'
  FROM item i LEFT JOIN item_option o
    ON i.ID = o.Item
 ORDER BY i.Name
```

Предложение ORDER BY может содержать не одно имя, а целый список. Столбцы перечисляются в порядке убывания приоритетов. В первую очередь записи сортируются по столбцу, указанному первым. Если возникает "конфликт" (две или более строк имеют одинаковое значение в данном поле), проверяется второй столбец, затем третий и т.д. По умолчанию сортировка всех столбцов ведется по возрастанию. Если нужно задать обратный порядок, укажите рядом с именем столбца ключевое слово DESC. В листинге 6.16 показан пример сортировки товаров по цене, начиная от самых дорогих.

```
SELECT i.Name, i.Price, o.Name AS 'Option Name'
  FROM item i LEFT JOIN item_option o
    ON i.ID = o.Item
 ORDER BY i.Price DESC
```

Группировка результатов запроса

Предложение GROUP BY позволяет группировать записи, вошедшие в результаты запроса, с тем чтобы над ними можно было выполнять статистические функции. Все элементы списка возвращаемых столбцов должны иметь одно значение для каждой группы строк. Столбцы, указываемые в предложении GROUP BY, называются *столбцами группировки*. Они по определению имеют одинаковое значение во всех строках группы. В предложении SELECT запроса с группировкой разрешается указывать только столбцы группировки истатистические функции, а также выражения с ними.

В листинге 6.17 представлен запрос, в котором записи объединенной таблицы группируются по названию товара и его цене. В принципе, достаточно выполнять группировку только по полю Name, но в этом случае столбец Price нельзя было бы включить в результаты запроса.

```
mysql> SELECT i.Name, i.Price, COUNT(o.ID) AS 'Options'
    ->   FROM item i LEFT JOIN item_option o
    ->     ON i.ID = o.Item
    ->  GROUP BY i.Name, i.Price
    ->  ORDER BY Options;
+------------+-------+---------+
| Name       | Price | Options |
+------------+-------+---------+
| Toothpaste |  0.75 |       0 |
| Toothbrush |  1.25 |       0 |
| Brush      |  3.00 |       0 |
| Comb       |  2.50 |       2 |
+------------+-------+---------+
4 rows in set (0.00 sec)
```

В третьем столбце содержатся результаты выполнения встроенной статистической функции COUNT (). Для каждой группы она подсчитывает число значений, не равных NULL.

Ограничение числа возвращаемых записей

С помощью предложения LIMIT можно ограничивать число записей, возвращаемых в упорядоченной таблице результатов запроса. Если задано одно число, будет выдана таблица, содержащая указанное количество строк, начиная с самой первой.

Если указаны два числа, разделенных запятой, то первое из них определяет начальную строку выборки, а второе — размер выборки.

В листинге 6.18 показан запрос, возвращающий описание товара, у которого наибольшее число вариантов изготовления. Если таких записей окажется две или более, будет выбрана та из них, которая стоит первой в списке. Предложение LIMIT инструкции SELECT является специфичной особенностью MySQL.

```
SELECT i.Name, i.Price, count(o.ID) AS 'Options'
  FROM item i LEFT JOIN item__option o
    ON i.ID =o.Item
 GROUP BY i.Name, i.Price
 ORDER BY Options DESC, Name
 LIMIT 1
```

Изменение определения таблицы

Инструкция ALTER TABLE позволяет менять определение таблицы. Можно добавлять, удалять и модифицировать определения столбцов, а также добавлять и удалять индексы. Допускается переименование таблицы.

Инструкция, показанная в листинге 6.19, добавляет в таблицу item столбец Inventory, в котором будет указываться количество товара на складе. После выполнения запроса каждая запись таблицы будет содержать в поле Inventory значение 0.

```
ALTER TABLE item
    ADD COLUMN Inventory INT(4) NOT NULL DEFAULT 0
```

ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

В этой главе…

Спецификация требований
Спецификация проекта
Составление схемы базы данных
Реализация модели
Тестирование
Планирование жизненного цикла

Глава 7

Удачные базы данных редко возникают сами по себе, по мере добавления в них таблиц. Неизбежно приходится сталкиваться с множеством проблем, например с избыточностью. В основе любого успешного проекта лежат тщательный анализ и концептуальное моделирование. В этой главе мы поговорим о том, как осуществляется планирование, проектирование и реализация баз данных.

Процесс проектирования начинается с постановки исходных требований. Далее следует моделирование логической структуры данных. Обычно при этом создается графическое представление модели, называемое диаграммой отношения объектов. После этапа проектирования приступают к реализации базы данных. Здесь важно правильно выбрать типы столбцов таблиц.

Будучи введенной в эксплуатацию, база данных требует сопровождения. Исходные требования могут меняться, вследствие чего возникают новые проблемы. Как и в случае любой сложной системы, при создании базы данных необходимо спланировать ее жизненный цикл.

В основном в настоящей главе излагаются стандартные принципы разработки программных систем, примененные к базам данных. Некоторые из идей основаны на личном опыте автора. В приложении Е, "Пример базы данных", содержится готовый пример структуры базы данных.

Спецификация требований

Прежде чем начать проектировать систему, важно понять ее назначение. Обычно формулируют лишь общее словесное требование, например "нам нужно хранить информацию о клиентах в базе данных". Традиционный подход, провальный в своей наивной простоте, заключается в создании на скорую руку прототипа, удовлетворяющего примерно двадцати пяти процентам реальных требований. Далее в прототип начинают вносить изменения, пытаясь приблизить его к желаемой функциональности, но в ходе последующего тестирования выясняется, что исходные требования к прототипу были неверны и подлежат модификации.

Описанная ситуация встречается повсеместно. Будем справедливы: она не обязательно означает провал. Зачастую на каждом следующем витке процесса проектирования разработчики все яснее начинают представлять себе требования к системе, вследствие чего проект "стабилизируется". Но главной проблемой подобного экспериментаторского подхода остается риск того, что проект никогда не стабилизируется.

Альтернативный вариант состоит в тщательной проработке исходных требований к системе. Это должно быть следствием планирования и анализа. Первый этап заключается в определении основных целей системы. Можно начать со стадии "мозгового штурма", а затем составить письменное описание целей. Поинтересуйтесь требованием заинтересованных лиц и уточните еще раз задачи системы.

Постарайтесь мыслить общими категориями, не привязываясь ни к какому языку программирования. Задачи системы должны быть функциональны по своей природе и понятны каждому. Если офис завален кипами документов, целью базы данных может стать сокращение времени поиска информации о клиентах. Приложению, хранящему данные в обычных файлах, может потребоваться предоставить другим программам простой доступ к своим данным.

Из сформулированных задач системы вытекает набор требований к ней. Если цели являются самыми общими, то требования должны быть предельно конкретны. Все требования необходимо свести в документ, называемый *спецификацией требований*. В этом документе, по сути, описано ожидаемое поведение системы как "черного ящика". Сюда входят набор функциональных требований и предусмотренные ограничения системы. Принципы ее функционирования не должны здесь упоминаться.

Предположим, к примеру, что одним из требований к системе является необходимость уведомления об изменении информации о клиенте. Отсюда вытекает требование фиксировать время последнего изменения записей. Способ реализации этого требования должен быть оставлен на усмотрение разработчика. Можно добавлять метку времени к каждой записи каждой таблицы, вести журнал операций и т.д. Директивный выбор конкретного варианта может быть продиктован лишь наличием других требований, например обязанностью взаимодействия с уже существующей системой, которая, допустим, умеет читать журнальный файл определенного формата.

Спецификация требований является формальной и структурированной, но она должна быть понятна людям, не являющимся экспертами в технологии реализации баз данных. Описание поведения системы есть часть контракта между заказчиком и изготовителем. Четкое изложение исходных требований позволит избежать недоразумений, которые дорого обойдутся на поздних этапах разработки. Вряд ли стоит говорить о том, насколько важна точность формулировок этого документа. По возможности требования должны выражаться в легко измеримых величинах. Например, если сказано, что скорость поиска информации о клиенте по номеру социального страхования не должна превышать 10 с, то это невызывает возражений и впоследствии легко проверяется. Если же сказано лишь, что система должна быстро находить записи, могут возникнуть вопросы.

Ниже перечислены шесть качеств, которым обязана удовлетворять идеальная спецификация требований:

1) должно быть описано только внешнее поведение системы;
2) должны быть определены ограничения реализации;
3) спецификация должна легко модифицироваться;

4) спецификация должна служить справочным руководством для тех, кто будет заниматься сопровождением системы;
5) должен быть заранее описан цикл функционирования системы;
6) должна быть предусмотрена приемлемая реакция на нежелательные события.

В спецификации должно присутствовать описание лишь внешних аспектов функционирования системы. Каждое требование должно быть выражено в виде ответа на вопрос "Что", а не "Как". Что делает система, если пользователь хочет узнать, когда клиент сделал заказ? Она возвращает дату заказа. Вопрос о том, как она определяет эту дату, рассматривается в спецификации проекта.

В спецификации требований должны быть четко определены ограничения системы. Это лучше всего выражается в таких показателях, как общее количество записей о клиентах и максимальная задержка между вводом запроса и получением результатов. Эти ограничения предопределяют выбор разработчиком определенных решений. Система, хранящая миллионы записей, реализуется не так, как система, хранящая лишь несколько сотен записей.

Спецификация должна быть составлена так, чтобы впоследствии ее легко можно было модифицировать. Отсюда вывод: избегайте многословия и повествовательного стиля. Это не только усложнит модификацию, но сделает документ сложным для восприятия. Нумеруйте разделы документа и поясняйте сложные места диаграммами.

Спецификация требований должна быть документом, который поможет будущему программисту познакомиться с возможностями системы. Не удивляйтесь, если через полгода таким программистом окажетесь вы сами. И не допустите, чтобы данное качество спецификации вступило в конфликт с предыдущим качеством: простотой модификации. Спецификация требований — это "визитная карточка" системы, но не ее основная документация.

В спецификации должен быть описан весь жизненный цикл функционирования системы. Редко какие системы вечно остаются неизменными, поэтому необходимо отразить требования к сопровождению системы. Нужно также учесть возможность перехода на новую систему, если эта устареет.

В спецификации должно быть описано поведение системы при неблагоприятных обстоятельствах. Таковыми являются, с одной стороны, аварийные сбои системы, с другой — неправильный ввод пользователем данных. Если система должна остаться доступной после выхода из строя жесткого диска, необходимо предусмотреть создание активной резервной копии.

Ниже перечислены девять рекомендуемых разделов спецификации:

1) обзор целей и задач системы;
2) среда функционирования и разработки;
3) внешние интерфейсы и потоки данных;
4) функциональные требования;
5) требования к производительности;
6) обработка исключительных ситуаций;
7) приоритеты реализации;
8) возможные модификации;
9) проектные рекомендации.

Начальный обзор должен занимать не больше страницы и содержать описание того, для чего предназначена система. Важно объяснить, что привело к разработке системы. Из этого вытекают начальные требования, а из них — проект системы.

Наша книга посвящена программе MySQL и базам данных, но в полной программной системе всегда есть какие-то подсистемы. Web-приложению требуется Web-сервер, сервер приложений и Web-броузер. Можно составить общую спецификацию всей системы или отдельно описать каждую подсистему, но позаботьтесь о том, чтобы база данных упоминалась в правильном контексте.

Знание среды функционирования помогает определить список внешних интерфейсов. MySQL работает по технологии "клиент/сервер". Клиенты подключаются по протоколам TCP/IP, через UNIX-сокеты или именованные каналы. СУБД может создавать на диске журнальные файлы и образы таблиц. Опишите потоки данных в системе. Лучше всего сделать это с помощью диаграмм.

Функциональные требования составляют основную часть документа. Если выше уже была составлена диаграмма потоков данных, то общее представление о том, каким образом система должна быть разбита на модули, уже имеется. Чем выше степень такого секционирования, тем проще будет проектировать систему, потому что появится возможность сосредоточиться на решении более локальных проблем.

Требования к производительности накладывают ограничения на функционирование системы. Должны быть заданы как минимальные, так и максимальные требования. Например, следует указать минимальное число записей, поддерживаемое системой, и максимальное время выполнения запроса. Сюда также входят нефункциональные ограничения, в частности требуемые технологии. Если все программисты в команде являются экспертами Java, имеет смысл выбрать Java языком реализации.

В разделе обработки исключительных ситуаций описана работа системы в случае аппаратных сбоев или неправильного ввода данных. Подумайте, как должна повести себя система в момент выхода из строя важного аппаратного компонента. Определите требуемую периодичность создания резервных копий: каждый час, раз в день или раз в неделю, а может и реже. Решите, что нужно делать с неправильными данными: отказаться добавлять запись или подставить значения по умолчанию.

Если заказчик настаивает на определенной очередности реализации модулей, опишите ее. Личный опыт автора говорит о том, что заказчик, которого поджимают сроки, будет просить в первую очередь реализовать самые необходимые для него функции. Остальное может быть не критичным, и заказчик согласится подождать. Важно, чтобы разработчики и программисты знали об этом заранее.

Любая система рано или поздно подвергается модификации. Ее функции улучшаются, а требования к нейрастут. Система, которая поначалу поддерживала одновременную работу пяти пользователей, через год должна будет справляться уже с десятью пользователями. Подобную возможность следует предусмотреть заранее.

Последняя часть спецификации требований содержит список рекомендаций разработчикам. Здесь можно заранее предупредить разработчиков о потенциальных просчетах, привести краткое описание схожего проекта и предложить конкретные подходы к проектированию.

Спецификация проекта

В спецификации проекта описывается, как система реализует предъявленные к ней требования. Здесь документируются принятые решения, касающиеся управления потоками данных, реализации системы и взаимодействия ее компонентов.

Ниже приведен перечень свойств, которыми должна обладать хорошая спецификация проекта:

1) должно быть упомянуто каждое исходное требование;
2) должны быть определены возможные варианты реализации;
3) спецификация должна легко модифицироваться;
4) спецификация должна служить справочным руководством для тех, кто будет заниматься сопровождением системы;
5) спецификация должна придерживаться простых решений;
6) связи между модулями должны быть слабыми.

Первые два свойства отражают основное назначение спецификации: выбрать точный способ реализации исходных требований. Требования простоты модификации и справочного использования унаследованы от предыдущей спецификации. Последние два свойства свидетельствуют об оптимальности принятых решений.

Все, что перечислено в спецификации требований, должно найти свое отражение в спецификации проекта. О полноте проектного реше ния судят, проверяя все исходные требования. Если какое-то требование осталось нереализованным, оно должно быть помечено как таковое. Например, во входной спецификации может говориться о необходимости обеспечить многоязыковую поддержку. Но если ограниченные сроки и бюджет не позволяют этого сделать, разработчик может остановиться только на основном языке. Отдельным параграфом следует дать обоснование такого решения.

В основном разработчик решает, как, собственно, создавать систему. Конкретные детали реализации оставляются на усмотрение программистов, а в проектной части описываются принципы реализации. Нет необходимости указывать имя каждой переменной, достаточно задать правила выбора имен. Язык программирования, СУБД и другие структурные компоненты выбираются на этапе проектирования.

Проектирование— это итерационный процесс, поэтому проектная спецификация должна быть такой, чтобы изменения было легко вносить. Разбейте документ на логические разделы и пронумеруйте их. Существуют программы, предназначенные для построения проектных диаграмм. Среди них отметим открытый пакет Dia *(www.lysator.liu.se/alla/dia)*.

Программисты руководствуются спецификацией проекта при создании конечного продукта. И впоследствии они неоднократно обращаются к ней в процессе исправления ошибок и внесения улучшений. Убедитесь в том, что в документе описаны все детали функционирования системы.

Хороший проект является простым. Простые проекты легче понять, следовательно, реализация получится более качественной. Чрезмерная сложность проекта обычно свидетельствует о преждевременных попытках оптимизировать систему. В контексте баз данных процесс оптимизации называется нормализацией (рассматривается в главе 8, "Нормализация").

Полученное проектное решение должно иметь модульную структуру, в которой все модули обособлены, но в то же время связаны друг с другом. Обособленный модуль служит одной-единственной цели. Наличие таких модулей помогает сделать систему простой, так как при реализации модуля можно сосредоточиться только на нем и не беспокоиться о том, как реализуются остальные модули.

Принцип связности касается взаимодействия модулей друг с другом. Оно должно происходить посредством четко определенных интерфейсов, которые позволяют менять внутреннюю структуру модулей, не затрагивая работу других частей системы. Это называется слабой связностью. Сильная связность возникает, когда структуры модулей оказываются взаимозависимыми.

При проектировании баз данных слабая связность достигается путем обособления основных групп информации. Например, торговая компания использует адреса клиентов в разных ситуациях: при оформлении заказов на покупку, при оформлении заказов на доставку, в маркетинговых отчетах. Нет необходимости повторять адреса в каждой таблице. Можно вынести их в отдельную таблицу, установив с ней связи из других таблиц посредством внешних ключей.

В спецификации проекта должны присутствовать четыре важных раздела:

1) стратегия;
2) архитектура;
3) правила;
4) детали проекта.

В разделе стратегии описаны внешние компоненты и средства, которые должны быть использованы в процессе последующей реализации. Сюда входят операционная система, СУБД и другие программные платформы. Их выбор влияет на архитектуру. Рассмотрите основные методы решения главных проблем.

В разделе архитектуры дается высокоуровневый анализ системы и рассматриваются связи между модулями. Детали работы самих модулей сюда не относятся, но диаграммы таблиц и вешних ключей будут очень полезными.

В разделе правил собраны рекомендации программистам. Сюда должны быть включены требования к оформлению исходных текстов. Здесь же можно указать правила интеграции новых модулей, местоположение совместно используемых ресурсов и принципы тестирования.

В последнем разделе описаны все модули и их компоненты. Описание компонентов дается в виде названий алгоритмов или псевдокода. У таблиц базы данных должны быть перечислены столбцы и внешние ключи.

Составление схемы базы данных

При проектировании часто пользуются диаграммами, так как успех проекта в значительной степени зависит от точности его спецификации. На основе этой спецификации программисты реализуют модули, и любая неоднозначность приведет к тому, что исходные требования не будут соблюдены.

Существуют два основных типа диаграмм: диаграммы потоков данных и диаграммы отношений объектов (entity relationship, ER). На примере первых демонстрируются функциональные преобразования, претерпеваемые данными при переходе из

одного модуля в другой. Диаграммы отношений объектов отображают логическую структуру данных и связи между модулями. Потоковые диаграммы плохо подходят для описания баз данных. Реляционные базы данных лучше всего описываются в виде структур данных.

Диаграммы отношений объектов соответствуют одному из трех уровней абстракции: физическому, программному или концептуальному. Диаграмма физического уровня описывает аппаратные компоненты. Это взгляд на систему как на совокупность физических устройств. На программном уровне система анализируется с точки зрения одного или нескольких программных компонентов. Концептуальный уровень представляет собой взгляд на систему со стороны пользователя.

Спецификация требований описывает систему на концептуальном уровне. Диаграммы в данном случае отражают происходящие в системе бизнес-процессы и взаимодействие пользователя с ней. На рис. 7.1 изображен продавец, решающий две системные задачи. Первая задача заключается в проверке цены продаваемого товара. Вторая задача — это собственно продажа товара. Здесь возникает подзадача: добавление записи о клиенте.

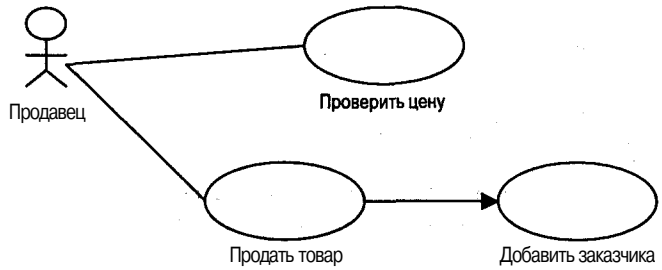


Рис. 7.1. Концептуальная диаграмма

Каждый программный компонент имеет собственное представление о системе. СУБД MySQL рассматривает данные как информационные блоки файлов на диске. Приложения работают с данными как с записями, которые возвращаются в результате SQL-запросов.

На рис. 7.2 приведена диаграмма отношений между четырьмя таблицами. В этих таблицах хранятся сведения об адресах, клиентах, заказах и товарах. Каждый прямоугольник представляет собой таблицу. Линии, соединяющие таблицы, показывают отношения "внешний ключ — первичный ключ". Между таблицами клиентов и заказов существует отношение "один ко многим". Клиент может сделать произвольное число заказов (в том числе ни одного), но любой заказ принадлежит только одному клиенту. Кружок на диаграмме говорит о том, что отношение является необязательным.

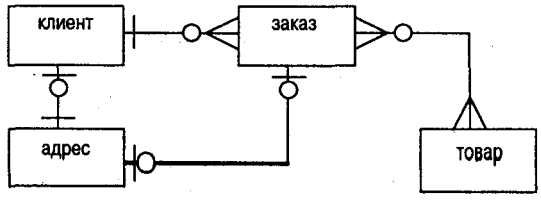


Рис. 7.2. Диаграмма таблиц

В заказ включается один или несколько товаров. Обратите внимание на отношение "многие ко многим". Логически каждый заказ связан с группой товаров, а один и тот же товар может входить в несколько заказов. Вспомните из главы 5, "Реляционная модель", что реляционная модель не позволяет напрямую формировать отношения "многие ко многим". В ходе этапа проектирования это отношение преобразуется таким образом, что появится промежуточная таблица.

В таблицу адресов вынесены столбцы, общие для таблиц заказов и клиентов. В таблице заказов указывается адрес доставки, если заказ делается через Web-узел. В таблице клиентов приводятся их основные адреса, используемые при непосредственном общении с клиентами. Наличие таблицы адресов позволяет гарантировать, что все адреса будут иметь одинаковую структуру.

На физическом уровне описывается архитектура системы. Диаграммы этого уровня отражают взаимодействие аппаратных компонентов. Здесь могут быть представлены сетевые соединения, а также каналы записи данных в постоянные хранилища.

Языки моделирования

Существует несколько стандартов моделирования информационных структур вообще и баз данных в частности. Среди них диаграммы Бейчмана, модель отношений объектов, стандарты IDEF1X и UML (Uniform Modeling Language — единый язык моделирования). Диаграммы Бейчмана были разработаны Чарльзом Бейчманом для описания сетевых баз данных, но некоторые их идеи применимы и для реляционных баз данных. Модель отношений объектов впервые была описана Питером Ченом (Peter Chen) в 1976 г. под влияниемстатьи доктора Кодда, посвященной реляционной алгебре. Стандарт IDEF1X был принят правительством США в 1993 г. UML— это популярный язык моделирования, созданный Гради Бучем (Grady Booch), Айваром Якобсоном (Ivar Jacobson) и Джимом Рамбо (Jim Rumbaugh) из компании Rational Software Corporation.

Люди часто говорят "база данных", имея в виду "система управления реляционными базами данных". Это объясняется двумя причинами: удобством и тем, что реляционная модель намного превосходит по популярности все остальные модели. Точно так же диаграммами отношений объектов стали называть любые диаграммы, в которых информационные сущности изображаются в виде замкнутых фигур, а отношения между ними обозначаются линиями. Строгое определение таких диаграмм было дано в исходной работе Чена под названием "Модель отношений объектов: попытка унифицированного представления данных" ("The Entity-Relationship Model: Toward a Unified View of Data").

Дополнительную информацию о стандарте IDEF1X можно получить на Web-узле *www.idef.com*. Там выложен 145-страничный документ, содержащий описание методик составления диаграмм.

Компания Rational Software ведет Web-узел, посвященный языку UML *(www.rational.com/umt)*. Стандарт UML основан на методологии объектно-ориентированного проектирования, позволяющей создавать многократно используемые программные компоненты. Концепции многих моделей хорошо выражаются диаграммами UML. Правда, считается, что традиционные диаграммы лучше подходят для описания информационных систем, в частности баз данных.

Диаграммы отношений объектов

Диаграммы отношений объектов отображают три основных типа информации: объекты (сущности), отношения и характер этих отношений. Объекты представляются прямоугольниками. Внутри прямоугольника находится имя объекта. Объект — это общее понятие, но при моделировании реляционных баз данных оно означает таблицу. На рис. 7.3 представлено типичное изображение объекта.

Прямые линии обозначают отношения между объектами. В реляционных базах данных отношения реализуются путем объединения двух таблиц по группе столбцов, общих для обеих таблиц. Характер отношения определяется числом строк, участвующих в нем с каждой стороны. Таблице, над которой выполняется объединение, может быть поставлена в соответствие одна строка или несколько. Перпендикулярная черта означает единичную связь, а тройная линия — множественную.

На рис. 7.4 изображено отношение "один к одному" между таблицами клиентов и адресов. Специальные символы, выражающие характер отношения, ставятся на концах линии. Отношение может "читаться" в любом направлении, например "у каждого клиента есть один адрес" или "каждый адрес принадлежит кодному клиенту".

Рис. 7.3. Объект

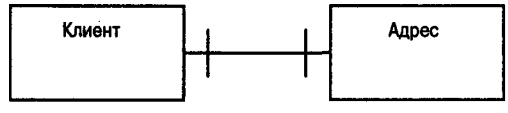


Рис. 7.4. Отношение "один к одному"

На рис. 7.5 изображено отношение "один ко многим" между таблицами категорий и товаров. Рисунок читается так: "в каждую категорию входит один или несколько товаров" либо "каждый товар принадлежит к одной и только к одной категории".

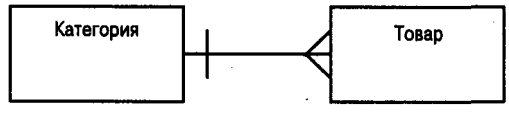


Рис. 7.5. Отношение "один ко многим"

Отношения между объектами могут быть необязательными. Чтобы подчеркнуть это, на линии рядом с символом отношения ставят незаштрихованный кружок. На рис. 7.6 изображено необязательное отношение между таблицами заказов и адресов. Оно читается следующим образом: "в заказе может быть указан один адрес" или "каждый адрес принадлежит к одномузаказу".

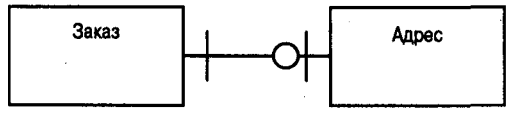


Рис. 7.6. Необязательное отношение "один к одному"

Создание диаграмм

Прежде чем приступать к созданию диаграмм отношений объектов, необходимо определиться с самими объектами. Анализ поведения системы поможет установить способы получения пользователями информации. Составьте список информационных объектов.

Создание диаграммы — это итерационный процесс. Получив первую версию диаграммы, вы обязательно обнаружите, что какие-то объекты и отношения пропущены. Продолжайте улучшать модель до тех пор, пока она не будет завершена. Затем сравните полученную диаграмму с исходными требованиями и определите ее полноту.

Если моделируется существующий бизнес-процесс, используйте бумажные документы и формуляры для определения атрибутов каждого объекта. Например, картотеку компании, торгующей автомобилями, наверняка можно взять за основу при выборе способа описания автомобилей. Поля учетной карточки трансформируются в столбцы таблицы базы данных.

Не все из нас художники, поэтому для составления диаграмм лучше пользоваться специализированными приложениями. Наиболее популярны пакет Visio компании Microsoft *(www.microsoft.com/office/visio)* и его открыто распространяемый эквивалент Dia *(www.lysator.liu.se/~alla/dia)*. В последнем применяется библиотека GTK, поэтому существуют его версии для Windows.

Пакеты Visio и Dia предназначены для создания диаграмм общего назначения. Программа ArgoUML *(http://argouml.tigris.org)* строит диаграммы UML. Она написана на Java, поэтому может применяться в любой операционной системе, где есть виртуальная машина Java.

Пакет ERWin компании Computer Associates *(www.ca.com)* позволяет моделировать базы данных. Он может подключаться к СУБД и строить диаграммы существующих баз данных, а также создавать базы данных на основании диаграмм. Аналогичные функции выполняетпакет DeZign компании Datanamic *(www.datanamic.com)*. К сожалению, оба они доступны только в Windows.

Реализация модели

Когда проект базы данных завершен, переходят к ее реализации. В соответствии с диаграммами создаются инструкции CREATE TABLE. Будьте готовы вернуться к этапу проектирования для внесения очередных модификаций: переход на язык SQL часто открывает различные упущения.

В спецификации проекта были описаны столбцы каждой таблицы. Для них необходимо подобрать типы данных, поддерживаемые в SQL. Дополнительно можно создать индексы, хотя зачастую схема их использования настолько сложна, что придется прибегнуть к инструкции EXPLAIN (описана в главе 13, "Инструкции SQL").

Все SQL-инструкции, создающие базу данных, должны быть сведены в одном или нескольких текстовых файлах, а не вводиться в интерактивном режиме. Это позволит стирать базу данных и создавать ее "с нуля". Эти же файлы послужат в качестве документации. Перед каждой инструкцией CREATE TABLE вставьте комментарии, поясняющие назначение таблицы. Комментарии игнорируются синтаксическим анализатором, поэтому они не сохраняются в базе данных, зато дают возможность человеку,

сопровождающему базу данных, понять ее структуру. Синтаксис комментариев описан в начале главы 13, "Инструкции SQL".

Можно также пользоваться опцией COMMENT инструкции CREATE TABLE. Такие комментарии хранятся вместе с таблицей. Для их просмотра необходимо ввести инструкцию SHOW TABLE STATUS.

При переходе от этапа проектирования к этапу реализации необходимо придерживаться пяти ключевых принципов:

- первичные ключи следует помечать спецификатором NOT NULL;
- пользуйтесь флагом AUTO_INCREMENT для автоматического создания идентификаторов;
- внешние ключи должны ссылаться на первичные ключи;
- отношения "многие ко многим" должны устанавливаться через промежуточную таблицу;
- таблицы, между которыми существуют отношения "один к одному", лучше объединять.

Как правило, всякая таблица должна иметь первичный ключ. Вспомните из главы 5, "Реляционная модель", что столбцы первичных ключей необходимо объявлять со спецификатором NOT NULL. В большинстве таблиц первичный ключ представляет собой отдельный столбец целочисленных значений, так как зачастую трудно сказать заранее, будут ли значения других полей уникальными. Например, всем клиентам можно присвоить целочисленные идентификаторы, что позволит различать тезок, но самим клиентам эти идентификаторы не нужны, они могут даже не догадываться об их существовании. В MySQL такие абстрактные первичные ключи реализуются в виде столбцов-счетчиков.

Внешние ключи участвуют в отношениях "один ко многим". Таблица, с которой существует "множественная" связь, будет содержать внешний ключ, указывающий на первичный ключ противоположной таблицы. MySQL не проверяет правильность значений внешних ключей, т.е. предложения FOREIGN KEY инструкции CREATE TABLE являются необязательными. Но есть ряд причин, по которым их все же стоит указывать. Во-первых, это послужит целям документации. Во-вторых, это позволит некоторым приложениям читать схему базы данных и строить на ее основы диаграммы. Наконец, впоследствии может понадобиться передать схему в другую СУБД, где внешние ключи играют более важную роль.

Если на диаграмме присутствуют отношения "многие ко многим", каждое из них должно быть преобразовано в два отношения "один ко многим" с промежуточной таблицей. Первичные столбцы исходных таблиц преобразуются во внешние ключи промежуточной таблицы. Это означает, что у нее будут два целочисленных столбца.

Проверьте целесообразность отношений "один к одному". Бывают случаи, когда они действительно необходимы, но, возможно, вы просто пропустили возможность слияния двух таблиц.

Если бы СУБД MySQL проверяла правильность внешних ключей, пришлось бы задавать правила обновления и удаления записей. В настоящий момент такие функции возлагаются на программиста. В этом есть определенный смысл. Если приложение будет работать с другой СУБД, в которой есть свои правила, и приложение попытает-

ся выполнить изменения в нарушение этих правил, будет выдано сообщение об ошибке. А раз код проверки уже заложен в программы, то дублирование его в СУБД приведет лишь к снижению производительности.

Выбор правильных типов столбцов — это творческий процесс, частично связанный с анализом исходных требований. Если необходимо заботиться об экономии дискового пространства, следует выбирать минимально возможные типы, охватывающие заданный диапазон значений. Например, если известно, что в таблице будет примерно 500 строк, то трехзначного первичного ключа окажется достаточно. Спецификация INT (3) является некорректной, поскольку для типа INT MySQLвсегда использует 32-разрядное целое число. Тип TINYINT занимает один байт, что недостаточно. В рассматриваемом случае подойдет тип SMALLINT, т.е. спецификация типа должна выглядеть так: UNSIGNED SMALLINT (3).

Значения в столбцах-счетчиках должны быть беззнаковыми, потому что отрицательные целые числа будут трактоваться как большие положительные числа. Следовательно, вставка в столбец отрицательного числа приведет к тому, что следующее число окажется очень большим. Таким образом, спецификация типа поля-счетчика всегда должна включать ключевое слово UNSIGNED.

Столбцы, хранящие короткие значения одинаковой длины, например коды деталей, должны иметь тип CHAR. Правда, если в таблице есть поля переменной длины, значения CHAR могут автоматически приводиться к типу VARCHAR. Строковые поля длиной более 255 символов должны иметь тип BLOB или TEXT.

Столбцы, хранящие денежные величины, должны иметь тип DECIMAL. Это тип чисел с фиксированным количеством цифр после запятой. В MySQL десятичные числа хранятся в строковом виде и не подлежат аппроксимации, в отличие от других типов чисел с плавающей запятой. Эффект аппроксимации приводит к тому, что числа, отличающиеся на очень незначительную величину, оказываются равными. В случае сравнения денежных сумм это может означать незапланированную потерю денег. Значения типа DECIMAL преобразуются из строковой формы в числа с плавающей запятой при выполнении математических операций. Чтобы защитить себя от ошибок аппроксимации, выполняйте эти операции в приложениях.

Индексы можно создавать не только для первичных и внешних ключей, но и для обычных столбцов. Индексы ускоряют поиск записей в запросах с объединением, группировкой или упорядочением записей. Наличие индекса позволяет сразу же перейти к нужной записи, а не сканировать всю таблицу по одной записи за раз. Однако трудно предугадать заранее, какие запросы будут выполняться тем или иным приложением. Поэтому может понадобиться проанализировать запросы с помощью инструкции EXPLAIN. Этот процесс рассматривается в главе 26, "Оптимизация".

Тестирование

Тестирование бывает двух видов: на предмет поиска недочетов и на предмет проверки полноты реализации. Недочеты являются следствием человеческих ошибок. Полной считается система, удовлетворяющая всем исходным требованиям.

На основании спецификации требований и спецификации проекта проверяются все функциональные возможности, реализованные в базе данных. Нужно убедиться, что всем информационным сущностям поставлены в соответствие таблицы или

столбцы. Выполните ряд типичных операций над базой данных и убедитесь в получении ожидаемых результатов.

Поиск недочетов поначалу кажется легкой задачей, и лишь впоследствии выясняется, что это не так. Синтаксические ошибки SQL-запросов выявляются сразу же и быстро исправляются, но есть ошибки, которые проявляются лишь в определенных ситуациях. Например, объединение двух таблиц без индекса выполняется быстро при малой загруженности системы. Однако когда 20 пользователей одновременно выполняют один и тот же запрос, производительность падает ниже критической отметки.

Если в исходных требованиях оговорены ограничения производительности, потребуются специальные программы, которые вызовут высокую загруженность базы данных. Только так можно будет проверить ее работу в экстремальных условиях.

В пакет MySQL входят специальные утилиты тестирования, позволяющие измерять производительность сервера. Они представляют собой Perl-сценарии, находящиеся в каталоге sql-bench дистрибутива.

Планирование жизненного цикла

Даже после ввода в эксплуатацию системе необходимы регулярный мониторинг и сопровождение. Исходные требования меняются, от системы ожидается все более высокая производительность. У любой системы есть свой жизненный цикл. Должен быть составлен план, описывающий действия, выполняемые в ходе нормальной эксплуатации системы, ее обновления и последующего списания.

В процессе эксплуатации системе необходим административный контроль. Журнальные файлы разрастаются и могут потребовать усечения. Следует регулярно создавать резервные копии на случай возможной аварии. Внешние события, например сбой питания, могут вызвать повреждение файлов MySQL, в которых хранятся таблицы, и их придется восстанавливать. Все эти вопросы рассматриваются в главе 23, "Администрирование баз данных".

Очень часто недостатки проекта выявляются только после длительного использования системы. Подумайте заранее о будущих улучшениях. Вряд ли допустим полугодичный простой, в течение которого система будет переписана, поэтому нужно составить план внедрения изменений с минимальным дискомфортом для пользователей. В MySQL есть очень гибкие средства изменения таблиц. Часто с помощью одной SQL-инструкции можно интегрировать в базу данных новую информационную структуру.

Рано или поздно должен произойти переход на новую версию системы или полный отказ от нее по причине устаревания. В главе 28, "Перенос данных в разные СУБД", рассказывается о том, как переносить базы данных в другие СУБД.

НОРМАЛИЗАЦИЯ

В этой главе…

Зачем нужна нормализация

Первая нормальная форма

Вторая нормальная форма

Третья нормальная форма

Нормальная форма Бойса-Кодда

Четвертая нормальная форма

Денормализация

Глава 8

Нормализация — это метод организации реляционной базы данных с целью сокращения избыточности. В ходе этого процесса неоптимальная таблица разбивается на две или более таблиц, между которыми создаются отношения. Нормализация является частью этапа проектирования и выполняется над существующими таблицами. Приобретя определенный опыт, можно научиться создавать нормализованные таблицы с первой попытки.

Нормализация позволяет в полной мере реализовать преимущества реляционной модели. Несмотря на все имеющиеся в распоряжении MySQL средства, ничто не мешает разработчику реализовать базу данных так, как если бы это была коллекция файлов. Нормализация заставляет разработчика создавать больше таблиц, равномернее распределяя в них информацию, что приводит к снижению избыточности.

Нормализация определяется в виде набора правил, известных как *нормальные формы*. После своей статьи, посвященной реляционной алгебре, доктор Кодд в 1972 г. опубликовал работу под названием "Дальнейшая нормализация реляционной модели баз данных" ("Further Normalization of the Data Base Relational Model"). В этом документе были описаны первые три нормальные формы. В последующих работах доктора Кодда и других авторов были определены три другие нормальные формы. Каждая нормальная форма основана на предыдущей, поэтому, например, третья форма более желанна, чем вторая.

Устранение избыточности не обязательно означает повышение производительности. Накладные расходы на выполнение операций объединения весьма значительны, поэтому разработчики иногда сознательно идут на нарушение правил нормализации. Это называется *денормализацией*.

Зачем нужна нормализация

Необходимость нормализации диктуется теми же соображениями, что и выбор реляционной СУБД: есть сложная информационная структура, и если выбрать любую другую модель представления данных, возникнет большая избыточность. В главе 4, "Концепции баз данных", рассматривались вопросы управления дублирующимися данными. Стоимость написания кода, который бы синхронизировал дублирующиеся значения, настолько высока, что оптимальным решением оказываются лишь самые простые базы данных.

Нормализация приводит к улучшению целостности данных. Потребность поддержания целостности в приложениях уменьшается, следовательно, повышается их производительность. Растет и производительность самой СУБД. Сокращается число значений NULL, обработка которых незначительно, но все же влияет на производительность, и повышается эффективность индексов.

Нормализация чаще всего выполняется на этапе проектирования, поэтому под рукой не оказывается готовых данных, на которых можно было бы выполнить проверку. Разработчики обычно создают разного рода заменители, например таблицы с небольшим числом строк. Но в действительности необходимо учитывать весь домен значений столбца, что требует немалого воображения. Иногда задача состоит в нормализации рабочей базы данных, но, как правило, приходится трансформировать схему, получаемую на основании диаграмм и инструкций CREATE TABLE.

В табл. 8.1 показано последовательное изменение правил для первых пяти уровней нормализации. Есть еще как минимум две нормальные формы, но их практическая ценность весьма сомнительна. В большинстве случаев достаточно третьей нормальной формы.

Нормальная форма	*Свойства таблицы*
Первая	Содержит информацию об одной сущности, имеет первичный ключ; каждая ячейка содержит одно значение
Вторая	Значения всех столбцов зависят от полного первичного ключа
Третья	Только первичный ключ определяет значения столбцов
Бойса-Кодда	Ни одна часть первичного ключа не зависит от столбца, который сам не может стать первичным ключом
Четвертая	Значения NULL во внешних ключах недопустимы, если этих ключей больше одного

Первая нормальная форма

В реляционной базе данныхтаблицы практически всегда по умолчанию находятся в первой нормальной форме. Проблемы возникают в ситуации, когда необходимо импортировать обычную файловую базу данных в MySQL. В такой базе данных отсут-

ствуют какие бы то ни было отношения, поэтому в ней наверняка есть дублирующаяся информация.

Рассмотрим инструкцию CREATE TABLE, приведенную в листинге 8.1. Это таблица музыкального каталога. Она может хранить описание любой пластинки, любого компакт-диска и любой кассеты, имеющихся в коллекции. Указываются дата приобретения носителя и его нынешняя цена.

```
CREATE TABLE recording (
    Artist VARCHAR(32),          # исполнитель
    Released DATE,               # дата выпуска
    Title VARCHAR(32),           # название
    Format VARCHAR(32),          # формат
    Label VARCHAR(32),           # студия звукозаписи
    Genre VARCHAR(32),           # жанр
    Length INT(11),              # продолжительность
    Purchased DATE,              # дата приобретения
    Cost DECIMAL(11,2},          # цена
    CurrentValue DECIMAL(11,2)   # нынешняя стоимость
);
```

Главный принцип первой нормальной формы заключается в том, что любая запись таблицы должна содержать описание одной сущности. Этому правилу удовлетворяет практически любая таблица, поскольку столбцы выбираются сознательно. В нашем случае в таблице находятся описания предметов музыкальной коллекции. Но если присмотреться, то окажется, что столбцы можно разделить на две группы. Столбцы первой группы (Artist, Released, Title, Format, Label, Genre, Length, CurrentValue) описывают любой экземпляр музыкального носителя, а не только тот, что имеется в коллекции. Столбцы второй группы (Purchased, Cost) относятся к одному конкретному экземпляру. Может показаться, что таблицу следует разбить на две части. Но прежде давайте познакомимся с двумя другими принципами первой нормальной формы.

Второй принцип требует наличия первичного ключа. С технической точки зрения созданная выше таблица не имеет ключа: я просто не указал его в инструкции CREATE TABLE. Вспомните из главы 7, "Проектирование баз данных", что в качестве первичного ключа можно использовать столбец-счетчик. Есть ли альтернативы? Допустим, столбцы Artist и Released формируют составной ключ. Тогда исполнитель не должен одновременно выпускать две разные работы. Это неразумное ограничение, потому что описанная ситуация возможна. Тогда предположим, что первичный ключ образуют столбцы Artist, Released и Title. Кажется глупым, чтобы исполнитель одновременно выпустил две работы с одинаковым названием. Но не забывайте о формате. Очень часто бывает, что альбом выходит на компакт-дисках и кассетах. Таким образом, можно продолжить включать столбцы в первичный ключ. По сути, несложно привести разумные доводы в пользу того, почему ключом следует сделать всю совокупность столбцов. Это чересчур непрактично, поэтому мы воспользуемся полем-счетчиком.

Третий принцип требует, чтобы ячейка не содержала группу значений. Это может показаться странным, но представьте себе список разделенных запятыми строк в поле

типа VARCHAR. Например, в столбце Format может быть записано "78,LP". Это означает, что носитель представляет собой долгоиграющую пластинку на 78 оборотов. Такой формат хранения не оптимален, поскольку пропадает возможность осуществлять отбор записей отдельно по каждому из этих критериев. Попробуйте найти все пластинки в коллекции. Можно, конечно, применить оператор LIKE, описанный в главе 10, "Типы данных, переменные и выражения". В MySQL есть даже тип SET, обозначающий неупорядоченный набор значений, но его поддержка ограничена. Следует очень внимательно контролировать ситуации, когда в ячейку заносится несколько значений.

В листинге 8.2 демонстрируется добавление к таблице первичного ключа, реализованного в виде счетчика. Это разумное решение, поскольку большинство столбцов не зависит друг от друга. В данном примере это не отражено, но подразумевается, что в поле Format заносится одно значение. Если один и тот же альбом приобретается в разных форматах, для него создаются дополнительные записи.

```
CREATE TABLE recording (
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Artist VARCHAR(32) NOT NULL,
    Released DATE,
    Title VARCHAR(32) NOT NULL,
    Format VARCHAR(32) NOT NULL,
    Label VARCHAR(32),
    Genre VARCHAR(32),
    Length INT(11),
    Purchased DATE,
    Cost DECIMAL(11,2),
    CurrentValue DECIMAL(11,2),
    PRIMARY KEY(ID)
);
```

Теперь таблица находится в первой нормальной форме, но многие проблемы все еще не решены. Основная из них — дублирование информации в случае, когда один и тот же альбом доступен в разных форматах. Как минимум, имя исполнителя и название альбома будут повторяться.

Вторая нормальная форма

Вторая нормальная форма требует, чтобы все столбцы зависели от полного первичного ключа, а не от его частей. Таблица нарушает данное правило, если первичный ключ является составным и какого-то подмножества его столбцов достаточно для идентификации записей.

Во второй нормальной форме подразумевается использование составного первичного ключа. Таблица, в которой первичным ключом является один столбец, автоматически считается имеющей вторую нормальную форму.

Рассмотрим таблицу, представленную в листинге 8.3. В ней отслеживается количество часов, за которые разработчики, действующие по контракту, выставляют счета по тем или иным проектам. У каждого проекта есть числовой идентификатор, а у раз-

работчика — номер социального страхования (social security number, SSN). Идентификаторы обоих типов являются уникальными и назначаются правительством в целях упрощения налогообложения. Каждый разработчик трудится по фиксированной ставке (число долларов в час). Эта ставка зависит от вида выполняемой работы (столбец ConractorWork). В столбце Quantity указано количество часов, затраченных на выполнение проекта.

```
CREATE TABLE work (
    Project INT(11) NOT NULL,
    ProjectName VARCHAR(64),
    ContractorSSN VARCHAR(16) NOT NULL,
    ContractorName VARCHAR(64),
    ContractorWork VARCHAR(16),
    ContractorRate DECIMAL(6,2),
    Quantity DECIMAL(6,2),
    PRIMARY KEY(Project, ContractorSSN)
);
```

Эта таблица находится в первой нормальной форме, но не во второй. Приведем пример. Представьте двух разработчиков, занимающихся одним и тем же проектом. Соответственно, название проекта повторится дважды. Оно зависит только от идентификатора проекта, т.е. от половины первичного ключа. То же самое можно сказать о столбцах ContractorName и ContractorRate.

Чтобы устранить дублирование, необходимо перенести информацию о проектах и разработчиках в отдельные таблицы. В листинге 8.4 исходная таблица разбита на три составляющие. В таблице work содержатся три столбца: Project, ContractorSSN и Quantity. Первичный ключ тот же, что и прежде, но теперь столбцы Project и ContractorSSN являются внешними ключами. Название проекта хранится в таблице project, а имя разработчика и процентная ставка — в таблице contractor.

```
CREATE TABLE work (
    Project INT(11) NOT NULL,
    ContractorSSN VARCHAR(16) NOT NULL,
    Quantity DECIMAL(6,2),
    PRIMARY KEY(Project, ContractorSSN),
    FOREIGN KEY(Project) REFERENCES project(ID),
    FOREIGN KEY(ContractorSSN) REFERENCES contractor(SSN)
);

CREATE TABLE project (
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Name VARCHAR(64),
    PRIMARY KEY(ID)
); .

CREATE TABLE contractor (
```

```
    SSN VARCHAR(16) NOT NULL,
    Name VARCHAR(64),
    WorkType VARCHAR(16),
    Rate DECIMAL(6,2),
    PRIMARY KEY(SSN)
);
```

Всегда обращайте внимание на имена столбцов с общими префиксами. Очень часто это свидетельствует о необходимости создания дополнительной таблицы, как в данном случае.

На рис. 8.1 изображена схема отношений между тремя таблицами.

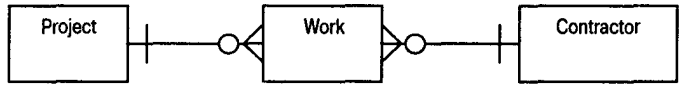


Рис. ***8.1.*** Диаграмма табличных связей

Третья нормальная форма

Вторая нормальная форма устраняет столбцы, зависящие от части первичного ключа. Третья нормальная форма устраняет столбцы, которые зависят от столбца, не являющегося первичным ключом. Это называется ***транзитивной зависимостью*** и ведет к ненужному дублированию данных.

Рассмотрим таблицу contractor в листинге 8.4. Столбец WorkType описывает вид выполняемой работы. Одни разработчики пишут программы на C, другие — готовят иллюстрации. Эти задачи оплачиваются по-разному. По сути, зная вид работы, можно определить процентную ставку. Правда, здесь применяется правило бизнес-логики: работа конкретного вида всегда оплачивается одинаково.

Решением проблемы снова будет разделение таблицы надвое (листинг 8.5). Вид работы и связанная с ним процентная ставка хранятся теперь в отдельной таблице. Обратите внимание на то, что строка с названием вида работы является первичным ключом таблицы `worktype`. Может показаться, что применение целочисленного идентификатора в качестве первичного ключа было бы логичнее, но это лишь приведет к возникновению транзитивной зависимости. Мы вернемся к рассмотрению данной ситуации при описании процесса **денормализации**.

```
CREATE TABLE contractor (
    SSN VARCHAR(16) NOT NULL,
    Name VARCHAR(64),
    WorkType VARCHAR(16) NOT NULL,
    PRIMARY KEY(SSN),
    FOREIGN KEY(WorkType) REFERENCES worktype(ID)
);

CREATE TABLE worktype (
    ID VARCHAR(16),
```

```
    Rate DECIMAL(6,2),
    PRIMARY KEY(ID)
);
```

Нормальная форма Бойса-Кодда

Первые три нормальные формы были описаны доктором Коддом в его основной работе, посвященной нормализации. Обычно третьей нормальной формы вполне достаточно для того, чтобы база данных считалась оптимизированной. Форма Бойса-Кодда решает более узкую проблему, когда часть составного первичного ключа зависит от значения другого столбца и этот столбец сам по себе не может быть первичным ключом.

В листинге 8.6 приведен пример таблицы, находящейся в третьей нормальной форме, но не в форме Бойса-Кодда. В таблице `software` отслеживаются служащие офиса, заказывающие программное обеспечение из архивов. Архивы расположены в разных частях офиса, а каждому служащему разрешен доступ только в один архив.

```
CREATE TABLE software (
    Cabinet INT(11) NOT NULL,
    Title VARCHAR(32) NOT NULL,
    Employee INT(11),
    CheckedOut ENUM('Y','N'),
    PRIMARY KEY(Cabinet, Title),
    FOREIGN KEY(Cabinet) REFERENCES cabinet(ID),
    FOREIGN KEY(Employee) REFERENCES employee(ID)
);

CREATE TABLE employee (
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Name VARCHAR(64),
    Cabinet INT(11) NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(Cabinet) REFERENCES cabinet(ID)
);
```

Таблица `software` не находится в форме Бойса-Кодда, потому что по идентификатору служащего можно определить архив. Обратите внимание на то, что столбца `title` недостаточно для идентификации записей, потому что в двух архивах могут находиться копии одной и той же программы. Точно так же один служащий может одновременно заказать несколько программ.

Решение, переводящее таблицу `software` в форму Бойса-Кодда, заключается в удалении из нее столбца Cabinet. Столбцов `Title` и Employee достаточно для однозначной идентификации записей. Новые определения таблиц приведены в листинге 8.7.

```
CREATE TABLE software (
    Title VARCHAR(32) NOT NULL,
    Employee INT(11),
    CheckedOut ENUM('Y','N'),
    PRIMARY KEY(Title, Employee),
    FOREIGN KEY(Employee) REFERENCES employee(ID)
);

CREATE TABLE employee (
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Name VARCHAR(64),
    Cabinet INT(11) NOT NULL,
    PRIMARY KEY(ID),
    FOREIGN KEY(Cabinet) REFERENCES cabinet(ID)
);
```

Четвертая нормальная форма

Четвертая нормальная форма также относится к узкому кругу случаев. Ее обычно применяют лишь к промежуточным таблицам, создаваемым для преобразования отношений "многие ко многим". Таблица нарушает требования четвертой нормальной формы, если служит мостом между более чем двумя таблицами.

Рассмотрим диаграмму на рис. 8.2. Программа работает с ресурсами двух видов: библиотеками функций и конфигурационными файлами. Это, в частности, справедливо для утилит MySQL. Большинство из них использует клиентскую библиотеку MySQL, а также библиотеку `stdio` языка C, храня при этом конфигурационные установки в файле `/etc/my.cnf`.

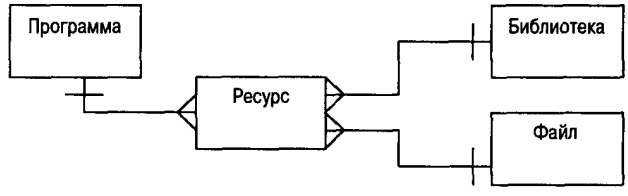


Рис. *8.2.* Многоцелевая промежуточная таблица

В листинге 8.8 описаны таблицы, соответствующие этой диаграмме.

```
CREATE TABLE program (
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Name VARCHAR(32),
    PRIMARY KEY(ID)
```

```
CREATE TABLE resource (
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Program INT(11) NOT NULL,
    Library INT (11),
    ConfigFile INT (11),
    PRIMARY KEY(ID),
    FOREIGN KEY(Program) REFERENCES program(ID),
    FOREIGN KEY(Library) REFERENCES library(ID),
    FOREIGN KEY(ConfigFile) REFERENCES configfile(ID)
);

CREATE TABLE library (
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Name VARCHAR(32),
    PRIMARY KEY(ID)
);

CREATE TABLE configfile (
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Name VARCHAR(32),
    PRIMARY KEY(ID)
);
```

Суть проблемы локализована в таблице resource и заключается в том, что в столбцах внешних ключей разрешены значения NULL. Это подразумевает их необязательность. С логической точки зрения необязательная связь между программой и ее ресурсами порождает неопределенность записей в таблице. Предположим, программа работает с одной библиотекой и одним файлом ресурсов. Тогда в таблицу следует добавить строку с соответствующими значениями в столбцах Library и ConfigFile. Но можно поступить иначе: добавить две записи, в которых одно из значений устанавливается явно, а другое равно NULL.

Когда программа перестает работать с библиотекой, возникает неприятная проблема. Внесение изменений в таблицу зависит от того, каким образом в нее добавлялись записи. Потребуется либо инструкция UPDATE, либо DELETE.

Решение опять состоит в разбивке таблицы resource на две части. Мы не будем это показывать, так как уже неоднократно проделывали то же самое. Таблицы, нарушающие четвертую нормальную форму, редки и, очевидно, не совсем естественны. При построении диаграмм с отношениями "многие ко многим" и преобразовании этих отношений в промежуточные таблицы несложно проконтролировать описанную ситуацию.

Денормализация

Следствием полной нормализации является переизбыток таблиц. Внешняя избыточность значений устранена, зато появилось множество значений, реализующих внутренние связи между таблицами. Кроме того, возникают дополнительные затраты на выполнение операций объединения, что влияет на производительность. Конечно, в 70-х годах **экономия** дискового пространства была оправданной, но сегодня большее значение имеет скорость вычислений.

Нормализация — это процесс логической группировки данных, направленный на то, чтобы приложению не нужно было заниматься синхронизацией дублирующихся значений. **Денормализация служит** противоположным целям ради повышения производительности.

К **денормализации** нельзя подходить легкомысленно. Выполняя **денормализацию,** вы берете на себя обязанность поддерживать целостность базы данных программным путем. В большинстве случаев на это пойти легко, потому что в приложениях есть определенные средства контроля целостности. Например, таблица адресов может содержать столбец с обозначениями штатов. Ему можно присвоить тип `CHAR(2)`. Приложение будет выдавать пользователям полные названия штатов, но в запросах указывать **двухбуквенные** коды. В распоряжении приложения будет внутренняя таблица соответствий между полными названиями и кодами. Это хорошее решение, поскольку в обозримом будущем в США не предвидятся внутриполитические реформации.

В листинге 8.9 показана **денормализация** таблиц, взятых из листинга 8.5. Там, как вы помните, они находились в третьей нормальной форме. На этот раз к таблице `worktype` добавляется первичный ключ, реализованный в виде поля-счетчика. В результате индекс будет состоять из целых чисел, а не строк, что приведет к существенному ускорению объединений. К тому же, значения первичного ключа вычисляются автоматически, что вряд ли приведет к нарушению целостности.

```
CREATE TABLE contractor (
    SSN VARCHAR(16) NOT NULL,
    Name VARCHAR(64),
    WorkType INT(11)
    PRIMARY KEY(SSN),
    FOREIGN KEY(WorkType) REFERENCES worktype(ID)
);

CREATE TABLE worktype (
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Name VARCHAR(16) NOT NULL,
    Rate DECIMAL(6,2),
    PRIMARY KEY(ID)
);
```

Объединения, включающие более трех таблиц, сложнее выполнять человеку, чем СУБД. Не стоит недооценивать способность MySQL эффективно выполнять объединения при наличии правильного набора индексов. Крупные объединения можно оптимизировать путем добавления индексов или изменения порядка таблиц в инструкции SELECT. Это сложно и утомительно, но зато это можно сделать один раз, после чего программа всегда будет работать оптимальным образом.

Во многих СУБД поддерживаются подчиненные запросы: инструкции SELECT, размещаемые в скобках. В MySQL понятие подчиненного запроса отсутствует, но есть временные таблицы. Можно выполнить часть объединения и поместить его результат во временную таблицу. В главе 13, "Инструкции SQL", приводится описание инструкции CREATE TABLE, где рассказывается о том, как создать таблицу по результатам работы инструкции SELECT.

ТРАНЗАКЦИИ И ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ

В этой главе.

Параллельные запросы
Транзакции
Блокировки
Последовательности

Глава 9

Представьте себе базу данных компании среднего размера, занимающейся Web-коммерцией. В любой момент времени на Web-узле компании могут находиться до пятидесяти посетителей. Для загрузки каждой страницы требуется один или несколько запросов. Web-приложение должно собрать содержимое "корзины" пользователя, а затем получить информацию о просматриваемом товаре. Иногда два пользователя могут одновременно выполнять запросы, и система должна обслуживать их независимо друг от друга. Это называется *параллелизмом*.

Отсутствие контроля над параллельным выполнением операций может привести к искажению данных. Если два или более потоков начнут осуществлять запись в один и тот же файл, результат будет непредсказуем. Решение проблемы состоит в том, чтобы в тот или иной момент времени только один поток мог обращаться к файлу. Web-**приложения,** хранящие данные в обычных файлах, при обращении к файлу блокируют его с помощью системных вызовов.

В реляционных СУБД блокировки реализованы на более абстрактном уровне в виде транзакций. Под транзакцией понимается последовательность SQL-инструкций, выполняемая полностью или не выполняемая вообще. Сервер изолирует **одновременные** потоки друг от друга, ограничивая их доступ к модифицируемым данным. По мере завершения транзакций сервер пытается выполнять их так, будто они происходят последовательно, а не паралле льно.

Транзакции являются относительно новым понятием в MySQL. Их можно применять лишь к таблицам определенных типов. В то же время в MySQL поддерж иваются табличные блокировки и другие функциональные возможности, позволяющие имитировать транзакции.

Параллельные запросы

Параллелизм — это сложная проблема для СУБД. MySQL является многопотоковой программой, поэтому вынуждена справляться с множественными запросами на подключение. Но помимо проблемы планирования возникает еще и проблема одновременного доступа к данным.

Представим себе двух менеджеров, пытающихся выяснить количество единиц одного и того же товара на складе. Первый менеджер вводит инструкцию SELECT и определяет, что на складе осталось 150 единиц. Второй менеджер получает те же самые данные. Первый менеджер оформляет покупку 30-ти единиц товара, поэтому он вводит инструкцию UPDATE, устанавливая объем складского запаса равным 120-ти единицам. А тем временем второй менеджер, ничего не подозревая, оформляет аналогичный заказ на 10 единиц и тоже вводит инструкцию UPDATE. Он думает, что уменьшает количество товара на 10 единиц, а на самом деле — увеличивает на 20. В результате в базу данных вносится недостоверная информация.

Программисты решают проблему параллельного доступа с помощью блокировок. Имеется центральный системный сервис, отслеживающий блокировки ресурсов и контролирующий работу потоков. Если какой-то поток захватил ресурс, поставив на него блокировку, все остальные потоки, обращающиеся к тому же самому ресурсу, вынуждены ждать его освобождения. Это может приводить к ощутимому снижению производительности, поэтому вводятся блокировки разных уровней, позволяющие точнее определять область действия допустимых и недопустимых операций.

В MySQL блокировки реализуются без прямого вмешательства со стороны пользователей. Одиночные запросы выполняются в атомарном режиме, в котором каждый запрос представляет собой отдельную транзакцию. Таким образом, описанную выше проблему несогласованного заказа можно решить, объединив несколько операций в одном запросе, например:

```
UPDATE item
   SET Inventory = Inventory - 30
 WHERE ID = 3
```

Не всякую последовательность операций можно выразить в виде одного запроса. Скажем, для обновления двух таблиц необходимы два запроса. В таких случаях нужно явно указывать начало и конец каждой транзакции.

Транзакции

Транзакция — это совокупность одной или нескольких SQL-инструкций, имеющая начало и конец. В конце транзакции происходит либо ее отмена, либо завершение. Отмена транзакции называется *откатом* (rollback), так как происходит последовательная отмена всех сделанных изменений. Завершение транзакции называется *фиксацией* (commit).

Считается, что правильная транзакция обладает следующими свойствами: атомарностью, согласованностью, изолированностью и устойчивостью. Под *атомарностью* понимают принцип неделимости: все инструкции, составляющие транзакцию, обязательно выполняются, иначе не выполняется ни одна из них. Никаких промежуточных состояний не существует. База данных поддерживает внутреннюю согласованность за

счет *сериализации* транзакций. Несмотря на кажущуюся одновременность событий, результаты транзакций заносятся в базу данных последовательно. Все транзакции изолируются друг от друга, а изменения, вызываемые каждой из них, становятся доступными лишь после завершения транзакции. Свойство *устойчивости* означает, что при успешном завершении транзакции в базу данных вносятся постоянные, неотменяемые изменения. Устойчивая база данных способна выдержать внезапную аварию, например сбой питания, и при этом остаться согласованной.

Транзакции реализуются путем ведения журнала всех изменений, вносимых в базу данных в ходе каждой транзакции. Когда происходит откат, СУБД сверяется с журналом и отменяет все изменения. По журналу легко можно восстановить согласованное состояние базы данных в случае сбоя. Ведение журнала транзакций приводит к снижению производительности, поэтому в MySQL для таблиц стандартного типа — MyISAM — транзакции не поддерживаются. Это одна из причин столь высокой скорости работы программы.

Транзакции появились в MySQL сравнительно недавно. Они поддерживаются для таблиц расширенных типов, таких как InnoDB, Berkeley DB и Gemini. Однако следует отметить, что во многих ситуациях транзакции не нужны, так как табличных блокировок будет более чем достаточно. В отличие от других СУБД, MySQL предоставляет пользователям право выбора: можно работать с более медленными таблицами, поддерживающими транзакции, или с более быстрыми таблицами, где транзакции недопустимы.

В крупных банковских базах данных транзакции, несомненно, необходимы. В подобных системах множество параллельных потоков может соперничать за право доступа к ресурсам, а потеря данных обходится слишком дорого. С другой стороны, Web-форумы, к примеру, не требуют столь тщательного управления, поскольку стоимость потери сообщения невысока. Конечно, активность посетителей форума может сравниться с активностью клиентов небольшого банка, но ведь форумы не приносят такого дохода, поэтому приобретение дополнительного оборудования для обработки транзакций является неоправданным. В основном работа с базой данных форума сводится к вставке новых сообщений и чтению существующих. Достаточно использовать первичные ключи, являющиеся автоматическими счетчиками.

Internet-магазину требуется такая же целостность базы данных, как и в банке, только интенсивность трафика заметно ниже. В этой ситуации можно воспользоваться табличными блокировками. Когда программа создает заказ, она блокирует определенное число таблиц. В результате никакой другой клиент не сможет завершить свой заказ в это же самое время. Но в случае небольшого магазина, выписывающего порядка сотни счетов в день, весьма маловероятно, чтобы два клиента сделали заказы одновременно. По мере роста активности посетителей производительность системы снижается, зато целостность остается неизменной. Таблицы, незаблокированные в ходе процедуры заказа, остаются нетронутыми. Например, незачем блокировать доступ к каталогу предлагаемых товаров. Как следствие, клиентам, помещающим товары в свои корзины, не приходится дожидаться разблокирования таблицы заказов.

Уровни изоляции

Теоретически СУБД должна обеспечивать полную изоляцию транзакций. На практике же вводится несколько уровней изоляции, самый высокий из которых соответствует полной изолированности. В MySQL выбор уровня осуществляется с помощью инструкции SETTRANSACTION, подробно описанной в главе 13, "Инструкции SQL".

В стандарте языка SQL определены четыре уровня изоляции: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ и SERIALIZABLE. Они перечислены в порядке от наименьшей к наивысшей изолированности и по очереди решают три основные проблемы, возникающие при использовании транзакций: появление пром ежуточных данных, появление несогласованных данных и появление строк-призраков.

Промежуточные данные появляются, когда незавершенная транзакция модифицирует строку таблицы, а другая транзакция читает эту строку. Если первая транзакция будет отменена (произойдет откат), то окажется, что вторая транзакция получила данные, которые официально никогда не существовали. Такое поведение проявляется на уровне READ UNCOMMITTED.

В режиме READ COMMITTED транзакциям разрешено читать только подтвержденные данные. Однако это не решает проблему несогласованных данных. Предположим, в ходе транзакции вводится запрос, определяющий число записей в таблице. По завершении этого запроса другая транзакция, работающая с той же самой таблицей, удаляет или добавляет записи. Если теперь первая транзакция повторно выполнит свой запрос, она получит другие результаты.

Проблема несогласованных данных решается на уровне REPEATABLE READ. В этом режиме строки, к которым транзакция обращается для чтения или записи, блокируются. Но и здесь есть проблема: строки-призраки. Транзакция может заблокировать все записи, с которыми ведется работа, но она не может помешать другой транзакции добавить строки в ту же самую таблицу. Если в ходе транзакции вводятся два запроса на выборку, а между ними вторая транзакция добавляет в таблицу новую строку, эта строка станет "фантомом", так как она внезапно появляется в ходе одной и той же транзакции.

В режиме SERIALIZABLE транзакции принудительно выполняются одна задругой. Именно такое поведение рекомендуется в стандарте SQL. Если две транзакции попытаются обновить одну и ту же строку, одна из них будет объявлена проигравшей в тупиковой ситуации и как следствие — отменена.

Естественно, на любом уровнениже SERIALIZABLE появляется риск нарушения целостности базы данных. Как уже поминалось выше, в некоторых ситуациях такой риск вполне допустим.

Выполнение транзакций

По умолчанию программа MySQL работает в режиме автоматического завершения транзакций. Каждый запрос представляет собой транзакцию, которая после получения результатов запроса немедленно завершается. Есть два способа изменить такое поведение: можно отключить режим автозавершения с помощью инструкции SET либо воспользоваться инструкцией BEGIN. В первом случае каждый последующий запрос будет входить в неявную транзакцию. Во втором случае инструкция BEGIN помечает начало транзакции.

Транзакции применимы лишь к некоторым типам таблиц. Для стандартного типа MyISAM транзакции не поддерживаются. Попытка обновить таблицу типа MyISAM в рамках транзакции просто приведет к немедленному обновлению таблицы. Это изменение уже нельзя будет отменить. Если транзакции используются довольно часто, можно изменить стандартный тип таблиц, отредактировав файл конфигурации MySQL, как описано в главе 14, "Утилиты командной строки".

Транзакция отменяется с помощью инструкции ROLLBACK. Если же необходимо принять изменения, воспользуйтесь инструкцией COMMIT. В листинге 9.1 показана простая транзакция, завершающаяся откатом.

```
mysql> BEGIN;
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT *
    ->   FROM config;
+      +       +
 \ name \ value \
+      +       +
 \ seed \ 12345 \
+      +       +
1 row in set  (0.00 sec)

mysql> SELECT @seed:=Value
    ->   FROM config
    ->  WHERE Name='seed';
+             -+
 \ @seed:=Value \
+              +
 \        12345 \
+              +
1 row in set  (0.01 sec)

mysql> UPDATE config
    ->    SET Value = RAND(@seed)*100000;
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT *
    ->   FROM config;
+      +       +
 \ name \ value \
+      +       +
 \ seed \ 18113 \
+      +       +
1 row in set  (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT *
    ->   FROM config;
```

```
+------+-------+
| name | value |
+------+-------+
| seed | 12345 |
+------+-------+
1 row in set (0.00 sec)
```

В транзакциях разрешается обновлять таблицы, но только с помощью инструкций INSERT и UPDATE. Изменения схемы базы данных выполняются вне транзакций. Когда вводится инструкция, вносящая изменение в схему, текущая транзакция тут же завершается. К завершению транзакции приводят следующие инструкции: ALTER TABLE, BEGIN, CREATE INDEX, DROP DATABASE, DROP TABLE, RENAME TABLE, TRUNCATE.

Блокировки

Блокировки — это механизм, применяемый в MySQL для реализации транзакций и обеспечения одновременного доступа к данным. Помимо этого можно явно запрашивать блокирование таблиц с помощью инструкции LOCK TABLES. Есть также функция GET_LOCK(), позволяющая создавать произвольные блокировки. Блокировки работают с любыми таблицами, даже теми, которые не поддерживают транзакции.

На внутреннем уровне программа MySQL блокирует таблицы целиком в случае необходимости. Допускается также блокировать строки, столбцы и страницы (под страницей понимается произвольный блок данных, связанных с таблицей). С точки зрения производительности преимущество той или иной модели проявляется по-разному. Табличное блокирование выгодно для Web-приложений. Блокирование на уровне строк лучше подходит для баз данных, в которых часто происходят откаты.

Инструкция LOCK TABLES описана в главе 13, "Инструкции SQL". С ее помощью можно ставить жесткую и нежесткую блокировки на одну или несколько таблиц. Нежесткая блокировка позволяет потокам осуществлять одновременное чтение данных из таблицы. Жесткая блокировка означает монопольный доступ к таблице со стороны одного-единственного потока. Инструкция UNLOCK TABLES снимает табличные блокировки.

Имитировать блокировку на уровне строк можно путем добавления к таблице специального столбца. Ячейки этого столбца будут иметь два состояния: заблокировано и незаблокировано. Тип столбца можно задать как SMALLINT или, лучше, ENUM (подробнее об этих типах рассказывается в главе 11, "Типы столбцов и индексов").

В листинге 9.2 приведено описание таблицы, посредством которой контролируется работа водяных насосов на ферме. Насосы расположены в разных местах, и каждый из них можно настроить на прокачку определенного объема воды в час. Программа, написанная на языке C, проверяет таблицу и посылает насосам команды по беспроводной сети. Если программе требуется изменить мощность насоса, она сначала пытается заблокировать строку, устанавливая значение столбца RowLock. В листинге 9.3 показана инструкция, блокирующая строку с идентификатором 2.

```
CREATE TABLE pump (
    ID INT NOT NULL AUTO_INCREMENT,
    Location INT NOT NULL,
    Volume FLOAT(5,2),
    RowLock ENUM('UNLOCKED', 'LOCKED') NOT NULL,
    PRIMARY KEY(ID)
```

```
UPDATE pump
   SET RowLock = 'LOCKED'
 WHERE ID = 2;
```

Когда СУБД MySQL попытается выполнить эту **инструкцию**, может оказаться, что столбец RowLock уже содержит значение LOCKED. MySQL никогда не обновляет строки, если это не приводит к изменению содержащихся в них значений. Следовательно, попытка заблокировать строку, которая до этого уже была кем-то заблокирована, ни к чему не приведет. MySQL сообщит программе о том, что изменению подверглись нуль строк, и программа поймет: со строкой работает другое приложение.

Функции GET_LOCK() и RELEASE_LOCK() реализуют другой механизм блокирования. Эти блокировки не связаны с какими-либо ресурсами и не контролируются самой СУБД. Поэтому их называют программными блокировками, т.е. их контроль должен осуществляться программным путем. У каждой такой блокировки есть имя, и в конкретный момент времени поток может ставить только одну программную блокировку.

С помощью указанных функций можно создавать блокировки произвольного уровня детализации. Например, перед каждым обновлением строки можно запрашивать блокировку, имя которой будет состоять из имени таблицы и значения первичного ключа. В случае обновления отдельной ячейки добавляется имя столбца.

Рассмотрим листинг 9.4. В первой инструкции запрашивается блокировка ячейки Price строки с идентификатором 3 в таблице item. Обратите внимание на то, что имя блокировки выбрано произвольно. Суть механизма в том, что все приложения придерживаются единого правила именования блокировок. Здесь нет тех издержек, которые свойственны транзакциям, хотя преимущества, по сути, те же самые.

```
SELECT GET_LOCK('Price on item.ID=3', 60);
UPDATE item SET Price=3.15 WHERE ID=3;
SELECT RELEASE LOCK('Price on item.ID=3');
```

Последовательности

Последовательность – это специальная конструкция, доступная в некоторых реляционных СУБД, включая Oracle. Она представляет собой счетчик, используемый для создания уникальных числовых идентификаторов. Текущее значение счетчика можно извлечь с помощью инструкции SELECT. Это происходит в атомарном режиме, т.е. гарантируется, что никакие два потока не получат два одинаковых идентификатора.

В MySQL уникальные идентификаторы строк лучше всего реализуются с помощью первичных **ключей-счетчиков**. Имитация последовательностей может потребоваться при переносе приложений в MySQL. Для этого необходимо создать таблицу, состоящую из одной ячейки. Таблица будет содержать один целочисленный столбец с единственным **значением** – начальным числом последовательности. Работа с **последовательностями** ведется посредством функции LAST_INSERT_ID(), описанной в главе 12, "Встроенные функции". Будучи вызванной без аргументов, она возвращает последнее значение счетчика, установленное путем **автоинкрементирования** или же самой функцией. *Если* вызвать функцию с аргументом, она вернетзначение аргумента.

Чтобы обновить значение счетчика последовательности, нужно вызвать функцию LAST_INSERT_ID(),передавейтекущеезначениесчетчика,увеличенноенаединицу. Это обновление выполняется атомарно, т.е. другие потоки не могут ему помешать. Пример работы с последовательностями показан в листинге 9.5.

```
mysql> CREATE TABLE seq (
    ->     nextval INT
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO seq VALUES (100);
Query OK, 1 row affected (0.00 sec)

mysql> UPDATE seq
    ->    SET nextval = LAST_INSERT_ID(nextval+1);
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT LAST_INSERT_ID();
+                  +
I last_insert_id() |
+                  +
I           101    |
+                  +
1 row in set (0.00 sec)

mysql> UPDATE seq
    ->    SET nextval = LAST_INSERT_ID(nextval+1);
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT LAST_INSERT_ID();
+                  +
```

```
| last_insert_id() |
+------------------+
|              102 |
+------------------+
1 row in set (0.00 sec)
```

Не пытайтесь извлекать значение счетчика последовательности напрямую из таблицы: другой поток мог изменить это значение.

Часть II

СПРАВОЧНИК MYSQL

та часть книги представляет собой справочник по программе MySQL. В главе 10, "Типы данных, переменные и выражения", рассматриваются поддерживаемые в MySQL типы данных, рассказывается о принципах составления выражений и об использовании переменных.

В главе 11, "Типы столбцов и индексов", речь пойдет о типах столбцов и индексов таблиц. Оказывается, в столбцах таблиц могут храниться не только числа и строки, но также значения даты/времени и даже большие двоичные объекты.

В главе 12, "Встроенные функции", описаны все функции MySQL. Эти функции могут включаться в SQL-инструкции с целью манипулирования столбцами, переменными и литералами. В главе 13, "Инструкции SQL", аналогичное описание дано для SQL-инструкций. В MySQL поддерживаются не только стандартные инструкции языка, но и целый ряд собственных расширений.

В главе 14, "Утилиты командной строки", содержится обзор утилит, входящих в дистрибутив MySQL. В главе 15, "Библиотека функций языка C", рассматриваются структуры и функции библиотеки языка C, с помощью которых можно писать программы, напрямую взаимодействующие с СУБД MySQL. Эта библиотека используется всеми утилитами MySQL, напрямую или посредством языков сценариев.

Обычно авторы помещают справочные материалы в конец книги. Я же вставил их в середину, поскольку надеюсь, что большую часть времени книга будет лежать открытой у вас на столе или на коленях. Чаще всего читатели пользуются именно справочником, и чтобы его было удобно раскрывать на нужной странице, он должен находиться посередине.

ТИПЫ ДАННЫХ, ПЕРЕМЕННЫЕ И ВЫРАЖЕНИЯ

В этой главе.

Глава 10

В этой главе рассматриваются основные типы данных, которым могут принадлежать литералы и переменные. Описываются также операторы, применяемые к значениям базовых типов, и выражения, формируемые из цепочек операторов.

Типы данных

В MySQL используются два фундаментальных типа данных: строки и числа. Значения этих типов заносятся в таблицы, и на основании типа данных столбца определяются правила его сравнения с другими столбцами.

Строки

Строки — это последовательности символов произвольной длины. От ключевых слов SQL они отделяются кавычками. Стандарт SQL требует использовать одинарные кавычки (`'`), но в MySQL, как и в других СУБД, вполне допускаются и двойные кавычки (`"`). В листинге 10.1 показано использование строк в качестве метки столбца и в операторе сравнения `LIKE`.

```
mysql> SELECT Db AS 'Leon''s Database', User
    ->   FROM db
    -> WHERE Host LIKE '%\%';
+-----------------+------+
| Leon's Database | User |
+-----------------+------+
| test            |      |
| test\_%         |      |
+-----------------+------+
2 rows in set (0.01 sec)
```

Поскольку некоторые символы имеют специальное назначение, для включения их в строку необходимо применять управляющие последовательности. Как правило, обратная косая черта заставляет синтаксический анализатор проигнорировать следующий символ. Есть также ряд управляющих последовательностей, начинающихся с обратной косой черты и заменяющих собой непечатаемые символы наподобие символа новой строки (\n). Однако вполне допускается вводить символы новой строки и табуляции нажатием клавиш <Enter> и **<Tab>** соответственно. Оба варианта продемонстрированы в табл. 10.1. Обратите внимание на то, что в обоих случаях строка "Line One Line Two" разбивается на две части.

С помощью управляющей последовательности \n

```
mysql> SELECT "Line One\nLine Two";
+----------+
| Line One
Line Two |
+----------+
| Line One
Line Two |
+----------+
1 row in set (0.00 sec)
```

Путем нажатия клавиши <Enter>

```
mysql> SELECT "Line One
"> Line TWO";
+----------+
| Line One
Line Two |
+----------+
| Line One
Line Two |
+----------+
1 row in set (0.00 sec)
```

Если одним из символов строки является кавычка, то в соответствии со стандартом SQL ее нужно удвоить. В MySQL то же самое применимо и в отношении двойных кавычек.

При работе со строками важно помнить о таком символе, как NUL (ASCII-код — 0). Он обозначает конец строки в C++ — языке, на котором написана программа MySQL. Включение его в середину строки может привести к непредсказуемым последствиям.

Допустимые управляющие последовательности перечислены в табл. 10.2.

Код	*Описание*
\'" *или* ""	Позволяет поставить двойные кавычки внутри строки, которая сама заключена в двойные кавычки
\' или ''	Позволяет поставить одинарную кавычку внутри строки, заключенной в одинарные кавычки
\%	Используется в выражениях оператора LIKE для отмены специального назначения символа %
\0	Соответствует символу NUL (ASCII-код 0)
\b	Соответствует символу возврата на одну позицию ("забой")
\n	Соответствует символу перевода строки (ASCII-код 10)
\r	Соответствует символу возврата каретки (ASCII-код 13)

\t	Соответствует символу горизонтальной табуляции (ASCII-код 11)
\\	Отменяет специальное назначение символа \
_	Используется в выражениях оператора LIKE для отмены специального назначения символа

Те, кто знакомы сязыком C, могут подумать, что вместо управляющих последовательностей можно использовать шестнадцатеричные ASCII-коды. Но это не так. Программа MySQL воспринимает только те коды, которые перечислены в табл. 10.2.

Допускается запись строк в шестнадцатеричном виде. Такие строки не нужно брать в кавычки. Нужно лишь добавить к строке префикс Ox (большая буква X интерпретируется неправильно). Пример такой строки показан в листинге 10.2. В MySQL версии 4.0 будет поддерживаться форма записи шестнадцатеричных строк, соответствующая стандарту ANSI: прописная литера X, за которой следует строка шестнадцатеричных символов, заключенная в одинарные кавычки, например X'4CG5GFGE'.

```
mysql> SELECT Ox4C656F6E AS 'Hex String';
+------------+
| Hex String |
+------------+
| Leon       |
+------------+
1 row in set (0.00 sec)
```

Числа

Числовые литералы записываются в виде цепочек цифр без каких-либо кавычек. Числу может предшествовать знак минус, а если это дробное число, то оно будет содержать десятичную точку. Максимальная разрядность целых чисел — 64 бита, т.е. числа, большие чем 2^{64}, будут представляться неправильно.

MySQL понимает научную запись дробныхчисел. Это означает, что после мантиссы числа можно указывать символ экспоненты и показатель степени. Например, запись 1.2e+3 соответствует числу 1200 $(1{,}2\times10^{3})$. Чтобы не возникало неоднозначности, показателю степени должен предшествовать знак "плюс" или "минус".

Шестнадцатеричные литералы, участвующие в числовых операциях, будут интерпретироваться как числа. Для этого они приводятся к целочисленному типу (листинг 10.3).

```
mysql> SELECT 0x01 + 1;
+----------+
| 0x01 + 1 |
+----------+
```

```
|         2 |
+-----------+
1 row in set (0.00 sec)
```

Значения NULL

В язык SQL введено понятие отсутствующего значения, записываемого как NULL. Это не то же самое, что 0 или пустая строка. Если одним из операндов выражения является значение NULL, результат также будет равен NULL. Отсюда следует, что проверку на равенство значению NULL необходимо выполнять с помощью специальных операторов IS NULL или <=>либо функции ISNULL(), но не стандартного оператора =.

Переменные

Переменные — это символ ические имена, ссылающиеся на изменяемые значения. Имя переменной может состоять из чисел, букв, знаков подчеркивания и доллара, а также точек. К нему всегда добавляется символ @, чтобы не возникала путаница с именами столбцов и ключевыми словами.

В MySQL переменные не требуют инициализации. Если происходит обращение к неопределенной переменной, ей присваивается значение NULL. По окончании сеанса переменные автоматически удаляются. Об управлении памятью можно не заботиться, но необходимо помнить отом, что приложение способно формировать пул соединений. В этом случае переменная будет оставаться в памяти до завершения всей группы сеансов.

Переменные могут использоваться везде, где допускается имя столбца, в том числе в предложениях SELECT и WHERE. Соответствующий пример показан в листинге 10.4. Запрещается указывать переменные в предложениях LIMIT и IGNORE, а также использовать переменные для именования столбцов.

```
mysql> SET @Special_Item = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT ID, Name
    ->   FROM item
    ->  WHERE ID = @Special_Item;
+----+------+
| ID | Name |
+----+------+
|  2 | Comb |
+----+------+
1 row in set (0.00 sec)
```

Программа MySQL назначает переменной тип лишь в момент инициализации, поэтому нужно внимательно следить за соответствием типов. Даже если переменная содержит число, но ее тип был определен как строковый, в ходе числовых операций потребуется выполнять преобразование типов, что может сказаться на производительности.

Разрешается присваивать переменной значение прямо в инструкции SELECT. Для этого предназначен оператор `:=`. Правым операндом может быть произвольное выражение, допустимое в списке возвращаемых столбцов. Эта особенность программы демонстрируется в листинге 10.5. Обратите внимание на то, что строка выражения стала надписью столбца в таблице результатов.

```
mysql> SELECT @Special_Item := ID
    ->   FROM item
    ->  WHERE Name = 'Brush';
+                      +
| @Special_Item := ID  |
+                      +
|                    3 |
+                      +
1 row in set (0.00 sec)

mysql> SELECT ID, Name
    ->   FROM item
    ->  WHERE ID = @Special_Item;
+    +        +
| ID | Name   |
+    +        +
|  3 | Brush  |
+    +        +
1 row in set (0.00 sec)
```

В показанном примере условию отбора соответствует одна строка. Если же запрос извлекает несколько строк, то столько же раз будет меняться значение переменной. Операция присваивания в предложении SELECT **выполняется тогда, когда строка форматируется и посылается клиенту. Следовательно, в предложении** WHERE **будет использовано то значение переменной, которое записалось в нее последним.**

Операторы

Операторы — это знаки записи математических и логических операций. Большинство операторов бинарно: один операнд записывается слева и один — справа. Существуют также унарные операторы, принимающие только один операнд.

За исключением оператора присваивания, все остальные операторы делятся на четыре категории: арифметические, реляционные (операторы сравнения), логические и побитовые. Арифметические операторы выполняют простейшие математические операции. Реляционные операторы предназначены для сравнения значений. С помощью логических операторов строятся булевы выражения. Побитовые операторы работают с битовым представлением чисел.

Если оба операнда принадлежат к одному и тому же типу, тип результата определяется тривиально. Строки, участвующие в арифметических выражениях, преобразуются в числа. Несовпадение типов в операциях сравнения решается следующим об-

разом: строки преобразуются в числа, целые числа — в числа с плавающей запятой, а числа — в значения даты/времени.

Если оба операнда равны NULL, результатом выражения тоже будет NULL. В случае булевых выражений значению NULL соответствует значение FALSE. Исключение составляет оператор <=>.

Арифметические операторы

К арифметическим операциям относятся сложение, вычитание, умножение и деление. Добавление к числу знака "минус" называется отрицанием. Операция деления по модулю возвращает остаток от целочисленного деления двух чисел. Существующие арифметические операторы перечислены в табл. 10.3.

Оператор	*Операция*
+	Сложение
-	Вычитание и унарное отрицание
*	Умножение
/	Деление
%	Деление по модулю

Арифметическими операндами могут быть только числа. В некоторых языках программирования поддерживается идея сложения строк, но в MySQL строки, входящие в состав арифметических выражений, будут преобразованы в числа. Если корректное преобразование невозможно, значением строки будет число 0. В случае необходимости целые числа будут преобразованы в числа с плавающей запятой, как показано в листинге 10.6.

```
mysql> SELECT 5+7, 9-4, 2*3, 10/3, 10%3;
+-----+-----+-----+------+------+
| 5+7 | 9-4 | 2*3 | 10/3 | 10%3 |
+-----+-----+-----+------+------+
|  12 |   5 |   6 | 3.33 |    1 |
+-----+-----+-----+------+------+
1 row in set (0.00 sec)
```

Оператор / возвращает число с плавающей запятой, точность которого на две цифры выше, чем количество цифр после запятой у самого точного из операндов. Например, результат операции 10/3 будет равен 3.33, а операции 10/3.0 — 3.333.

Оператор унарного отрицания (-) меняет знак литерала или переменной. С помощью оператора + можно указать на то, что значение является положительным.

Операторы сравнения

Реляционные операторы, иначе называемые операторами сравнения (табл. 10.4), возвращают булевы значения TRUE (истина) или FALSE (ложь). Если булево значение стоит в списке возвращаемых столбцов или в правой части оператора присваивания, то оно будет преобразовано в 1 (TRUE) или О (FALSE).

Оператор	*Проверка*
<	Меньше
>	Больше
<=	Меньше или равно (не больше)
>=	Больше или равно (не меньше)
BETWEEN *минимум* AND *максимум*	Принадлежность диапазону
IN (...)	Членство в множестве
NOT IN (...)	Отсутствие членства в множестве
=	Равно
<=>	Не равно (допускается сравнение со значением NULL)
LIKE	Соответствие шаблону
NOT LIKE	Несоответствие шаблону
REGEXP, RLIKE	Соответствие регулярному выражению
NOT REGEXP, NOT RLIKE	Несоответствие регулярному выражению
!=, <>	Не равно
IS NULL	Равно NULL
IS NOT NULL	Не равно NULL

Строка, сравниваемая с числом, сама будет преобразована в число. Если корректное преобразование невозможно, вместо строки будет подставлено значение 0. Целые числа, сравниваемые с числами с плавающей запятой, будут приведены к соответствующему типу.

Если в операции сравнения участвует значение **даты/времени**, программа MySQL попытается привести к этому типу второй операнд. Такое преобразование возможно для строки или числа, состоящих из восьми цифр. Первые четыре цифры считаются номером года, следующие две — номером месяца, а последние две — номером дня. В случае, когда операнд представлен в другом формате, произойдет противоположное преобразование: значения даты/времени — в строку или число. Столбцы типов TIME и DATE всегда сравниваются в строковом виде.

По умолчанию строки сравниваются в алфавитном порядке без учета регистра. Точный порядок сравнения определяется набором символов, который используется

по умолчанию. Ключевое слово BINARY заставляет учитывать регистр при сортировке строк.

Оператор BETWEEN является упрощенным вариантом комбинации операторов >= и <=. Таким образом, инструкции, показанные в листинге 10.7, являются эквивалентными.

```
SELECT SKU, ListPrice
  FROM invoice_sku
WHERE ListPrice >= 50
  AND ListPrice<= 100
```

```
SELECT SKU, ListPrice
  FROM invoice_sku
 WHERE ListPrice BETWEEN 50 AND 100
```

Операторы IN и NOT IN принимают в скобках группу разделенных запятыми значений, задающих допустимое множество. Если левый операнд совпадает с одним из значений в скобках, результат проверки будет истинным. В списке должны быть указаны литералы или имена столбцов. В отличие от некоторых СУБД, в MySQL не допускается указывать в скобках запрос. Подчиненные запросы появятся в MySQL версии 4.1.

Значения, указанные в скобках, приводятся к типу левого операнда. Если, например, целое число сравнивается с группой чисел с плавающей запятой, то последние будут округлены. Это означает, что проверка 5 IN (3.2, 4.1, 4.9) окажется истинной, а проверка 5.0 IN (3.2, 4.1, 4.9) — нет.

Оператор = проверяет равенство операндов, а операторы != и <> — их неравенство. Если любой из операндов равен NULL, то результат проверки также будет равен NULL, а не TRUE или FALSE, как можно предположить. К примеру, инструкция

```
SELECT 17 = NULL
```

вернет NULL, а не 0. Инструкция

```
SELECT NULL = NULL
```

тоже вернет NULL. Чтобы избежать такого поведения, пользуйтесь оператором <=>, специфичным для MySQL. Результат инструкции

```
SELECT NULL <=> NULL
```

равен 1. Аналогичные проверки выполняют операторы IS NULL и IS NOT NULL.

Операторы LIKE и NOT LIKE сравнивают левый операнд с шаблоном, указанным в правом операнде. Метасимвол % в шаблоне соответствует произвольному числу символов, а метасимвол _ — одиночному символу. Остальные символы воспринимаются буквально. Если необходимо выполнить сравнение с самим символом % или _, его нужно защитить от интерпретации с помощью обратной косой черты.

Строка шаблона анализируется дважды. При этом на первом проходе последовательности \%, _ и \\ будут заменены соответствующими литералами, а вот остальные управляющие последовательности не распознаются, поэтому \n превратится в n, а не в символ новой строки. Проверка обычных управляющих последовательностей выполняется на втором проходе. Это означает, что в данном случае символы обратной косой черты необходимо удваивать. Таким образом, чтобы вставить в шаблон символ табуляции, следует записать \\\t. Обратной косой черте соответствует запись \\\\.

Всего этого можно избежать, если воспользоваться предложением ESCAPE, которое задает символ, служащий началом управляющих последовательностей в данном конкретном операторе LIKE. В листинге 10.8 приведены примеры выражений с оператором LIKE, каждое из которых является истинным. Выражения с оператором LIKE чувствительны к регистру символов только в том случае, когда левый операнд помечен ключевым словом BINARY.

```
SELECT 'a\tb' LIKE a%';
SELECT 'a\tb' LIKE  %b';
SELECT 'a\tb' LIKE  a_b';
SELECT 'a\\b' LIKE  %\\\\%';
SELECT 'a\tb' LIKE  %^t%' ESCAPE '^';
```

Оператор REGEXP сравнивает левый операнд с регулярным выражением, стоящим справа. В регулярных выражениях применяется специальный язык описания шаблонов, спецификация которого содержится в стандарте POSIX 1003.2. Они всегда чувствительны к регистру символов. Регулярные выражения также подвержены двойному синтаксическому анализу, но предложение ESCAPE в операторе REGEXP не поддерживается.

На самом верхнем уровне регулярное выражение состоит из одного или нескольких блоков, разделенных вертикальной чертой (|). Этот символ аналогичен оператору OR в SQL, т.е. сравниваемая строка может соответствовать любой из ветвей. В табл. 10.5 приведен ряд примеров.

Регулярное выражение	*Чему соответствует*
apple	apple
apple|ball	apple или ball
begin|end|break	begin, end или break

Каждая ветвь состоит из одной или нескольких атомарных конструкций, за каждой из которых может следовать модификатор, определяющий число последовательных повторений конструкции. Символ * означает произвольное количество повторений, символ + — как минимум одно, а символ ? — не более одного.

Существуют и более точные модификаторы. Они записываются в фигурных скобках. Если в скобках стоит одно число, конструкция должна встречаться указанное количество раз. Число с запятой означает, что конструкция встречается как минимум указанное количество раз. Два числа, разделенных запятой, задают диапазон "от и до". Соответствующие примеры приведены в табл. 10.6.

Регулярное выражение	*Чему соответствует*
`a(b*)`	`a`, `ab`, `abb`,...
`a(b+)`	ab, abb, `abbb`, ...
`a(b?)`	`a` **или** `ab`
`a(b{3})`	abbb
`a(b{2,})`	abb, abbb, `abbbb`, ...
`a(b{2,4})`	`abb`, `abbb`, `abbbb`

Атомарная конструкция представляет собой последовательность символов, часть из которых имеет специальное назначение, а часть интерпретируется буквально. Метасимвол `.` соответствует произвольному символу. Метасимвол `^` обозначает начало строки, а метасимвол $ — ее конец. Если необходимо отменить интерпретацию специального символа, поставьте перед ним обратную косую черту. Разрешается группировать конструкции с помощью круглых скобок, чтобы все выражение в скобках стало атомарной конструкцией.

Квадратные скобки предназначены для задания диапазона возможных значений. Эти значения могут быть просто перечислены через запятую, но можно также указать начало и конец непрерывного диапазона, разделив их символом `-`. Если выражение в скобках начинается с символа `^`, то этому выражению соответствует все, что не входит в заданный диапазон. Обратите внимание на двойственность символа `^`.

Помимо списков и диапазонов в квадратных скобках можно указывать классы символов. Имена классов окружаются двоеточиями, например `[:alpha:]`, что значит "все символы алфавита". Допустимые классы таковы: `alnum`, `alpha`, `blank`, `cntrl`, `digit`, `graph`, `lower`, `print`, `punct`, `space`, `upper` и `xdigit`.

Наконец, есть два специальных шаблона, соответствующих началу и концу слова: `[:<:]` и `[:>:]`. Под словом здесь понимается произвольная последовательность алфавитно-цифровых символов и знаков подчеркивания. В табл. 10.7 показаны примеры различных регулярных выражений, в том числе с квадратными скобками.

Регулярное выражение	*Чему соответствует*
`a.c`	Любая **трехсимвольная** строка, начинающаяся с `a` и заканчивающаяся на `c`: `aac`, `abc`, `acc` и т.д.
`^a.*`	Любая строка, начинающаяся с `a`
`[a-c]*x$`	Любая строка, содержащая буквы `a`, `b` и `c` и заканчивающаяся на `x`: `x`, `ax`, `bx`, `abx` и т.д.

`b[ao]y`	bay или boy; возможна альтернативная запись bay | boy
`[^Zz]{5}`	Любая **пятисимвольная** строка, не содержащая Z или z
`[[:digit:]]`	Любая цифра; возможна альтернативная запись [0-9]
`[[:<:]]a.*`	Любое слово, начинающееся с a

Логические операторы

Логические операторы (табл. 10.8) работают с булевыми величинами. Обратите внимание на то, что у каждого оператора есть две формы записи: словесная и символьная.

Оператор	*Операция*
AND, `&&`	Логическое умножение (И)
OR, |	Логическое сложение (ИЛИ)
NOT, `!`	Логическое отрицание (НЕ)

Те, кто не знакомы с логическими операциями, могут обратиться к табл. 10.9. Булевы переменные `p` и `q` содержат значения "истина" или "ложь", что составляет четыре комбинации.

p	*q*	*p И q*	*p ИЛИ q*	*НЕ p*
Ложь	Ложь	Ложь	Ложь	Истина
Ложь	Истина	Ложь	Истина	Истина
Истина	Ложь	Ложь	Истина	Ложь
Истина	Истина	Истина	Истина	Ложь

Побитовые операторы

Побитовые операторы (табл. 10.10) работают с числами как с цепочками битов. Напомним, что двоичное число состоит из нулей и единиц, причем позиция каждой цифры соответствует степени числа 2, начиная с нулевой. Например, десятичное 9 в двоичной форме записывается как 1001 ($2^3+0+0+2^0$).

Оператор	*Операция*
|	Побитовое сложение (ИЛИ)
&	Побитовое умножение (И)
<<	Сдвиг всех битов влево
>>	Сдвиг всех битов вправо
	Побитовое отрицание (НЕ)

Побитовые операции И, ИЛИ и НЕ работают так же, как и их логические эквиваленты, но для каждой пары битов выполняется отдельная операция. К примеру, результатом операции 101 | 100 будет 101, поскольку она распадается на три операции: "1 ИЛИ 1", "О ИЛИ 0" и "1 ИЛИ О".

Операторы << и >> сдвигают все биты числа соответственно влево и вправо, заполняя пустые позиции нулями. Таким образом, 100 >> 2 равно 1, а 101 << 3 равно 101000.

Оператор ~ инвертирует каждый бит числа. Например, ~101 равно 010.

Прочие операторы

Оператор := предназначен для присваивания переменным значений в инструкции SELECT. О нем уже говорилось выше.

Оператор BINARY преобразует строку в двоичную форму, вследствие чего в операции сравнения строк будет учитываться регистр символов. Если одна из строк в операции сравнения является двоичной, то и другая тоже становится двоичной. Два типа операций сравнения демонстрируются в листинге 10.9. Строки, задаваемые в **шестнадцатеричном** виде, всегда считаются двоичными.

```
mysql> SELECT "core" = "Core";
+-----------------+
| "core" = "Core" |
+-----------------+
|               1 |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT BINARY "core" = "Core";
+------------------------+
| BINARY "core" = "Core" |
+------------------------+
|                      0 |
+------------------------+
```

Выражения

Выражения — это комбинации из идентификаторов и операторов. Идентификаторами могут быть литералы, а также имена столбцов и переменных. Выражения вычисляются слева направо в соответствии с приоритетом операторов (табл. 10.11). Например, умножение выполняется раньше, чем сложение. Операторы, имеющие равный приоритет, тоже оцениваются слева направо.

Таблица 10.11

Приоритет	*Оператор*
Самый высокий	- (унарный), + (унарный), BINARY

С помощью круглых скобок можно задавать принудительный порядок вычислений. Операнд, представляющий собой выражение в скобках, оценивается раньше, чем обычный операнд. Допускается вложение скобок. В листинге 10.10 показан пример влияния скобок на порядок вычислений.

```
mysql> SELECT 3 + 2 * 7;
+-----------+
| 3 + 2 * 7 |
+-----------+
|        17 |
+-----------+
1 row in set (0.00 sec)

mysql> SELECT (3 + 2) * 7;
+-------------+
| (3 + 2) * 7 |
+-------------+
|          35 |
+-------------+
1 row in set (0.00 sec)
```

Имена с пробелами

В SQL для разделения идентификаторов применяются пробелы. Вот почему строковые литералы заключаются в кавычки. Кавычки позволяют также отличать литералы от имен баз данных, таблиц и столбцов. Имя таблицы, взятое в кавычки, становится обычной строкой. Но если имя содержит пробелы, то можно заключить его в обратные кавычки (``), тогда оно будет интерпретироваться правильно.

ТИПЫ СТОЛБЦОВ И ИНДЕКСОВ

В этой главе..

Числа
Строки
Значения даты/времени
Альтернативные типы данных
Индексы

Глава 11

В MySQL поддерживаются три основных типа столбцов: числа, строки и значения даты/времени. К первому типу относятся целые числа и числа с плавающей запятой. Строки содержат цепочки текстовых или двоичных символов. Значения даты/времени могут быть представлены в самых разных форматах.

Тип столбца определяет диапазон значений, которые можно заносить в данный столбец. Здесь не допускается та гибкость, которая свойственна переменным. Тип столбца жестко задается при определении таблицы. При записи в ячейку значения другого формата осуществляется автоматическое приведение типа.

Спецификация большинства типов допускает указание размерности в скобках. Это значение определяет максимальное количество символов в представлении значения, не считая знака "минус" и цифр после запятой для чисел. В случае чисел с плавающей запятой можно указать вторую размерность, определяющую количество цифр после запятой.

Как правило, размерность не играет никакой роли для числовых столбцов. Более крупные значения принимаются и отображаются правильно. Разве что числа с плавающей запятой могут округляться до нужной разрядности. Размерность важна для строковых столбцов. Слишком длинные строки усекаются в момент вставки в таблицу.

Числа

Числа могут храниться в целом и десятичном форматах, а также в формате с плавающей запятой. Особенностью десятичных чисел является то, что они хранятся в целочисленном виде с указанием точного количества цифр после запятой. Числа с плавающей запятой вычисляются приблизительно, с той или иной точностью.

Целые числа

В табл. 11.1 перечислены целочисленные типы данных и соответствующие им диапазоны значений. Числа, выходящие за пределы диапазона, преобразуются в минимальное или максимальное значение.

Тип	*Знаковый диапазон*	*Беззнаковый диапазон*
TINYINT	-127-127	0-255
SMALLINT	-32768-32767	0-65535
MEDIUMINT	-8388608 – 8388607	0 – 16777215
INT,INTEGER	-2147483648 – 2147483647	0-4294967295
BIGINT	-9223372036854775808 – 9223372036854775807	0 – 18446744073709551615

Синтаксис спецификации целочисленного типа таков:

```
<тип>[(<размерность>)] [UNSIGNED] [ZEROFILL]
```

Как минимум, нужно указать имя типа. Следом за ним в скобках приводится требуемая размерность. Флаг UNSIGNED обозначает беззнаковое число. Флаг ZEROFILL говорит о том, что в случае необходимости число должно быть дополнено ведущими нулями до нужной размерности. Применение флага ZEROFILL продемонстрировано в листинге 11.1.

```
mysql> CREATE TABLE testint (
    -> i INT(11) UNSIGNED ZEROFILL
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO testint VALUES (123);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM testint;
+-------------+
| i           |
+-------------+
| 00000000123 |
+-------------+
1 row in set (0.00 sec)
```

Числа с плавающей запятой

Числа с плавающей запятой представляют собой приблизительные значения. Они подходят для столбцов, где хранятся дробные величины или числа, выходящие за пределы самого крупного целочисленного диапазона (BIGINT). Чтобы не происходило непредсказуемых погрешностей вычислений, MySQL округляет вставляемые числа до требуемой точности, которая указана в определении столбца, хотя на практике такие погрешности никогда не возникают.

Имеются два типа чисел с плавающей запятой: числа одинарной точности и числа двойной точности. Их диапазоны приведены в табл. 11.2.

Тип	*Диапазон*
FLOAT	±1,175494351E-38 - ±3,402823466E+38
DOUBLE, DOUBLE PRECISION, REAL	±2,2250738585072014E-308 - ±1,7976931348623157E+308

Синтаксис спецификации типа с плавающей запятой таков:

```
<тип>[(<размерность>,<точность>)] [ZEROFILL]
```

Точность – это количество цифр после запятой. Флаг ZEROFILL указывает на необходимость дополнения числа ведущими нулями до нужной размерности. Беззнаковые числа с плавающей запятой в MySQL не поддерживаются.

При описании столбца данного типа достаточно указывать одну лишь размерность. Это соответствует спецификации ODBC. Согласно ей, столбец, размерность которого меньше чем 24, будет иметь тип FLOAT, иначе — тип DOUBLE (листинг 11.2).

```
mysql> CREATE TABLE testfloat(
    -> f FLOAT(20),
    -> d FLOAT(40)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE testfloat;
+-------+--------+------+-----+---------+-------+
| Field | Type   | Null | Key | Default | Extra |
+-------+--------+------+-----+---------+-------+
| f     | float  | YES  |     | NULL    |       |
| d     | double | YES  |     | NULL    |       |
+-------+--------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

Десятичные числа

Десятичные числа имеют фиксированное количество цифр после запятой. Эти цифры вычисляются точно, в отличие от чисел с плавающей запятой. Столбцы данного типа удобно использовать для хранения денежных величин, где погрешности округлений недопустимы.

Создание столбца типа DECIMAL демонстрируется в листинге 11.3. Размерность определяет общее количество цифр числа, включая те, что стоят после запятой. В случае размерности 6 и точности 2 допустимый диапазон значений будет таким: -9999,99 — 9999,99. Беззнаковые десятичные числа не поддерживаются.

```
mysql> CREATE TABLE testdec (
    -> m DECIMAL(6,2) ZEROFILL
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO testdec VALUES (123.45)
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM testdec;
+---------+
| m       |
+---------+
| 0123.45 |
+---------+
1 row in set (0.00 sec)
```

Строки

Строки представляют собой цепочки символов и бывают трех типов: **ASCII-строки**, большие двоичные объекты и перечисления. Длина **ASCII-строк** ограничена 255-ю символами. Большой двоичный объект (Binary Large Object, BLOB) может содержать до 16 Мбайт текста, а в MySQL версии 4.1 этот предел будет расширен до 4 Гбайт. **Перечисления** — это строки с предопределенным набором возможных значений.

ASCII-строки

ASCII-строки имеют тип CHAR либо VARCHAR. Описание этих типов вместе с их псевдонимами приведено в табл. 11.3. Столбец типа CHAR имеет фиксированную размерность. Когда значения такого столбца записываются на диск, они дополняются до нужной размерности хвостовыми **пробелами**, которые автоматически удаляются при извлечении этих значений. По сути, если в такой столбец заносится строка с хвостовыми пробелами, они удаляются.

Столбец типа VARCHAR хранится в базе данных в виде строки переменной длины, а в остальном ведет себя подобно столбцу типа CHAR. Это подразумевает удаление хвостовых пробелов при записи строки в ячейку. В MySQL версии 4.1 в столбцах типа

VARCHAR хвостовые пробелы удаляться не будут. В настоящее время аналогичные функции можно реализовать с помощью типа TINYTEXT.

Тип	*Описание*
CHAR, CHARACTER	Строка фиксированной длины
VARCHAR, CHARACTER VARYING	Строка переменной длины

Синтаксис спецификации строкового типа таков:

[NATIONAL] <*тип*>[(<*размерность*>)] [BINARY]

Значение размерности определяет максимальную длину строки. Более длинные строки будут усекаться. Если присутствует флаг BINARY, то вoперациях сортировки и сравнения значений данного столбца будет учитываться регистр символов. В противном случае регистр не играет роли.

Флаг NATIONAL делает доменом столбца набор символов, установленный в системе по умолчанию. Это стандартный режим работы программы MySQL, поэтому данный флаг можно не указывать. Он присутствует для совместимости со стандартом SQL92.

Большие двоичные объекты

Большие двоичные объекты ведут себя подобно строкам переменной длины, но способны хранить огромные информационные массивы. Кроме того, хвостовые пробелы не отсекаются. Столбцы данного вида могут быть либо двоичными, либо текстовыми с дополнительной дифференциацией по размерности (табл. 11.4). Столбцы семейства BLOB являются двоичными, поэтому в операциях сравнения будет учитываться регистр. Столбцы семейства TEXT являются текстовыми и не чувствительны к регистру. Как и в случае обычных строк, значения, превышающие максимальную длину, усекаются. У столбцов данного типа отсутствуют значения по умолчанию. Подобная возможность появится в MySQL версии 4.1.

Тип	*Максимальнаядлина*
TINYBLOB, TINYTEXT	255 байтов
BLOB, TEXT	64 Кбайт (65535 байтов)
MEDIUMBLOB, MEDIUMTEXT	16 Мбайт (16777215 байтов)
LONGBLOB, LONGTEXT	4 Гбайт (4294967295 байтов)

Перечисления и множества

К строковым типам относятся также перечисления и множества (табл. 11.5). Ячейка типа ENUM (перечисление) в определенный момент времени содержит лишь одно значение из списка возможных. Ячейка типа SET (множество) может содержать несколько уникальных значений из числатех, что входят в множество.

Тип	*Максимальное число значений*
ENUM	65535
SET	64

Значение, заносимое в ячейку типа ENUM, указывается в строковом виде. Значение ячейки типа SET записывается в виде строки, содержащей разделенные запятыми элементы множества. Это подразумевает, что сами элементы не содержат запятых.

В листинге 11.4 демонстрируется создание таблицы со столбцами типа ENUM и SET, а также вставка значений в эти столбцы. Строки, которые не указаны в определении перечисления или множества, игнорируются, за исключением пустых строк.

```
mysql> CREATE TABLE testenum (
    -> e ENUM ('a', 'b', 'c'),
    -> s SET ('a', 'b', 'c')
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO testenum VALUES ('a', ('b,c,x'));
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO testenum VALUES (2, 5);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM testenum;
+------+------+
| e    | s    |
+------+------+
| a    | b,c  |
| b    | a,c  |
+------+------+
2 rows in set (0.00 sec)
```

Если в качестве значения перечисления указано число, оно считается индексом элемента в определении перечисления. Нумерация элементов начинается с единицы. Ячейки типа SET интерпретируются как битовые поля. Первому элементу множества соответствует младший значащий бит.

Значения даты/времени

В MySQL поддерживаются пять типов столбцов, хранящих информацию о дате/времени (табл. 11.6). Чаще всего применяются типы DATETIME и TIMESTAMP. Для всех типов значение 0 является допустимым.

Тип	*Диапазон значений*
DATE	'1000-01-01' — '9999-12-31'
DATETIME	'1000-01-01 00:00:00' - '9999-12-31 23:59:59'
TIME	'-838:59:59' - '838:59:59'
TIMESTAMP	'1970-01-01 00:00:00' - '2037-12-31 23:59:59'
YEAR	1970 — 2069 или 1901 — 2155

Значения даты/времени вполне можно хранить в целочисленных или строковых столбцах, но лишь благодаря специальным типам данных появляется возможность эффективно манипулировать этими значениями. Например, поддерживается операция сложения дат, что продемонстрировано в листинге 11.5. Для управления такими столбцами на уровне строк или чисел потребовалось бы писать специальные программы. Значения даты/времени могут присутствовать в предложении ORDER BY и их можно сравнивать друг с другом.

```
mysql> CREATE TABLE testtime (
    -> ts TIMESTAMP,
    -> dt DATETIME,
    -> d DATE,
    -> t TIME,
    -> y YEAR(2)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO testtime (dt, d, t, y)
    ->      VALUES ('1989-10-17 17:04:13', 19970322,
    ->              '10:15:00', 95);
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM testtime \G
*************************** 1.row ***************************
ts: 20010418111943
dt: 1989-10-17 17:04:13
 d: 1997-03-22
 t: 10:15:00
 y: 95
1 row in set (0.01 sec)
```

```
mysql> UPDATE testtime
    ->     SET y = NOW(),
    ->         dt = dt + INTERVAL 15 MONTH;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 1

mysql> SELECT * FROM testtime \G
*************************** 1. row ***************************
ts: 20010418111957
dt: 1991-01-17 17:04:13
 d: 1997-03-22
 t: 10:15:00
 y: 01
1 row in set (0.00 sec)
```

Значения данного типа являются составными, представляющими собой набор целочисленных значений. У каждого компонента свой диапазон. Например, номер месяца должен находиться в интервале от 1 до 12, номер дня — от 1 до 31, номер часа — от 0 до 23, а число минут и секунд может меняться от 0 до 59.

MySQL не проверяет осмысленность таких дат, как, например, 31 февраля. Разработчики MySQL справедливо полагают, что подобные проверки должны осуществляться не в СУБД, а в прикладных программах. Допускаются неполные даты наподобие 1970-00-00. Это означает, что дата относится к 1970 году, но месяц и день неизвестны.

Форматы, указанные в табл. 11.6, используются при выводе значений на экран. Вводимые данные могут быть представлены по-другому, в строковом или числовом виде. Предполагается, что дата записывается как набор целых чисел в следующем порядке: год, месяц, день, час, минуты, секунды. Номер года может состоять из двух или четырех цифр. Поля года, месяца и дня являются обязательными. Отсутствующие поля времени заполняются нулями.

Значение даты, записанное в числовом виде, представляет собой цепочку цифр. Дробная часть числа отбрасывается. В строковом представлении компоненты разделяются знаками пунктуации. Это может быть любой печатный символ кроме буквы и числа. Между номером дня и часа может стоять пробел, но между всеми остальными компонентами пробелы недопустимы. Неправильные даты обнуляются. К двухсимвольным номерам года в диапазоне 00–69 добавляется значение 2000, а к номерам в диапазоне 70–99 добавляется 1900.

Спецификация типов DATETIME, DATE и TIME не поддерживает понятия размерности. Значения этих типов всегда содержат определенное количество информации. Столбцы типа YEAR могут иметь размерность 2 или 4.

Столбцы типа TIMESTAMP по умолчанию имеют размерность 14. Допускаются также размерности 12, 8 и 6. Нечетное число преобразуется в ближайшее большее четное. Размерность определяет, какая часть спецификации даты/времени должна отображаться в результатах запроса. Реальные значения столбца всегда хранят полную спецификацию. Необычный диапазон значений столбца TIMESTAMP объясняется тем, что в UNIX время измеряется ввиде количества секунд, прошедших с начала так называемой *эпохи* (1 января 1970 г.). Это значение хранится в виде четырехбайтового целого числа.

В самую первую ячейку типа TIMESTAMP автоматически записывается текущее время при изменении других ячеек строки. Данная особенность была продемонстрирована в листинге 11.5. Чтобы этого избежать, нужно явно занести в ячейку какое-то значение.

Альтернативные типы данных

В MySQL поддерживается несколько альтернативных названий базовых типов данных (табл. 11.7). Благодаря им упрощается перенос баз данных из других СУБД в MySQL.

Псевдоним	*Тип в MySQL*
BINARY (*число*)	CHAR.(*число*) BINARY
CHAR (*число*)	VARCHAR (*число*)
FLOAT4	FLOAT
FLOAT8	DOUBLE
INT1	TINYINT
INT2	SMALLINT
INT3	MEDIUMINT
INT4	INT
INT8	BIGINT
LONG VARBINARY	MEDIUMBLOB
LONG VARCHAR	MEDIUMTEXT
MIDDLEINT	MEDIUMINT
VARBINARY(*число*)	VARCHAR (*число*) BINARY

Индексы

Индексы — это внутренние механизмы, предназначенные для ускорения поиска строк. Они всегда связаны со списком столбцов какой-либо таблицы. Создать индекс можно при определении таблицы или позднее с помощью инструкции ALTER TABLE. То же самое делает инструкция CREATE INDEX, поддерживаемая другими СУБД. Синтаксис создания индексов будет описан в главе 13, "Инструкции SQL". В табл. 11.8 перечислены типы индексов, поддерживаемые в MySQL.

Тип	*Описание*
INDEX, KEY	Обычный индекс
UNIQUE	Индекс, накладывающий на столбец ограничение уникальности
PRIMARY KEY	Индекс, используемый для идентификации записей
FULLTEXT	Индекс, используемый для поиска больших дв оичных объектов

Индексы используются при выполнении операций объединения. Обычный индекс может быть создан для любого столбца, даже такого, в котором есть повторяющиеся или пустые значения.

Индекс типа UNIQUE содержит как указатели, так и ограничения на значения столбца. Он определяет ключ-кандидат, описанный в главе 5, "Реляционная модель". Каждое значение такого индекса уникальным образом идентифицирует строку таблицы.

Первичный ключ — это уникальный индекс, выбранный для идентификации записей. У таблицы может быть многоуникальных индексов и лишь один первичный ключ.

В случае строковых столбцов можно создать индекс по нескольким первым символам. Иногда это при водит к существенному ускорению поиска. Столбцы типа BLOB или TEXTдопускают лишь такой способ индексации. Максимальная размерность индекса при этом составляет 1024 символа.

Индекс типа FULLTEXT применяется дляпоиска слов в больших двоичных объектах. Этот индекс содержит показатели релевантности поиска, находящиеся в диапазоне от 0,0 до 1,0. Все слова в индексируемом тексте разделяются пробелами. Количество появлений слова в тексте определяет вес этого слова, причем слова, встречающиеся чаще, имеют более низкую релевантность. Слова, состоящие не более чем из трех символов, а также слова, встречающиеся в более чем половине строк, отбрасываются и считаются нерелевантными.

Индексы типа FULLTEXT предназначены для относительно крупных столбцов. Вероятно, их применение не даст полезных результатов, если индексом охвачено менее 100000слов.

В MySQL версии 4.1 флаг UNIQUE будет задавать режим, в котором на столбец накладывается ограничение уникальности, но индекс не строится. Будет также улучшенапроизводительностьиндексовтипаFULLTEXT.

ВСТРОЕННЫЕ ФУНКЦИИ

В этой главе...

Отладка и конфигурирование
Управляющие функции
Статистические функции
Математические функции
Строки
Функции работы с датой и временем
Прочие функции
Процедуры

Глава 12

Функции — это подпрограммы, возвращающие значения, иногда основываясь на значениях входных параметров. Функции можно использовать везде, где разрешены выражения, в том числе вместо параметров самих функций. Другими словами, допускаются вложенные вызовы функций.

Большинство функций принимает параметры в круглых скобках, причем между именем функции и открывающей скобкой не должно быть пробелов. Это помогает программе MySQL отличать имена функций от имен столбцов. Подобное поведение программы можно отменить с помощью аргументов командной строки, но тогда нельзя будет использовать функции в списке возвращаемых столбцов.

Некоторые функции правильнее было бы назвать операторами. Я решил поместить их здесь, поскольку их работа во многом напоминает работу функций. Круглые скобки им, естественно, не требуются.

Описание каждой функции начинается с заголовка, где указан формат ее вызова. Слова, набранные прописными буквами, являются зарезервированными. Строчными буквами даны параметры, которые могут быть литералами, именами столбцов или выражениями. MySQL распознает имена функций, записанные произвольной комбинацией прописных и строчных букв.

Троеточие (`...`) означает, что функция принимает список значений, разделенных запятыми. Все аргументы, находящиеся в квадратных скобках, являются необязательными. У некоторых функций есть несколько уровней необязательных параметров.

Не забывайте о том, что результатом большинства выражений, в которых участвует значение NULL, будет тоже NULL. Это справедливо и в отношении вызовов функций. Например, конкатенация значения NULL и произвольной строки даст в итоге NULL.

Отладка и конфигурирование

Описанные ниже функции возвращают информацию о сервере либо помогают при отладке.

BENCHMARK(циклы, значение)

Функция BENCHMARK() вычисляет требуемое выражение столько раз, сколько указано в первом аргументе. Она всегда возвращает нуль, но если используется клиент mysql, то будет также подсчитано время, затраченное на выполнение функции. В листинге 12.1 показано, сколько времени ушло на выполнение 100000 вызовов функции SOUNDEX(). Здесь подразумевается время, прошедшее с момента ввода запроса до момента получения его результатов, а не время использования центрального процессора. Время выполнения функции может существенно замедляться из-за загруженности сервера или сети.

```
mysql> SELECT BENCHMARK(100000, SOUNDEX('Core'));
+------------------------------------+
| BENCHMARK(100000, SOUNDEX('Core')) |
+------------------------------------+
|                                  0 |
+------------------------------------+
1 row in set (0.26 sec)
```

CONNECTION_ID()

Эта функция возвращает идентификатор текущего соединения (листинг 12.2).

Листинг 12.2.

```
+                 +
I CONNECTION_ID() |
+                 +
I              91 |
+                 +
1 row in set (0.00 sec)
```

DATABASE()

Эта функция возвращает имя стандартной базы данных (листинг 12.3). Если таковая не выбрана, возвращается NULL.

```
mysql> SELECT DATABASE()
+------------+
| DATABASE() |
+------------+
| test       |
+------------+
1 row in set (0.00 sec)
```

LAST_INSERT_ID([идентификатор])

Для каждого соединения программа MySQL хранит последнее значение счетчика, служащего первичным ключом. Функция LAST_INSERT_ID() возвращает это значение (листинг 12.4). Если функции передать аргумент, она установит его в качестве нового значения счетчика.

```
mysql> SELECT LAST_INSERT_ID();
+------------------+
| last_insert_id() |
+------------------+
|                0 |
+------------------+
1 row in set (0.00 sec)

mysql> INSERT INTO item VALUES (NULL, 'Soap', 3.75, '3-Pack', 200);
Query OK, 1 row affected (0.06 sec)

mysql> SELECT LAST_INSERT_ID();
+------------------+
| last_insert_id() |
+------------------+
|                5 |
+------------------+
1 row in set (0.00 sec)
```

Существуют два основных применения функции LAST_INSERT_ID(). Чаще всего она используется со столбцами типа AUTO_INCREMENT. Если вследствие какого-либо запроса значение данного столбца увеличивается автоматически, то это значение запоминается и впоследствии возвращается функцией LAST_INSERT_ID(). Если же значение столбца задается вручную, то на функции это никак не отражается. В листинге 12.4 первый аргумент предложения VALUES инструкции INSERT равен NULL, что приводит к автоматическому увеличению счетчика.

С другой стороны, с помощью функции LAST_INSERT_ID() можно имитировать последовательные счетчики, связав ее с таблицей, состоящей из одного столбца. Об этом подробно рассказывалось в главе 9, "Транзакции и параллельные вычисления".

SESSION_USER()

Это синоним функции USER().

SYSTEM_USER()

Это синоним функции USER().

USER()

Эта функция возвращает имя пользователя, начавшего текущий сеанс работы с сервером (листинг 12.5). Сообщается также имя компьютера, за которым работает пользователь.

```
mysql> SELECT USER();
+----------------+
| USER()         |
+----------------+
| root@localhost |
+----------------+
1 row in set (0.00 sec)
```

VERSION()

Эта функция сообщает номер версии сервера MySQL (листинг 12.6).

```
mysql> SELECT VERSION();
+-----------+
| VERSION() |
+-----------+
| 3.23.36   |
+-----------+
1 row in set (0.00 sec)
```

Управляющие функции

Описанные ниже функции управляют выполнением программного кода. Функции CASE и CASE WHEN реализованы подобно инструкциям языков третьего поколения и не используют скобки, но в остальном они аналогичны функциям. Две функции блокирования, а также функция, заставляющая главную базу данных дождаться синхронизации подчиненной базы данных, полезны при работе в многопользовательской среде.

CASE

Синтаксис конструкции CASE **таков:**

```
CASE проверяемое_выражение
     WHEN значение1 THEN возвращаемое_значение1
     [WHEN значение2 THEN возвращаемое_значение2]

     [ELSE значение_по_умолчанию]
END
```

Конструкция CASE **вычисляет проверяемое выражение и сравнивает результат с одним из значений, указанных в предложении** WHEN. **Если найдено совпадение, возвращается соответствующее значение предложения** THEN. **В противном случае возвращается значение по умолчанию.**

В листинге 12.7 демонстрируется применение конструкции CASE **по отношению к таблице, состоящей из пяти строк. Когда идентификатор равен 1 или 2, возвращается строка** One **или** Two, **иначе отображается строка** Other Value.

```
mysql> SELECT ID, CASE ID
    ->            WHEN 1 THEN 'One'
    ->            WHEN 2 THEN 'Two'
    ->            ELSE 'Other Value'
    ->            END AS 'CASE Result'
    ->   FROM item;
+----+-------------+
| ID | CASE Result |
+----+-------------+
|  1 | One         |
|  2 | Two         |
|  3 | Other Value |
|  4 | Other Value |
|  5 | Other Value |
+----+-------------+
5 rows in set (0.00 sec)
```

Тип возвращаемого значения соответствует типу аргумента самого первого предложения THEN.

CASE WHEN

Синтаксис альтернативного варианта конструкции CASE **таков:**

```
CASE
     WHEN условие1 THEN возвращаемое_значение1
     [WHEN условие2 THEN возвращаемое_значение2]

     [ELSE значение_по_умолчанию]
END
```

Здесь проверяется каждое условие по отдельности. Возвращается то значение, для которого условие является истинным. Возвращаемое значение может быть произвольным выражением, в том числе результатом вызова функции.

```
mysql> SELECT ID, CASE
    ->            WHEN ID < 3 THEN POWER(ID, 2)
    ->            WHEN ID = 4 THEN ID
    ->             ELSE Name
    ->              END AS 'CASE Result'
    ->   FROM item;
+----+-------------+
| ID | CASE Result |
+----+-------------+
| 1  |    1.000000 |
| 2  |    4.000000 |
| 3  |       Brush |
| 4  |           4 |
| 5  |        Soap |
+----+-------------+
5 rows in set (0.00 sec)
```

Поскольку результат конструкции CASE может менять свой тип в зависимости от условия, столбцу присваивается тип аргумента самого первого предложения THEN.

GET_LOCK(имя, тайм-аут)

Функция GET_LOCK() запрашивает именованную блокировку на указанное число секунд. Функция не завершится до тех пор, пока не истечет период тайм-аута или блокировка не будет получена.

Одному сеансу может принадлежать только одна блокировка. Вызов функции GET_LOCK() приводит к снятию всех удерживаемых блокировок, но лучше все же снимать их явно с помощью функции RELEASE_LOCK(). Блокировка снимается также в случае завершения сеанса.

Наличие блокировки не дает ее владельцу никаких преимуществ и никак не ограничивает работу других сеансов. Тем не менее при согласованном **применении** именованных блокировок несколькими программами появляется возможность реализовать блокирование произвольного уровня детализации. Пример использования функции GET_LOCK() для блокирования записей приводился в главе 9, "Транзакции и параллельные вычисления". Похожий пример дан в листинге 12.9.

```
mysql> SELECT GET LOCK('item.ID=3', 60);
+---------------------------+
| GET_LOCK('item.ID=3', 60) |
+---------------------------+
|                         1 |
```

```
+                              +
1 row in set  (0.00 sec)

mysql> UPDATE item SET Price=3.15 WHERE ID=3;
Query OK, 1 row affected (0.00 sec)
Rows matched 1 Changed 1 Warnings 0

mysql> SELECT RELEASE LOCK('item.ID=3');
+---------------------------+
| RELEASE_LOCK('item.ID=3') |
+---------------------------+
|                         1 |
+---------------------------+
1 row in set (0.00 sec)
```

IF (условие, значение_истина, значение_ложь)

Функция IF() возвращает разные значения в зависимости от того, истинным или ложным является проверяемое выражение. Результат проверки приводится к целому типу.

Как и в случае конструкции CASE, тип результата функции IF() нельзя определить заранее. Если одно из возвращаемых значений — строка, результат будет строковым. Если же это условие не выполняется, но одно из значений является числом с плавающей запятой, то и результат будет аналогичным. В противном случае оба значения приводятся к целочисленному типу.

В листинге 12.10 с помощью функции IF() проверяется, является ли текущий день будним или выходным.

```
mysql> SELECT IF(DAYOFWEEK(NOW()) in (0,6,7), 'weekend', 'weekday');
+-------------------------------------------------------+
| IF(DAYOFWEEK(NOW()) in (0,6,7), 'weekend','weekday') |
+-------------------------------------------------------+
|                                                weekday |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

IFNULL(проверяемое_значение, возвращаемое_значение)

Функция IFNULL() возвращает проверяемое значение, если оно не равно NULL. В противном случае возвращается второй аргумент. В листинге 12.11 показано, как с помощью этой функции заменить значения NULL описательной фразой.

```
mysql> SELECT Name,
    -> IFNULL(Description, 'No Description') AS Description
    ->   FROM item;
+------------+-----------------+
| Name       | Description     |
+------------+-----------------+
| Toothbrush | No Description  |
| Comb       | No Description  |
| Brush      | No Description  |
| Toothpaste | Mint-Flavor     |
| Soap       | 3-Pack          |
+------------+-----------------+
5 rows in set (0.01 sec)
```

MASTER_POS_WAIT(имя, позиция)

Функция MASTER_POS_WAIT() дожидается, пока подчиненный сервер синхронизируется с главным сервером в процессе репликации. Необходимо указать имя журнального файла и позицию, которой должен достигнуть подчиненный сервер. Если данный компьютер не сконфигурирован в качестве главного сервера, функция немедленно возвращает NULL. Если же подчиненный сервер еще не запущен, функция блокируется до тех пор, пока сервер не запустится и не достигнет указанной позиции журнального файла.

Функция возвращает число событий, зарегистрированных в процессе ожидания. Подробнее о репликации рассказывается в главе 29, "Распределенные базы данных".

NULLIF(проверяемое_значение1, проверяемое_значение2)

Функция NULLIF() возвращает NULL, если оба проверяемых значения равны друг другу. В противном случае возвращается первый аргумент. С помощью этой функции удобно преобразовывать значения 0 в NULL (листинг 12.12).

```
mysql> SELECT Name, NULLIF(Inventory, 0) AS Inventory
    -> FROM item;

| Name       | Inventory |
+------------+-----------+
| Toothbrush | NULL      |
| Comb       | NULL      |
| Brush      | NULL      |
| Toothpaste | NULL      |
| Soap       | 200       |
+------------+-----------+
5 rows in set (0.01 sec)
```

RELEASE_LOCK(имя)

Функция RELEASE_LOCK() снимает указанную именованную блокировку, которая ранее была получена с помощью функции GET_LOCK(). Если имя блокировки не было зарегистрировано, возвращается NULL. Пример использования этой функции был приведен при описании функции GET_LOCK().

Статистические функции

Описанные ниже функции выполняются по отношению к совокупности значений целого столбца. Если предложение GROUP BY отсутствует, обработке подвергается каждая запись. Ячейки со значением NULL не входят в оценочное множество. В листинге 12.13 приведено определение таблицы, которая используется в примерах данного раздела.

```
CREATE TABLE grouptest (
        ID INT NOT NULL AUTO_INCREMENT,
        Team VARCHAR(16),
        Score INT,
        PRIMARY KEY(ID)
);
INSERT INTO grouptest (Team, Score) VALUES
        ('Red', 11)
        ('Red', 45)
        ('Red', 98)
        ('Red', 19)
        ('Red', 11)
        ('Red', 37),
        ('Red', 17)
        ('Red', 75)
        ('Blue', 23)
        ('Blue' 91)
        ('Blue' 80)
        ('Blue' 63)
        ('Blue' 55)
        ('Blue' 89)
        ('Blue' 64)
        ('Blue' 5);
```

AVG(столбец)

Функция AVG() вычисляет среднее арифметическое группы значений (листинг 12.14). Это число определяется путем суммирования всех элементов группы и деления результата на общее число элементов.

```
mysql> SELECT Team, AVG(Score)
    ->   FROM grouptest
    ->  GROUP BY Team;
+------+------------+
| Team | AVG(Score) |
+------+------------+
| Blue |    58.7500 |
| Red  |    39.1250 |
+------+------------+
2 rows in set (0.01 sec)
```

BIT_AND(столбец)

Функция BIT_AND() выполняет побитовое умножение всех элементов группы и возвращает результат в виде целого числа (см. ниже).

BIT_OR(столбец)

Функция BIT_OR() выполняет побитовое сложение всех элементов группы и возвращает результат в виде целого числа (листинг 12.15).

```
mysql> SELECT Team, BIT_AND(Score), BIT_OR(Score)
    ->   FROM grouptest
    ->  GROUP BY Team;
+------+----------------+---------------+
| Team | BIT_AND(Score) | BIT_OR(Score) |
+------+----------------+---------------+
| Blue |              0 |           127 |
| Red  |              0 |           127 |
+------+----------------+---------------+
2 rows in set (0.00 sec)
```

COUNT(столбец)

Функция COUNT() подсчитывает количество элементов группы, не равных NULL. Если нужно определить число записей в таблице, можно вместо имени столбца указать звездочку (*). Предикат DISTINCT позволяет исключить из группы повторяющиеся значения (листинг 12.16).

```
mysql> SELECT Team, COUNT(Score), COUNT(DISTINCT Score)
    ->   FROM grouptest
    ->  GROUP BY Team;
+------+--------------+----------------------+
| Team | COUNT(Score) | COUNT(DISTINCT Score) |
+------+--------------+----------------------+
| Blue |            8 |                    8 |
| Red  |            8 |                    7 |
+------+--------------+----------------------+
2 rows in set (0.01 sec)
```

COUNT([DISTINCT] ...)

Этот вариант функции COUNT() возвращает число уникальных комбинаций столбцов, указанных в списке аргументов (см. листинг 12.16).

MAX(...)

Функция MAX() возвращает значение наибольшего элемента группы (листинг 12.17). Максимальным строковым значением считается то, которое стоит последним по алфавиту. Поскольку даты тоже можно упорядочивать, то максимальной датой считается та, которая идет последней по порядку. Правда, двухзначные номера годов могут сортироваться неправильно, если они не хранятся в столбце типа YEAR. Следовательно, может потребоваться преобразовать их в четырехзначные значения путем добавления нуля с помощью функции DATE_ADD().

```
mysql> SELECT Team, MIN(Score), MAX(Score)
    ->   FROM grouptest
    ->  GROUP BY Team;
+------+------------+------------+
| Team | MIN(Score) | MAX(Score) |
+------+------------+------------+
| Blue |          5 |         91 |
| Red  |         11 |         98 |
+------+------------+------------+
2 rows in set (0.01 sec)
```

MIN(...)

Функция MIN() возвращает значение наименьшего элемента группы (см. листинг 12.17). Минимальным строковым значением считается то, которое стоит первым по алфавиту.

STD(...)

Функция STD() вычисляет среднее отклонение элементов группы (листинг 12.18).

```
mysql> SELECT Team, STD(Score)
    ->   FROM grouptest
    ->  GROUP BY Team;
+------+------------+
| Team | STD(Score) |
+------+------------+
| Blue |    28.7956 |
| Red  |    30.1431 |
+------+------------+
2 rows in set (0.01 sec)
```

STDDEV(...)

Это синоним функции STD().

SUM(...)

Функция SUM() вычисляет сумму элементов группы (листинг 12.19). Строки и даты приводятся к целому типу.

```
mysql> SELECT Team, SUM(Score)
    ->   FROM grouptest
    ->  GROUP BY Team;
+------+------------+
| Team | SUM(Score) |
+------+------------+
| Blue |        470 |
| Red  |        313 |
+------+------------+
2 rows in set (0.00 sec)
```

Математические функции

Описанные ниже функции выполняют различные математические операции. В качестве аргументов большинство из них принимает числа с плавающей запятой и возвращает результат аналогичного типа.

ABS(число)

Эта функция возвращает модуль числа (листинг 12.20).

```
mysql> SELECT ABS(-17);
+----------+
| ABS(-17) |
+----------+
|       17 |
+----------+
1 row in set (0.01 sec)
```

ACOS(число)

Эта функция возвращает арккосинус числа (листинг 12.21). Диапазон допустимых значений — от -1 до 1. Вне этого диапазона значение арккосинуса не определено.

```
mysql> SELECT ACOS(1);
+----------+
| ACOS(1)  |
+----------+
| 0.000000 |
+----------+
1 row in set (0.00 sec)
```

ASIN(число)

Эта функция возвращает арксинус числа (листинг 12.22). Диапазон допустимых значений — от -1 до 1.Вне этого диапазона значениеарксинуса не определено.

```
mysql> SELECT ASIN(1);
+----------+
| ASIN(1)  |
+----------+
| 1.570796 |
+----------+
1 row in set (0.02 sec)
```

ATAN(число)

Эта функция возвращает арктангенс числа (листинг 12.23).

```
mysql> SELECT ATAN(1);
+----------+
| ATAN(1)  |
+----------+
| 0.785398 |
+----------+
1 row in set (0.00 sec)
```

ATAN2(число, число)

Эта функция возвращает угол в радианах точки с заданными координатами.

```
mysql> SELECT ATAN2(3,7);
+------------+
| ATAN2(3,7) |
+------------+
|   0.404892 |
+------------+
1 row in set (0.00 sec)
```

CEILING(число)

Эта функция округляет число до ближайшего большего целого числа (листинг 12.25).

```
mysql> SELECT CEILING(1.3);
+--------------+
| CEILING(1.3) |
+--------------+
|            2 |
+--------------+
1 row in set (0.01 sec)
```

COS(число)

Эта функция возвращает косинус числа в радианах (рис. 12.26).

```
mysql> SELECT COS(1);
+----------+
| COS(1)   |
+----------+
| 0.540302 |
+----------+
1 row in set (0.00 sec)
```

COT(число)

Эта функция возвращает котангенс числа (рис. 12.27).

```
mysql> SELECT COT(1);
+-------------+
| COT(1)      |
+-------------+
| 0.64209262 |
+-------------+
1 row in set (0.02 sec)
```

DEGREES(число)

Эта функция переводит радианы в градусы (листинг 12.28).

```
mysql> SELECT DEGREES(1);
+------------------+
| DEGREES(1)       |
+------------------+
| 57.295779513082 |
+------------------+
1 row in set (0.00 sec)
```

EXP(число)

Эта функция возводит число *e* (основание натурального логарифма) в заданную степень (листинг 12.29).

```
mysql> SELECT EXP(2);
+          +
I EXP(2)   I
+          +
I 7.389056 |
+          +
1 row in set (0.00 sec)
```

FLOOR(число)

Эта функция округляет число до ближайшего меньшего целого числа (листинг 12.30).

```
mysql> SELECT FLOOR(1.7);
+             +
I FLOOR(1.7) |
+             +
I           1 I
+             +
1 row in set (0.00 sec)
```

GREATEST(...)

Эта функция возвращает наибольшее значение из списка (листинг 12.31). Она может работать как с числами, так и со строками.

```
mysql> SELECT GREATEST(1,2,3);
+                 +
| GREATEST(1,2,3) I
+                 +
|               3 |
+                 +
1 row in set (0.00 sec)
```

LEAST(...)

Эта функция возвращает наименьшее значение из списка (листинг 12.32). Она может работать как с числами, так и со строками.

```
mysql> SELECT LEAST(1,2,3);
+-------------+
| LEAST(1,2,3) |
+-------------+
|            1 |
+-------------+
1 row in set (0.00 sec)
```

LOG(число)

Эта функция возвращает натуральный логарифм числа (листинг 12.33).

```
mysql> SELECT LOG(10);
+-----------+
| LOG(10)   |
+-----------+
| 2.302585  |
+-----------+
1 row in set (0.00 sec)
```

LOG10(число)

Эта функция возвращает десятичный логарифм числа (листинг 12.34).

```
mysql> SELECT LOG10(1234);
+-------------+
| LOG10(1234) |
+-------------+
|    3.091315 |
+-------------+
1 row in set (0.01 sec)
```

MOD(число, число)

Эта функция возвращает остаток от деления первого числа на второе (листинг 12.35), подобно оператору %.

```
mysql> SELECT MOD(35, 4);
+-------------+
| MOD(35, 4)  |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)
```

PI()

Эта функция возвращает значение числа л с точностью шесть цифр после запятой (листинг 12.36). Сама программа MySQL хранит это значение в формате с двойной точностью.

```
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
1 row in set (0.00 sec)
```

POW(число, число)

Эта функция возвращает результат возведения первого числа в степень, заданную вторым числом (рис. 12.37).

```
mysql> SELECT POW(2, 10);
+-------------+
| POW(2, 10)  |
+-------------+
| 1024.000000 |
+-------------+
1 row in set (0.00 sec)
```

POWER(число, число)

Это синоним функции POW().

RADIANS(число)

Эта функция преобразует градусы в радианы (листинг 12.38).

```
mysql> SELECT RADIANS(45);
+                  +
I RADIANS(45)      I
+                  +
I 0.78539816339745 I
+                  +
1 row in set  (0.00 sec)
```

RAND([начальное_число])

Эта функция возвращает псевдослучайное число в интервале от 0 до 1 (листинг 12.39). Аргумент функции инициализирует генератор псевдослучайных чисел. Если аргумент отсутствует, используется значение системных часов.

```
mysql> SELECT RAND(12345);
+                  +
I RAND(12345)      |
+                  +
I 0.18113073909761 |
+                  +
1 row in set  (0.00 sec)
```

ROUND(число[, точность])

Эта функция округляет число с плавающей запятой до целого числа или, если указан второй аргумент, до заданного количества цифр после запятой (листинг 12.40).

```
mysql> SELECT ROUND(15.666, 2);
+                  --+
I ROUND(15.666, 2) |
+                  +
I           15.67  |
+                  +
1 row in set  (0.00 sec)
```

SIGN(число)

Эта функция возвращает -1, если число является отрицательным, и 1, если оно неотрицательно (листинг 12.41).

```
mysql> select SIGN(-10);
+-----------+
| SIGN(-10) |
+-----------+
|        -1 |
+-----------+
1 row in set (0.00 sec)
```

SIN(число)

Эта функция возвращает синус числа (листинг 12.43).

```
mysql> SELECT SIN(1);
+----------+
| SIN(1)   |
+----------+
| 0.841471 |
+----------+
1 row in set (0.00 sec)
```

SQRT(число)

Эта функция возвращает квадратный корень числа (листинг 12.43).

```
mysql> SELECT SQRT(15);
+-----------+
| SQRT(15)  |
+-----------+
| 3.872983  |
+-----------+
1 row in set (0.00 sec)
```

TAN(число)

Эта функция возвращает тангенс угла в радианах (листинг 12.44).

```
mysql> SELECT TAN(1);
+----------+
| TAN(1)   |
+----------+
| 1.557408 |
+----------+
1 row in set (0.00 sec)
```

TRUNCATE(число, точность)

Эта функция усекает число до требуемой точности (листинг 12.45).

```
mysql> SELECT TRUNCATE(1.2345, 2);
+---------------------+
| TRUNCATE(1.2345, 2) |
+---------------------+
|                1.23 |
+---------------------+
1 row in set (0.00 sec)
```

Строки

Описанные ниже функции принимают строки в качестве аргументов либо возвращают строки. В MySQL есть также операторы сравнения строк, например LIKE и REGEXP.

ASCII(символ)

Эта функция возвращает ASCII-код первого символа заданной строки (листинг 12.46).

```
mysql> SELECT ASCII ('a');
+-------------+
| ASCII('a') |
+-------------+
|          97 |
+-------------+
1 row in set (0.00 sec)
```

BIN(целое)

Эта функция возвращает двоичное представление заданного целого числа (листинг 12.47).

```
mysql> SELECT BIN(13);
+---------+
| BIN(13) |
+---------+
| 1101    |
+---------+
1 row in set (0.01 sec)
```

BINARY строка

Ключевое слово BINARY объявляет строку двоичной, т.е. операции сравнения с ней будут чувствительными к регистру (листинг 12.48). Это слово имеет более высокий приоритет, чем операторы сравнения.

```
mysql> SELECT 'a'='A', BINARY 'a'='A';
+---------+-----------------+
| 'a'='A' | BINARY 'a'='A' |
+---------+-----------------+
|       1 |               0 |
+---------+-----------------+
1 row in set (0.01 sec)
```

CHAR(...)

Эта функция возвращает строку, заданную в виде списка ASCII-кодов (листинг 12.49).

```
mysql> SELECT CHAR(97,98,99);
+-------------------+
| CHAR(97,98,99) |
+-------------------+
| abc               |
+-------------------+
1 row in set (0.00 sec)
```

CHARACTER_LENGTH(строка)

Это синоним функции CHAR_LENGTH().

CHAR_LENGTH(строка)

Эта функция возвращает количество символов в строке (листинг 12.50). Многобайтовые символы учитываются один раз.

```
mysql> SELECT CHAR_LENGTH('MySQL')
+                        +
I CHAR_LENGTH('MySQL') |

I                     5 |
+                        +
1 row in set (0.00 sec)
```

CONCAT(...)

Эта функция конкатенирует (объединяет) группу строк (листинг 12.51). Если какая-либо строка равна NULL, то и результат будет равен NULL.

```
mysql> SELECT CONCAT('a','b','c')
+                          +
I CONCAT('a','b','c') |
+-------    ------------+
I abc                  |
+                          +
1 row in set (0.00 sec)
```

CONCAT_WS(разделитель, ...)

Эта функция конкатенирует строки, вставляя между ними разделитель (листинг 12.52). В отличие от функции CONCAT(), значения NULL в списке аргументов игнорируются, но если строка-разделитель равна NULL, то и результат будет равен NULL.

```
mysql> SELECT CONCAT_WS('<>', 'a','b','c');
+                                 +
I CONCAT WS('<>', 'a','b','c') |
```

```
+                                  +
| a<>b<>c                          |
+                                  +
1 row in set (0.00 sec)
```

CONV(целое, исходное_основание, конечное_основание)

Эта функция преобразует целое число из одной системы счисления в другую. С ее помощью можно дублировать работу функций BIN(), HEX() и OCT(). В листинге 12.53 показан перевод числа 100 из восьмеричной в десятичную систему.

```
mysql> SELECT CONV('100', 8, 10);
+                                +
| CONV('100', 8, 10) |
+                                +
| 64                 |
+                                +
1 row in set (0.00 sec)
```

DECODE(зашифрованный_текст, пароль)

Эта функция расшифровывает строку, созданную функцией ENCODE() (листинг 12.54).

```
mysql> SELECT DECODE(ENCODE('MySQL', 'password'), 'password');
+                                                +
| DECODE(ENCODE('MySQL', 'password'), 'password') |
+                                                +
| MySQL                                          |
+                                                +
1 row in set (0.01 sec)
```

ELT(индекс, ...)

Эта функция возвращает элемент списка с указанным индексом (листинг 12.55). Нумерация элементов начинается с единицы.

```
mysql> SELECT ELT(3, 'a','b','c','d');
+                                    +
I ELT(3, 'a','b','c','d') |

I c                                  I
+                                    +
1 row in set  (0.00 sec)
```

ENCODE(обычный_текст, пароль)

Эта функция возвращает строку, зашифрованную с помощью заданного пароля (листинг 12.56). Полученная строка является двоичной и имеет ту же длину, что и оригинал. Для расшифровки строки предназначена функция DECODE().

```
mysql> UPDATE user
    ->    SET password = ENCODE('secret', 'pass')
    ->  WHERE ID=1;
Query OK, 1 row affected (0.00 sec)
Rows matched 1 Changed 1 Warnings 0
```

Не применяйте данную функцию для шифрования столбца паролей в таблице привилегий MySQL. Этой цели служит функция PASSWORD().

ENCRYPT(строка[, примесь])

Эта функция является оболочкой функции crypt() языка C. Она реализует алгоритм необратимого шифрования (листинг 12.57). Вторым аргументом может быть **двухсимвольная** строка, повышающая степень случайности шифрования. Более длинные строки усекаются.

Эта функция несовместима с функцией PASSWORD(). Кроме того, в каждой операционной системе может быть своя реализация функции crypt().

```
mysql> SELECT ENCRYPT('password', 'ab');
+                                    +
I ENCRYPT('password', 'ab') |
+                                    +
I abJnggxhB/yWI  .                   |
+                                    +
1 row in set (0.00 sec)
```

EXPORT_SET(битовое_поле,строка _единицы, строка_нуля[, разделитель[, счетчик_битов]])

Эта функция возвращает строку флагов, соответствующих значениям битов первого аргумента. Биты интерпретируются от младшего к старшему, а сама строка строится слева направо. Во втором и третьем аргументах задаются строки, которые подставляются при обнаружении соответственно единичного и нулевого бита.

По умолчанию разделителем служит запятая, но эту установку можно менять. Кроме того, по умолчанию анализируются все 64 бита целого числа, но в пятом аргументе можно задать максимальное количество битов.

В листинге 12.58 вместо единичных битов подставляется Y, вместо нулевых битов — N, а разделителем служит вертикальная черта. Анализируются первые восемь битов числа 9.

```
mysql> SELECT EXPORT_SET(9, 'Y', 'N', '|', 8);
+--------------------------------+
| EXPORT_SET(9, 'Y', 'N', '|', 8) |
+--------------------------------+
| Y|N|N|Y|N|N|N|N                |
+--------------------------------+
1 row in set (0.00 sec)
```

FIELD(элемент, ...)

Эта функция возвращает индекс указанного элемента в приведенном далее списке (листинг 12.59). Нумерация элементов начинается с единицы. Если элемент не найден, возвращается 0.

```
mysql> SELECT FIELD('b', 'a','b','c','d');
+----------------------------+
| FIELD('b', 'a','b','c','d') |
+----------------------------+
|                          2 |
+----------------------------+
1 row in set (0.00 sec)
```

FIND_IN_SET(элемент, список)

Эта функция возвращает индекс указанного элемента в списке, представляющем собой строку разделенных запятыми элементов (листинг 12.60). Нумерация элементов начинается с единицы.

```
mysql> SELECT FIND_IN_SET('c', 'a,b,c,d');
+-----------------------------+
| FIND_IN_SET('c', 'a,b,c,d') |
+-----------------------------+
|                           3 |
+-----------------------------+
1 row in set (0.00 sec)
```

FORMAT(число, точность)

Эта функция возвращает число с указанным количеством цифр после десятичной точки и запятыми, вставленными между группами разрядов (листинг 12.61).

```
mysql> SELECT FORMAT(12345678.909112, 2);
+-----------------------------+
| FORMAT(12345678.909112, 2) |
+-----------------------------+
| 12,345,678.91               |
+-----------------------------+
1 row in set (0.00 sec)
```

HEX(целое)

Эта функция возвращает **шестнадцатеричное** представление целого числа (листинг 12.62).

```
mysql> select HEX(563823);
+-------------+
| HEX(563823) |
+-------------+
| 89A6F       |
+-------------+
1 row in set (0.00 sec)
```

INET_ATON(адрес)

Эта функция преобразует IP-адрес, записанный в строковом виде, в числовую форму (листинг 12.63). Понимаются как 4-байтовые, так и 8-байтовые адреса.

```
mysql> SELECT INET_ATON('64.28.67.70');
+--------------------------+
| INET_ATON('64.28.67.70') |
+--------------------------+
|               1075594054 |
+--------------------------+
1 row in set (0.00 sec)
```

INET_NTOA(адрес)

Эта функция возвращает строковое представление IP-адреса, записанного в числовом виде (листинг 12.64).

```
mysql> SELECT INET_NTOA('1075594054');
+-------------------------+
| INET_NTOA('1075594054') |
+-------------------------+
| 64.28.67.70             |
+-------------------------+
1 row in set (0.00 sec)
```

INSERT(длина,позиция, длина, подстрока)

Эта функция вставляет в строку заданную подстроку. Второй аргумент определяет позицию вставки, а третий аргумент указывает на то, сколько символов можно затереть, начиная с этой позиции. В листинге 12.65 показана вставка в строку подстроки ABC в позицию 3 с перезаписью одного символа.

```
mysql> SELECT INSERT('abcdefg', 3, 1, 'ABC');
+--------------------------------+
| INSERT('abcdefg', 3, 1, 'ABC') |
+--------------------------------+
| abABCdefg                      |
+--------------------------------+
1 row in set (0.00 sec)
```

INSTR(строка, подстрока)

Эта функция возвращает позицию первого вхождения указанной подстроки в строку (листинг 12.66).

```
mysql> SELECT INSTR('batboy', 'boy');
+-------------------------+
| INSTR('batboy', 'boy') |
+-------------------------+
|                       4 |
+-------------------------+
1 row in set (0.00 sec)
```

LCASE(строка)

Эта функция возвращает строку, все символы которой переведены в нижний регистр (листинг 12.67).

```
mysql> SELECT LCASE('AbCd');
+---------------+
| LCASE('AbCd') |
+---------------+
| abcd          |
+---------------+
1 row in set (0.00 sec)
```

LEFT(строка, длина)

Эта функция извлекает из строки подстроку заданной длины (листинг 12.68).

```
mysql> SELECT LEFT('abcdef', 3);
+-------------------+
| LEFT('abcdef', 3) |
+-------------------+
| abc               |
+-------------------+
1 row in set (0.01 sec)
```

LENGTH(строка)

Эта функция возвращает количество байтов в строке (листинг 12.69). Двухбайтовые символы учитываются дважды. Длину строки многобайтовых символов можно узнать с помощью функции CHAR_LENGTH().

```
mysql> SELECT LENGTH('abc');
+---------------+
| LENGTH('abc') |
+---------------+
|             3 |
+---------------+
1 row in set (0.00 sec)
```

LOAD_FILE(имя)

Эта функция возвращает содержимое **файла**, расположенного в файловой системе сервера (листинг 12.7). Пользователь MySQL должен иметь право доступа к этому файлу, а сам файл должен быть доступен для всеобщего чтения.

```
UPDATE messages
SET body=LOAD_FILE('/home/bbs/1234.txt')
WHERE ID=245
```

LOCATE(подстрока, строка[, позиция])

Эта функция находит позицию первого вхождения заданной подстроки в строку. Необязательный третий аргумент определяет начальную позицию поиска (листинг 12.71).

```
mysql> SELECT LOCATE('b', 'abcabc', 3);
+--------------------------+
| LOCATE('b', 'abcabc', 3) |
+--------------------------+
|                        5 |
+--------------------------+
1 row in set (0.00 sec)
```

LOWER(строка)

Это синоним функции LCASE().

LPAD(строка,длина, заполнитель)

Эта функция дополняет строку до указанной длины, вставляя слева строку-заполнитель. В случае необходимости строка-заполнитель будет продублирована (листинг 12.72).

```
mysql> SELECT LPAD('abc', 15, '.');
+----------------------+
| LPAD('abc', 15, '.') |
+----------------------+
| ............abc      |
+----------------------+
1 row in set (0.00 sec)
```

LTRIM(строка)

Эта функция удаляет из строки начальные пробелы (листинг 12.73).

```
mysql> SELECT LTRIM('     abc');
+--------------------+
| LTRIM('     abc') |
+--------------------+
| abc                |
+--------------------+
1 row in set (0.00 sec)
```

MAKE_SET(битовое_поле, ...)

Эта функция возвращает строку-список, созданную путем выбора элементов заданного списка на основании флагов битового поля. Единичный бит означает выбор элемента, индекс которого соответствует номеру бита. В листинге 12.74 двоичное представление первого аргумента (число *5)* равно 0101, поэтому из списка выбираются первый и третий элементы.

```
mysql> SELECT MAKE_SET(5, 'a', 'b', 'c');
+----------------------------+
| MAKE_SET(5, 'a', 'b', 'c') |
+----------------------------+
| a,c                        |
+----------------------------+
1 row in set (0.00 sec)
```

MATCH (...) AGAINST (строка)

Конструкция MATCH сравнивает заданную строку со списком столбцов и возвращает число в интервале от 0,0 до 1,0 (листинг12.75). Для перечисленных столбцов должен существовать индекс типа FULLTEXT (см. главу 11, "Типы столбцов и индексов"). Программа MySQL разбивает строку на слова, разделенные пробелами, причем слова длиной три и менее символов игнорируются. Слова могут быть заключены в кавычки.

Конструкция MATCH вызывает сортировку записей по релевантности в порядке убывания. Записи с нулевой релевантностью не включаются в результаты запроса.

```
SELECT ID
FROM message
WHERE MATCH (body) AGAINST ('MySQL');
```

MD5(строка)

Эта функция возвращает **32-символьный хэш-код**, правило построения которого описано в документе RFC 1321 (листинг 12.76). Идентификаторы MD5 теоретически являются уникальными для всех строк.

```
mysql> SELECT MD5('Who is John Galt?')
+----------------------------------+
| MD5('Who is John Galt?')         |
+----------------------------------+
| bebcd5657c9c3d62f9e22f2e0730868a |
+----------------------------------+
1 row in set (0.04 sec)
```

MID(строка, позиция, длина)

Это синоним функции SUBSTRING().

OCT(целое)

Эта функция возвращает восьмеричное представление целого числа (листинг 12.77).

```
mysql> SELECT OCT(16);
+---------+
| OCT(16) |
+---------+
| 20      |
+---------+
1 row in set (0.00 sec)
```

OCTET_LENGTH(строка)

Это синоним функции LENGTH().

ORD(строка)

Эта функция возвращает порядковый номер самого левого символа строки. В отличие от функции ASCII(),функция ORD() работает и с многобайтовыми символами.

PASSWORD(строка)

Эта функция шифрует пароль, заданный в текстовом виде (листинг 12.78). Процесс шифрования является необратимым. Функция PASSWORD() предназначена для задания паролей в файле mysql.user, где содержится таблица пользовательских привилегий.

Листинг

```
UPDATE user
SET Password=PASSWORD('secret')
WHERE User='leon'
```

POSITION(подстрока IN строка)

Это альтернативный вариант функции LOCATE() (листинг 12.79).

```
mysql> SELECT POSITION('b' IN 'abcabc');
+---------------------------+
| POSITION('b' IN 'abcabc') |
+---------------------------+
|                         2 |
+---------------------------+
1 row in set (0.00 sec)
```

REPEAT(строка, счетчик)

Эта функция возвращает строку, которая состоит из заданной строки, повторяющейся указанное число раз (листинг 12.80).

```
mysql> SELECT REPEAT('a', 10);
+-----------------+
| REPEAT('a', 10) |
+-----------------+
| aaaaaaaaaa      |
+-----------------+
1 row in set (0.00 sec)
```

REPLACE(строка, старая_подстрока, новая подстрока)

Эта функция меняет в исходной строке каждое вхождение старой подстроки на новую подстроку (листинг 12.81).

```
mysql> SELECT REPLACE('a-b-c-d', '-', '/');
+-----------------------------+
| REPLACE('a-b-c-d', '-', '/') |
+-----------------------------+
| a/b/c/d                     |
+-----------------------------+
1 row in set (0.00 sec)
```

REVERSE(строка)

Эта функция меняет порядок символов в строке на обратный (листинг 12.82).

```
mysql> SELECT REVERSE('abcdef');
+-------------------+
| REVERSE('abcdef') |
+-------------------+
| fedcba            |
+-------------------+
1 row in set (0.00 sec)
```

RIGHT(строка, счетчик)

Эта функция возвращает указанное число символов строки, считая с ее правого конца (листинг 12.83). Функция LEFT () работает с противоположного конца.

```
mysql> SELECT RIGHT('abcdef', 3);
+--------------------+
| RIGHT('abcdef', 3) |
+--------------------+
| def                |
+--------------------+
1 row in set (0.00 sec)
```

RPAD(строка, длина, заполнитель)

Эта функция дополняет строку до указанной длины, вставляя справа строку-заполнитель (листинг 12.84). Функция LPAD () дополняет строку слева.

```
mysql> SELECT RPAD('abc', 15, '.');
+-----------------------+
| RPAD('abc', 15, '.') |
+-----------------------+
| abc.............      |
+-----------------------+
1 row in set (0.00 sec)
```

RTRIM(строка)

Эта функция удаляет из строки конечные пробелы. В листинге 12.85 строка очищается от хвостовых пробелов, после чего конкатенируется с другой строкой.

```
mysql> SELECT CONCAT(RTRIM('abc  '), 'def');
+-------------------------------+
| CONCAT(RTRIM('abc  '), 'def')  |
+-------------------------------+
| abcdef                        |
+-------------------------------+
1 row in set (0.00 sec)
```

SOUNDEX(строка)

Эта функция возвращает хэш-код, основанный на особенностях звучания слов строки. Данный алгоритм описан Дональдом Кнутом в третьем томе книги "Искусство программирования". Хэш-код состоит из четырех символов и начинается с буквы. В листинге 12.86 показано получение хэш-кодов двух слов, близких по звучанию.

```
mysql> SELECT SOUNDEX('lion'), SOUNDEX('lying');
+-----------------+------------------+
| SOUNDEX('lion') | SOUNDEX('lying') |
+-----------------+------------------+
| L500            | L520             |
+-----------------+------------------+
1 row in set (0.00 sec)
```

SPACE(счетчик)

Эта функция возвращает строку, состоящую из указанного количества пробелов (листинг 12.87). То же самое можно сделать с помощью функции REPEAT().

```
mysql> SELECT CONCAT('a', SPACE(10), 'b');
+-----------------------------+
| CONCAT('a', SPACE(10), 'b') |
+-----------------------------+
| a          b                |
+-----------------------------+
1 row in set (0.00 sec)
```

STRCMP(строка, строка)

Эта функция сравнивает две строки, возвращая 0, если строки равны, -1, если первая строка предшествует второй по алфавиту, и 1, если вторая строка предшествует первой (листинг 12.88).

```
mysql> SELECT STRCMP('abc', 'abd');
+----------------------+
| STRCMP('abc', 'abd') |
+----------------------+
|                   -1 |
+----------------------+
1 row in set (0.00 sec)
```

SUBSTRING(CTpOKa FROM позиция [FORдлина])

Эта функция извлекает из строки подстроку заданной длины, начиная с указанной позиции. В данной версии аргументы разделяются ключевыми словами, а не запятыми, что соответствует стандарту ANSI.

SUBSTRING(строка,позиция[,длина])

Эта функция извлекает из строки подстроку заданной длины, начиная с указанной позиции (листинг 12.89). Нумерация символов строки начинается с единицы. Если длина подстроки не задана, возвращается остаток строки.

```
mysql> SELECT SUBSTRING('abcdef', 3, 2);
+---------------------------+
| SUBSTRING('abcdef', 3, 2) |
+---------------------------+
| cd                        |
+---------------------------+
1 row in set (0.00 sec)
```

SUBSTRING_INDEX(строка, разделитель, счетчик)

Эта функция возвращает подстроку, которая содержит заданное количество разделителей. Если счетчик является положительным числом, подстрока извлекается слева, в противном случае — справа. В листинге 12.90 функция возвращает первые два элемента списка, компоненты которого разделены запятыми.

```
mysql> SELECT SUBSTRING_INDEX('a,b,c,d', ',', 2);
+-----------------------------------+
| SUBSTRING_INDEX('a,b,c,d', ',', 2) |
+-----------------------------------+
| a,b                               |
+-----------------------------------+
1 row in set (0.00 sec)
```

TRIM([[BOTH| LEADING \ TRAILING] заполнитель FROM] строка)

Эта функция удаляет из строки символы-заполнители. По умолчанию удаляются начальные и конечные пробелы (листинг 12.91).

```
mysql> SELECT TRIM('   abc   ');
+-------------------+
| TRIM('   abc   ') |
+-------------------+
| abc               |
+-------------------+
1 row in set (0.00 sec)
```

UCASE(строка)

Эта функция возвращает строку, все символы которой переведены в верхний регистр (листинг 12.92). Обратное преобразование осуществляется с помощью функции LCASE().

```
mysql> SELECT UCASE('AbCd');
+---------------+
| UCASE('AbCd') |
+---------------+
| ABCD          |
+---------------+
1 row in set (0.00 sec)
```

UPPER(строка)

Это синоним функции UCASE ().

Функции работы с датой и временем

Описанные ниже функции работают со значениями даты/времени. Будучи извлеченными из базы данных, эти значения приводятся к целочисленному или строковому типу, в зависимости от контекста. Например, функция NOW() по умолчанию возвращает строку вида "2001-04-20 12:59:58", но может вернуть и число вида 200104200125958, если по контексту требуется целое число.

Любая функция, ожидающая значение даты или времени, понимает значение, в котором указана как дата, так и время.

ADDDATE(дата, INTERVAL значение тип)

Это синоним функции DATE_ADD ().

CURDATE()

Эта функция возвращает значение текущей даты (листинг 12.93). Она эквивалентна функции CURRENT_DATE.

```
mysql> SELECT CURDATE();
+------------+
| CURDATE()  |
+------------+
| 2001-04-20 |
+------------+
1 row in set (0.00 sec)
```

CURRENT_DATE

Эта функция возвращает значение текущей даты (листинг 12.94). Она эквивалентна функции CURDATE(), но не требует скобок.

```
mysql> SELECT CURRENT_DATE;
+--------------+
| CURRENT_DATE |
+--------------+
| 2001-04-20   |
+--------------+
1 row in set (0.01 sec)
```

CURRENT_TIME

Эта функция возвращает значение текущего времени (листинг 12 .95). Она эквивалентна функции CURTIME(), но не требует скобок.

```
mysql> SELECT CURRENT_TIME;
+--------------+
| CURRENT_TIME |
+--------------+
| 13:11:10     |
+--------------+
1 row in set (0.00 sec)
```

CURRENT_TIMESTAMP

Эта функция возвращает значение текущих даты и времени (листинг 12.96). Она эквивалентна функции NOW O, но нетребует скобок.

```
mysql> SELECT CURRENT_TIMESTAMP;
+---------------------+
| CURRENT_TIMESTAMP   |
+---------------------+
| 2001-04-20 13:12:00 |
+---------------------+
1 row in set (0.00 sec)
```

CURTIME()

Эта функция возвращает значениетекущего времени (листинг 12.97). Она эквивалентна функции CURRENT_TIME.

```
mysql> SELECT CURTIME();
+-----------+
| CURTIME() |
+-----------+
| 13:14:18  |
+-----------+
1 row in set (0.00 sec)
```

DATE_ADD(дата, INTERVAL значение тип)

Эта функция добавляет значение времени к значению даты или времени. Прибавляемое значение задается после ключевого слова INTERVAL с указанием требуемого типа данных. Возможные типы описаны в табл. 12.1.

Тип	*Формат*
DAY	Дни
DAY_HOUR	**'дни часы'**
DAY_MINUTE	**'дни**
DAY_SECOND	**'дни часы:минуты:секунды'**
HOUR	Часы
HOUR_MINUTE	**'часы:минуты'**
HOUR_SECOND	**'часы:минуты:секунды'**
MINUTE	Минуты
MINUTE_SECOND	**'минуты:секунды'**
MONTH	Месяцы
SECOND	Секунды
YEAR	Годы
YEAR_MONTH	**'годы-месяцы'**

Прибавить к значению даты интервальное значение позволяет оператор +, например NOW + INTERVAL 1 DAY. Вычесть интервальное значение можно с помощью оператора - или функции DATE_SUB(). Если какая-то часть интервального значения пропущена, программа MySQL будет интерпретировать его справа налево и присваивать пропущенным компонентам 0. Например, спецификация 1:2для интервала типа DAY_MINUTE эквивалентна использованию интервала **типа** MINUTE_SECOND.

В листинге 12.98 вычисляется текущее время и время через две недели.

```
mysql> SELECT NOW(), DATE_ADD(NOW(), INTERVAL 14 DAY);
+---------------------+-----------------------------------+
| NOW()               | DATE_ADD(NOW(), INTERVAL 14 DAY)  |
+---------------------+-----------------------------------+
| 2001-04-20 13:36:27 | 2001-05-04 13:36:27               |
+---------------------+-----------------------------------+
1 row in set (0.01 sec)
```

DATE_FORMAT(дата, формат)

Эта функция форматирует значение даты в соответствии с заданной спецификацией. Строка формата может содержать произвольное число кодов, начинающихся с символа % и обозначающих тот или иной компонент даты. Остальные символы попадают в возвращаемую строку без изменения.

Код	Описание	Примеры
%%	Литеральный символ %	%
%a	Сокращенное название дня недели	Sun...Sat
%b	Сокращенное название месяца	Jan... Dec
%c	Номер месяца без ведущего нуля	1...12
%D	Номер дня месяца с англ ийским суффиксом	1st, 2nd, 3rd, 4th,
%d	Номер дня месяца с ведущим нулем	01...31
%e	Номер дня месяца	1...31
%H	Номер часа в 24-часовом формате с ведущим нулем	01...23
%h	Номер часа в 12-часовом формате с ведущим нулем	01...12
%I	То же, что и %h	01... 12
%i	Число минут с ведущим нулем	00...59
%j	Номер дня года с ведущими нулями	001...366
%k	Номер часа в 24-часовом формате без ведущего нуля	1...23
%l	Номер часа в 12-часовом формате без ведущего нуля	1...12
%M	Название месяца	January...December
%m	Номер месяца с ведущим нулем	01... 12
%p	Обозначение периода суток	AM, PM
%r	Время в 12-часовом формате	01:15:30 AM
%S	Число секунд	00...59
%s	Числосекунд	00... 59

Код	*Описание*	*Примеры*
%T	Время в 24-часовом формате	15:32:00
%u	Номер недели года, в котором первым днем недели является понедельник	0...53
%U	Номер недели года, в котором первым днем недели является воскресенье	0...53
%v	Номер недели года, в котором первым днем недели является понедельник; используется совместно с кодом %x	1. . .53
%V	Номер недели года, в котором первым днем недели является воскресенье; используется совместно с кодом %X	1...53
%w	Номер дня недели	0...6
%W	Название дня недели	Sunday...Saturday
%x	Год, первым днем которого является понедельник; используется совместно с кодом %v	0000...9999
%X	Год, первым днем которого является воскресенье; используется совместно с кодом %V	0000...9999
%y	Год столетия	00...99
%Y	Год	0000...9999

Коды %v, %V, %x и %X работают по тому принципу, что год должен начинаться либо с воскресенья, либо с понедельника. В соответствии с этой логикой 1 января 1970 г. относилось к 53-й неделе 1969 г.

В листинге 12.99 значение даты (1 января 1970г.) отформатировано так, чтобы отображалось название дня.

```
mysql> SELECT DATE_FORMAT('1970-01-01','%W, %M %D, %Y');
+-----------------------------------------+
| DATE_FORMAT('1970-01-01','%W, %M %D, %Y') |
+-----------------------------------------+
| Thursday, January 1st, 1970               |
+-----------------------------------------+
1 row in set (0.00 sec)
```

DATE_SUB(дата, INTERVAL значение тип)

Эта функция вычитает значение времени из значения даты. Способ ее использования такой же, как и у функции DATE_ADD(). Аналогичную операцию реализует оператор -.

DAYNAME(дата)

Эта функция возвращает название дня, соответствующего заданной дате (листинг 12.100).

```
mysql> SELECT DAYNAME('1970-01-01')
+-----------------------+
| DAYNAME('1970-01-01') |
+-----------------------+
| Thursday              |
+-----------------------+
1 row in set (0.00 sec)
```

DAYOFMONTH(дата)

Эта функция возвращает номер дня месяца, соответствующий заданной дате (листинг 12.101).

```
mysql> SELECT DAYOFMONTH('1970-01-01');
+--------------------------+
| DAYOFMONTH('1970-01-01') |
+--------------------------+
|                        1 |
+--------------------------+
1 row in set (0.00 sec)
```

DAYOFWEEK(дата)

Эта функция возвращает номер дня недели, соответствующий заданной дате (листинг 12.102). Первым днем недели считается воскресенье.

```
mysql> SELECT DAYOFWEEK('1970-01-01');
+-------------------------+
| DAYOFWEEK('1970-01-01') |
+-------------------------+
```

```
|                          5 |
+----------------------------+
1 row in set (0.00 sec)
```

DAYOFYEAR(дата)

Эта функция возвращает номер дня с начала года (листинг 12.103). Первым днем считается 1 января.

```
mysql> SELECT DAYOFYEAR('1984-05-06 01:30:00');
+-----------------------------------+
| DAYOFYEAR('1984-05-06 01:30:00') |
+-----------------------------------+
|                               127 |
+-----------------------------------+
1 row in set (0.00 sec)
```

EXTRACT(тип FROM дата)

Эта функция извлекает иззначения даты значение указанного типа (листинг 12.104). Названия типов были перечислены в табл. 12.1.

```
mysql> SELECT EXTRACT(YEAR FROM '1970-01-01');
+---------------------------------+
| EXTRACT(YEAR FROM '1970-01-01') |
+---------------------------------+
|                            1970 |
+---------------------------------+
1 row in set (0.00 sec)
```

FROM_DAYS(дата)

Эта функция вычисляет дату по количеству дней, прошедших с начала летоисчисления (листинг 12.105). Правда, даты до 1582 г. определяются неправильно, поскольку MySQL не учитывает изменения, связанные с появлением григорианского календаря. Обратное преобразование выполняет функция `TO_DAYS()`.

```
mysql> SELECT FROM_DAYS(719528);
+------------------+
| FROM_DAYS(719528) |
+------------------+
| 1970-01-01       |
+------------------+
1 row in set (0.00 sec)
```

FROM_UNIXTIME(секунды[, формат])

Эта функция вычисляет дату по количеству секунд, прошедших с начала эпохи UNIX (1 января 1970 г.). Необязательный аргумент задает формат отображаемой строки. Коды формата были перечислены в табл. 12.2.

Функция FROM_UNIXTIME() учитывает время по Гринвичу. Например, в листинге 12.106 результат получен в системе, находящейся в тихоокеанском часовом поясе (разница по Гринвичу — минус 8 часов).

```
mysql> SELECT FROM_UNIXTIME(28800)
+---------------------+
| FROM_UNIXTIME(28800) |
+---------------------+
| 1970-01-01 00:00:00  |
+---------------------+
1 row in set (0.00 sec)
```

HOUR(время)

Эта функция возвращает номер часа, соответствующий заданному времени (листинг 12.107).

```
mysql> SELECT HOUR('01:23:45');
+-----------------+
| HOUR('01:23:45') |
+-----------------+
|               1 |
+-----------------+
1 row in set (0.00 sec)
```

MINUTE(время)

Эта функция возвращает число минут, соответствующее заданному времени (листинг 12.108).

```
mysql> SELECT MINUTE('01:23:45');
+--------------------+
| MINUTE('01:23:45') |
+--------------------+
|                 23 |
+--------------------+
1 row in set (0.01 sec)
```

MONTH(дата)

Эта функция возвращает номер месяца, соответствующий заданной дате (листинг 12.109).

Листинг 12.109. Функция MO

```
mysql> SELECT MONTH('1970-01-01');
+---------------------+
| MONTH('1970-01-01') |
+---------------------+
|                   1 |
+---------------------+
1 row in set (0.00 sec)
```

MONTHNAME(дата)

Эта функция возвращает название месяца, соответствующее заданной дате (листинг 12.110).

```
mysql> SELECT MONTHNAME('1970-01-01');
+-------------------------+
| MONTHNAME('1970-01-01') |
+-------------------------+
| January                 |
+-------------------------+
1 row in set (0.00 sec)
```

NOW()

Эта функция определяет текущие дату и время. Формат возвращаемого значения соответствует контексту (листинг 12.111).

```
mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2001-04-20 17:20:26 |
+---------------------+
1 row in set (0.00 sec)
```

PERIOD_ADD(период, месяцы)

Эта функция прибавляет указанное число месяцев к заданному периоду, который представляет собой обозначение номера месяца конкретного года. Пример, показанный в листинге 12.112, говорит о том, что через 15 месяцев после января 1970 г. будет апрель 1971 г.

```
mysql> SELECT PERIOD_ADD(197001, 15);
+-------------------------+
| PERIOD_ADD(197001, 15) |
+-------------------------+
|                  197104 |
+-------------------------+
1 row in set (0.00 sec)
```

PERIOD_DIFF(период1, период2)

Эта функция определяет разницу в месяцах между двумя периодами (листинг 12.113). Период — это номер месяца конкретного года.

```
mysql> SELECT PERIOD_DIFF(197001, 197104);
+-----------------------------+
| PERIOD_DIFF(197001, 197104) |
+-----------------------------+
|                         -15 |
+-----------------------------+
1 row in set (0.00 sec)
```

QUARTER(дата)

Эта функция определяет квартал года, соответствующий заданной дате (листинг 12.114). К первому кварталу относятся первые три месяца года.

```
mysql> SELECT QUARTER('1970-01-01');
+-----------------------+
| QUARTER('1970-01-01') |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

SECOND(время)

Эта функция возвращает число секунд, соответствующее заданному времени (листинг 12.115).

```
mysql> SELECT SECOND('01:23:45');
+--------------------+
| SECOND('01:23:45') |
+--------------------+
|                 45 |
+--------------------+
1 row in set (0.00 sec)
```

SEC_TO_TIME(секунды)

Эта функция вычисляет время по указанному количеству секунд (листинг 12.116). Значение времени не обязательно находится в пределах суток. Так, если число секунд будет превышать 86400, то количество часов окажется большим, чем 24. Функция `TIME_TO_SEC()` выполняет обратное преобразование.

```
mysql> SELECT SEC_TO_TIME(5025);
+-------------------+
| SEC_TO_TIME(5025) |
+-------------------+
| 01:23:45          |
+-------------------+
1 row in set (0.00 sec)
```

SUBDATE(дата,INTERVAL значение тип)

Это синоним функции DATE_SUB().

SYSDATE()

Это синоним функции NOW().

TIME_FORMAT(время, формат)

Эта функция возвращает значение времени, отформатированное в соответствии с заданной спецификацией. Допустимые коды формата были перечислены в табл. 12.2. В листинге 12.117 значение времени выдается в 12-часовом формате.

```
mysql> SELECT TIME_FORMAT('23:45:01', '%r');
+-----------------------------+
| TIME_FORMAT('23:45:01', '%r') |
+-----------------------------+
| 11:45:01 PM                 |
+-----------------------------+
1 row in set (0.00 sec)
```

TIME_TO_SEC(время)

Эта функция преобразует заданное значение времени в количество секунд (листинг 12.118). Функция SEC_TO_TIME() выполняет обратное преобразование.

```
mysql> SELECT TIME_TO_SEC('01:23:45');
+-------------------------+
| TIME_TO_SEC('01:23:45') |
+-------------------------+
|                    5025 |
+-------------------------+
1 row in set (0.00 sec)
```

TO_DAYS(дата)

Эта функция вычисляет количество дней, прошедших с начала летоисчисления до указанной даты (листинг 12.119). Даты до 1582 г. определяются неправильно, поскольку MySQL не учитывает изменения, связанные с появлением григорианского календаря. Обратное преобразование выполняет функция FROM_DAYS().

С помощью этой функции можно находить число дней между двумя датами. Например, результатом выражения TO_DAYS ("2001-09-01") -TO_DAYS ("2001-02-01") будет212.

```
mysql> SELECT TO_DAYS('1970-01-01');
+-----------------------+
| TO_DAYS('1970-01-01') |
+-----------------------+
|                719528 |
+-----------------------+
1 row in set  (0.00 sec)
```

UNIX_TIMESTAMP([дата_время])

Эта функция возвращает значение текущего времени в виде метки времени UNIX (листинг 12.120), т.е. количества секунд, прошедших с начала эпохи UNIX (1 января 1970 г., среднее время по Гринвичу). Если указан аргумент, функция вернет метку, соответствующую заданному значению даты/времени.

```
mysql> SELECT UNIX_TIMESTAMP();
+------------------+
| UNIX_TIMESTAMP() |
+------------------+
|        987812906 |
+------------------+
1 row in set  (0.00 sec)
```

WEEK(дата[, первый_день])

Эта функция возвращает номер недели, соответствующий заданной дате (листинг 12.121). Нумерация ведется от первой недели года. По умолчанию первым днем недели считается воскресенье, но если второй аргумент равен 1, то первым днем будет считаться понедельник.

```
mysql> SELECT WEEK('1970-06-01', 0);
+-----------------------+
| WEEK('1970-06-01', 0) |
+-----------------------+
|                    22 |
+-----------------------+
1 row in set (0.00 sec)
```

WEEKDAY(дата)

Эта функция возвращает номер дня недели, соответствующий заданной дате. Понедельник считается днем номер 0. **Пример,** показанный в листинге 12.122, говорит о том, что 6 июня 1970 г. пришлось на субботу.

```
mysql> SELECT WEEKDAY('1970-06-06');
+------------------------+
| WEEKDAY('1970-06-06') |
+------------------------+
|                      5 |
+------------------------+
1 row in set (0.00 sec)
```

YEAR(дата)

Эта функция возвращает номер года, соответствующий заданной дате (листинг 12.123).

```
mysql> SELECT YEAR('1970-06-06');
+--------------------+
| YEAR('1970-06-06') |
+--------------------+
|               1970 |
+--------------------+
1 row in set (0.00 sec)
```

YEARWEEK(дата[, первый_день])

Эта функция возвращает значение, содержащее номер года и номер недели года, соответствующие заданной дате. Второй аргумент определяет день начала недели: воскресенье (0) или понедельник (1). Пример, показанный в листинге 12.124, говорит о том, что 6 июня 1970 г. пришлось на 22-ю неделю года.

```
mysql> SELECT YEARWEEK('1970-06-06', 0);
+---------------------------+
| YEARWEEK('1970-06-06', 0) |
+---------------------------+
|                    197022 |
+---------------------------+
1 row in set (0.00 sec)
```

Прочие функции

Описанные ниже функции не попадают ни в одну из вышеперечисленных категорий.

BIT_COUNT(целое)

Эта функция определяет количество единичных битов в двоичном представлении заданного целого числа (листинг 12.125).

```
mysql> SELECT BIT_COUNT(19), BIN(19);
+---------------+---------+
| BIT_COUNT(19) | BIN(19) |
+---------------+---------+
|             3 | 10011   |
+---------------+---------+
1 row in set (0.00 sec)
```

COALESCE(...)

Эта функция возвращает первый (самый левый) элемент списка, не равный NULL (листинг 12.126). Если все элементы равны NULL, возвращается NULL.

```
mysql> SELECT COALESCE(NULL, NULL, 1, NULL, 2, 3);
+-------------------------------------+
| COALESCE(NULL, NULL, 1, NULL, 2, 3) |
+-------------------------------------+
| 1                                   |
+-------------------------------------+
1 row in set (0.00 sec)
```

INTERVAL(проверяемое_значение, ...)

Эта функция возвращает номер позиции в списке, где проверяемое значение больше предыдущего элемента, но меньше следующего. Элементы списка должны быть отсортированы по возрастанию (при сортировке строк учитывается регистр). Нумерация элементов начинается с единицы.

Пример, показанный в листинге 12.127, говорит о том, что число 9 больше, чем элемент номер 5 (число 7), но меньше, чем элемент номер 6 (число 11).

```
mysql> SELECT INTERVAL(9,1,2,3,5,7,11,13,17);
+-------------------------------+
| INTERVAL(9,1,2,3,5,7,11,13,17) |
+-------------------------------+
|                             5 |
+-------------------------------+
1 row in set (0.00 sec)
```

ISNULL(значение)

Эта функция возвращает 1, если аргумент равен NULL, в противном случае возвращается 0 (листинг 12.128).

```
mysql> SELECT ISNULL(1), ISNULL(null), ISNULL(1+NULL);
+-----------+-------------+----------------+
| ISNULL(1) | ISNULL(null) | ISNULL(1+NULL) |
+-----------+-------------+----------------+
|         0 |            1 |              1 |
+-----------+-------------+----------------+
1 row in set (0.00 sec)
```

Процедуры

Процедуры — это функции, которые выполняются над результатами запроса до того, как эти результаты будут возвращены клиенту. Процедуры вызываются в конце инструкции SELECT. В MySQL включена **одна-единственная** процедура: `analyse()`.

В главе 31, "Расширение возможностей MySQL", рассказывается о том, как писать собственные процедуры.

analyse([число_элементов[, объем_памяти]])

Процедура analyse() анализирует таблицу результатов запроса (листинг 12.129). Можно ограничить максимальное число анализируемых элементов, а также максимальный объем памяти, используемой в процессе анализа.

```
mysql>    SELECT Name, Price
    ->      FROM item
    ->     WHERE ID IN (1,2,3)
    -> PROCEDURE analyse() \G
```

```
*************************** 1. row ***************************
             Field_name:  item.Name
              Min_value:  Brush
              Max_value:  Toothbrush
             Min_length:  4
             Max_length:  10
       Empties_or_zeros:  0
                  Nulls:  0
Avg_value_or_avg_length:  6.3333
                    Std:  NULL
      Optimal_fieldtype:  ENUM('Brush','Comb','Toothbrush') NOT NULL
*************************** 2. row ***************************
             Field_name:  item.Price
              Min_value:  1.25
              Max_value:  3.15
             Min_length:  4
             Max_length:  4
       Empties_or_zeros:  0
                  Nulls:  0
Avg_value_or_avg_length:  2.30
                    Std:  0.79
      Optimal_fieldtype:  ENUM('1.25','2.50','3.15') NOT NULL
2 rows in set (0.00 sec)
```

ИНСТРУКЦИИ SQL

В этой главе.

- Комментарии
- Полный список инструкций

Глава 13

В этой главе описаны инструкции языка SQL. В основном они являются частью стандарта ANSI, а некоторые — расширениями программы MySQL. Имена инструкций нечувствительны к регистру, но рекомендуется набирать их прописными буквами в отличие от пользовательских идентификаторов.

Для каждой инструкции приводится ее прототип. Слова, написанные прописными буквами, являются зарезервированными, как SELECT. Строчными буквами даны параметры, которые могут быть литералами, именами столбцов или таблиц. Элементы, находящиеся в квадратных скобках, считаются необязательными. Иногда допускается несколько уровней необязательных параметров. Символами вертикальной черты разделяются альтернативные варианты выбора.

Комментарии

Комментарии — это текст, сопровождающий инструкции и помогающий читателям разобраться в назначении программного кода. Программа MySQL игнорирует любой текст, стоящий после символа # или двух символов косой черты (//) с последующим пробелом. Игнорируется также текст, заключенный в блок /* и */.

Комментарий, начинающийся с двух дефисов, описан в стандарте SQL. Но разработчики MySQL решили, что обязательным признаком такого комментария должен быть пробел после дефисов. Это позволяет избегать неоднозначности таких конструкций, как, например, вычитание отрицательного числа. Поддержка подобных комментариев помогает переносить в MySQL базы данных, созданные в других СУБД (об этом пойдет речь в главе 28, "Перенос данных в разные СУБД").

В листинге 13.1 показаны три эквивалентных варианта комментариев.

```
SELECT NOW(); # Определяем текущее время
SELECT NOW(); — Определяем текущее время
/*
** Определяем текущее время
*/
SELECT NOW();
```

Полный список инструкций

Ниже описаны все инструкции, поддерживаемые в MySQL.

ALTER TABLE

Инструкция ALTER TABLE позволяет менять определение таблицы. Для этого необходимо иметь привилегии ALTER, CREATE и INSERT. Общий ее формат таков:

```
ALTER [IGNORE] TABLE имя
    спецификация[, спецификация ...]
```

Поскольку программа MySQL способна вносить незаметные изменения в определения таблиц, существует вероятность того, что инструкция ALTER не возымеет никакого эффекта. Подробнее о такого рода изменениях пойдет речь при рассмотрении инструкции CREATE TABLE:

В качестве параметров инструкция ALTER TABLE принимает имя таблицы и как минимум одну спецификацию изменений. Спецификации отделяются друг от друга запятыми, и подобная форма записи является расширением стандарта языка SQL.

Флаг IGNORE заставляет программу MySQL игнорировать дубликаты, если данные, уже находящиеся в таблице, конфликтуют с ее новым определением. Например, когда столбец, содержащий дублирующиеся данные, объявляется первичным ключом, то по умолчанию программа отказывается вносить изменения. При наличии флага IGNORE изменения будут учтены, а все дублирующиеся записи, кроме одной, — выброшены.

В случае добавления нового столбца соответствующие ячейки существующих записей будут заданы равными NULL, если столбец это допускает, или же в них будут записаны значения по умолчанию.

Ниже описаны все возможные варианты спецификаций.

ADD [COLUMN] определение [FIRST | AFTER столбец]

С помощью этой спецификации к таблице добавляется новый столбец. Формат определения столбца должен быть таким же, как и в инструкции CREATE TABLE. По умолчанию столбец добавляется в конец списка, но с помощью ключевого слова FIRST его можно объявить первым, а с помощью предложения AFTER — стоящим после заданного столбца.

В листинге 13.2 демонстрируется добавление двух столбцов, располагаемых в определенном порядке. Возможно, эти столбцы не были учтены при создании таблицы.

Другая причина добавления столбцов — введение первичного ключа. Так, в листинге 13.3 в таблицу вставляется **столбец-счетчик**. В результате каждая строка получит уникальный идентификатор в поле `ID`.

```
ALTER TABLE address
        ADD middleName VARCHAR(32) AFTER firstName,
        ADD prefix VARCHAR(32) FIRST
```

```
ALTER TABLE address
        ADD ID INT NOT NULL AUTO INCREMENT PRIMARY KEY FIRST
```

ADD [COLUMN] (определение, определение, ...)

С помощью этой **спецификации** в конец списка столбцов добавляется группа **новых** столбцов. Определения столбцов разделяются запятыми (листинг 13.4).

```
ALTER TABLE address
        ADD (
            middleName   VARCHAR(32),
            prefix VARCHAR(32)
```

ADD [CONSTRAINT имя] FOREIGN KEY имя (столбец, ...) ссылка

Эта спецификация существует для совместимости с другими СУБД. Факт **существования** столбцов не проверяется, а информация об ограничении не сохраняется в таблице. Разработчики MySQL планируют добавить функции хранения внешних ключей в версию 4.0. Спецификация ссылки рассматривается в разделе, посвященном инструкции CREATE TABLE.

ADD FULLTEXT [имя] (столбец, ...)

Эта спецификация предназначена для добавления к указанным столбцам **полнотекстового** индекса. Об этом рассказывалось в главе 11, "Типы столбцов и индексов".

ADD {KEY | INDEX} [имя] (столбец, ...)

Эта спецификация позволяет добавить индекс к одному или нескольким столбцам (листинг 13.5). Ключевые слова KEY и INDEX являются синонимами. Аналогичную функцию выполняет также инструкция CREATE INDEX.

```
ALTER TABLE address
        ADD INDEX  (lastName)
```

ADD PRIMARY KEY (столбец, ...)

С помощью этой спецификации к **таблице** добавляется первичный ключ (листинг 13.6). У таблицы может быть только один такой ключ, поэтому **существующий** ключ необходимо предварительно удалить.

```
ALTER TABLE address
        ADD PRIMARY KEY(ID)
```

ADD UNIQUE [имя] (столбец, ...)

Эта спецификация налагает на заданные столбцы ограничение уникальности. Если в столбцах содержатся дублирующиеся значения, инструкция ALTER завершится неудачей. Воспользуйтесь флагом IGNORE, чтобы вызвать принудительное изменение.

ALTER [COLUMN] столбец DROP DEFAULT

С помощью этой спецификации из определения столбца удаляется описание значения по умолчанию (листинг 13.7). Для столбца будет выбрано новое стандартное значение на основании его типа и допустимости значений NULL. Если значения NULL разрешены, выбор будет сделан в их пользу. В противном случае будет выбрано значение 0 или пустая строка.

```
ALTER TABLE address
      ALTER prefix DROP DEFAULT
```

ALTER [COLUMN] столбец SET DEFAULT литерал

Эта спецификация назначает столбцу значение по умолчанию (листинг 13.8). Подобное изменение не затрагивает существующие записи.

```
ALTER TABLE address
      ALTER prefix SET DEFAULT 'Mr.'
```

CHANGE [COLUMN] столбец определение

С помощью этой спецификации меняется определение столбца: его имя, размерность и тип. Подобная возможность не предусмотрена в стандарте языка SQL. Другие СУБД тоже допускают модификацию определений столбцов, но программа MySQL является наиболее гибкой в этом плане.

MySQL пытается привести существующие данные к новому типу. Если столбец проиндексирован, его размерность не может стать меньше, чем размерность индекса. Например, наличие индекса первых 16 символов столбца типа VARCHAR означает, что размерность столбца тоже должна составлять не менее 16 символов.

В листинге 13.9 показано, как тип столбца меняется с VARCHAR(32) на ENUM. Обратите внимание на то, что имя столбца повторяется дважды. Это не ошибка. Первый раз идентифицируется существующий столбец, а второй раз дается его новое определение.

```
ALTER TABLE address
     CHANGE prefix
     prefix ENUM('Mr.','Mrs.','Miss','Ms')
```

DROP [COLUMN] столбец

Эта спецификация предназначена для удаления столбца из таблицы и не является частью стандарта языка SQL (листинг 13.10). Индексы, охватывающие удаляемый столбец, автоматически перестраиваются. Если в индекс входил только один этот столбец, индекс удаляется.

```
ALTER TABLE address
       DROP middleName
```

DROP PRIMARY KEY

Эта спецификация аналогична предыдущей, но удаляется не произвольный столбец, а лишь первичный ключ. Если для таблицы не задан первичный ключ, удаляется первый уникальный индекс.

DROP INDEX индекс

С помощью этой спецификации удаляется указанный индекс (листинг 13.11). Подобная возможность не предусмотрена в стандарте.

```
ALTER TABLE address
       DROP INDEX lastName
```

MODIFY [COLUMN] определение

Эта спецификация служит для изменения определения столбца, кроме его имени (листинг 13.12). Данная возможность появилась в MySQL под влиянием СУБД Oracle.

```
ALTER TABLE address
      MODIFY prefix ENUM('Mr.','Mrs.','Miss','Ms')
```

ORDER BY столбец

Данная спецификация предназначена для изменения физического порядка записей по значениям заданного столбца. В некоторых случаях это позволяет ускорить выполнение запросов к таблице. Обратите внимание на то, что переупорядочение записей происходит лишь один раз. Последующие операции вставки и удаления приведут к изменению порядка записей.

RENAME [TO] имя

Эта спецификация позволяет менять имя таблицы (листинг 13.13). Аналогичную функцию выполняет инструкция RENAME TABLE.

```
ALTER TABLE address
      RENAME addr
```

Опции

В инструкции ALTER можно задавать те же самые табличные опции, что и в инструкции CREATE TABLE (см. ниже). Например, разрешается менять тип таблицы или максимальное число записей в ней (листинг 13.14).

```
ALTER TABLE address
       TYPE=BDB
```

ANALYZE TABLE

Инструкция ANALYZE TABLE анализирует и запоминает распределение значений ключей указанной таблицы, что повышает эффективность операций объединения. Синтаксис инструкции таков:

```
ANALYZE TABLE имя[, ИМЯ ...]
```

Эта инструкция выполняет те же действия, что и команда `myisamchk -a`. Разрешается анализировать только таблицы типа MyISAM или BDB.

В листинге 13.15 приведены результаты анализа таблицы пользователей. Если с момента последнего анализа в таблицу не вносились никакие изменения, будет выдано сообщение, стоящее в четвертом столбце.

```
mysql> ANALYZE TABLE user;
+------------+---------+----------+----------------------------+
| Table      | Op      | Msg_type | Msg_text                   |
+------------+---------+----------+----------------------------+
| mysql.user | analyze | status   | Table is already up to date |
+------------+---------+----------+----------------------------+
1 row in set (0.00 sec)
```

BACKUP TABLE

Инструкция BACKUP TABLE записывает информацию о заданных таблицах в указанный каталог. Формат инструкции таков:

```
BACKUP TABLE имя[, имя ...]
         TO каталог
```

Во время выполнения инструкции программа MySQL блокирует таблицу в режиме "только чтение", после чего копирует файлы с расширениями `.MYD` и `.frm` в требуемый каталог и снимает блокировку. За один раз блокируется одна таблица. Чтобы сделать мгновенный снимок таблиц, заблокируйте их с помощью инструкции LOCK TABLES.

Указанный каталог должен быть доступен для записи пользователю, запустившему демон MySQL, а копируемые таблицы должны иметь тип MyISAM. В листинге 13.16 показано копирование таблиц, хранящих информацию о привилегиях.

```
mysql> BACKUP TABLE columns_priv, db, func, host, tables_priv, user
    -> TO '/tmp/backup';
+--------------------+--------+----------+----------+
| Table              | Op     | Msg_type | Msg_text |
+--------------------+--------+----------+----------+
| mysql.columns_priv | backup | status   | OK       |
| mysql.db           | backup | status   | OK       |
| mysql.func         | backup | status   | OK       |
| mysql.host         | backup | status   | OK       |
| mysql.tables_priv  | backup | status   | OK       |
| mysql.user         | backup | status   | OK       |
+--------------------+--------+----------+----------+
6 rows in set (0.11 sec)
```

В MySQL версии 4.0 должна появиться отдельная утилита, выполняющая описанные выше действия.

BEGIN [WORK]

Эта инструкция начинает новую транзакцию. О **транзакциях** рассказывалось в главе 9, "Транзакции и параллельные вычисления".

CHANGE MASTER

Эта инструкция изменяет параметры взаимодействия с главным сервером:

```
CHANGE MASTER TO опция, ...
```

Получив такую инструкцию, подчиненный сервер прекращает работать со старым главным сервером и начинает репликацию — процесс синхронизации с новым сервером. Параметры синхронизации перечислены в табл. 13.1. Необходимо задавать лишь те параметры, которые требуют изменения.

Опция	*Описание*
master_connect_retry	Период ожидания (в секундах) перед повторной попыткой установить соединение
master_host	Доменное имя или IP-адрес главного сервера
master_log_file	Имя журнального файла на главном сервере
master_log_pos	Начальная позиция в журнальном файле
master_password	Пароль для регистрации на главном сервере
master_port	Порт для подключения к главному серверу
master_user	Имя пользователя для регистрации на главном сервере

Инструкция CHANGE MASTER приводит лишь к временной смене главного сервера. Когда подчиненный сервер перезапускается, он руководствуется значениями, содержащимися в файле конфигурации. Если необходимо сменить главный сервер на постоянной основе, нужно остановить подчиненный сервер и внести изменения в его файл конфигурации (листинг 13.17).

```
change master to
master_host='master2.mycompany.com',
master_log_file='master2-bin.001',
master_log_pos=344;
```

CHECK TABLE

Инструкция CHECK TABLE проверяет таблицу на предмет наличия ошибок. Ее синтаксис таков:

```
CHECK TABLE имя[, имя ...]
    [CHANGED] [EXTENDED] [FAST] [MEDIUM] [QUICK]
```

Эта инструкция выполняет те же действия, что и команда `myisamchk -m`. Разрешается проверять только таблицы типа MyISAM. Описание утилиты myisamchk приведено в главе 14, "Утилиты командной строки".

Можно указывать любое число таблиц и произвольные комбинации опций, хотя некоторые из них лишены смысла. Опция CHANGED задает проверку только тех таблиц, которые изменились с момента последней проверки или же были неправильно закрыты. Проверка FAST выполняется над таблицами, которые помечены как неправильно закрытые. Обе эти проверки предназначены для запуска в пакетном режиме, возможно с помощью утилиты-планировщика.

Опция EXTENDED задает проверку каждого элемента каждого индекса. Это занимает гораздо больше времени, зато гарантирует полную согласованность базы данных. Данная проверка предназначена для ситуаций, когда остальные проверки не выявляют ошибок, но в работе базы данных все равно наблюдаются аномалии.

Опция MEDIUM принята по умолчанию. В данном случае проверяется, удалены ли записи, которые помечены для удаления. Проверяются также контрольные суммы записей и ключей. Опция QUICK позволяет избежать проверки плохих ссылок. Оба этих режима подходят для большинства случаев.

В листинге 13.18 демонстрируется проверка таблицы пользователей.

```
mysql> CHECK TABLE user;
+------------+-------+----------+----------+
| Table      | Op    | Msg_type | Msg_text |
+------------+-------+----------+----------+
| mysql.user | check | status   | OK       |
+------------+-------+----------+----------+
1 row in set (0.00 sec)
```

В результате проверки выдается как минимум одна строка, но их может быть и несколько. Если нет ошибок, последняя строка будет содержать сообщение OK. В противном случае таблица может оказаться недоступной для работы, пока не будет исправлена с помощью инструкции REPAIR TABLE.

COMMIT

Инструкция COMMIT объявляет все изменения, сделанные в ходе транзакции, постоянными. О транзакциях рассказывалось в главе 9, "Транзакции и параллельные вычисления".

CREATE DATABASE

Инструкция CREATE DATABASE создает базу данных (листинг 13.19). Синтаксис инструкции таков:

```
CREATE DATABASE [IF NOT EXISTS] имя
```

Если база данных с таким именем существует, а спецификатор IF NOT EXISTS не указан, будет выдано сообщение об ошибке. В MySQL каждая база данных хранится в отдельном подкаталоге, поэтому инструкция создаст пустой каталог. Просмотреть список существующих баз данных можно с помощью инструкции SHOW DATABASE.

```
CREATE DATABASE IF NOT EXISTS freetrade;
```

CREATE FUNCTION

Инструкция CREATE FUNCTION загружает код функции, хранящийся в совместно используемой объектной библиотеке. Эта функция работает так же, как и любая встроенная функция. Синтаксис инструкции таков:

```
CREATE [AGGREGATE] FUNCTION имя
    RETURNS тип
    SONAME библиотека
```

Флаг AGGREGATE **разрешает** использовать **функцию в** предложении GROUP BY. **Тип** возвращаемого **значения** может **быть** STRING, REAL **или** INTEGER. Последний **аргумент инструкции** — это путевое **имя** библиотеки.

Будучи загруженной, функция остается доступной, пока не будет удалена с помощью инструкции DROP FUNCTION. Информация о функции сохраняется в файле `mysql.func`, поэтому необходимо иметь право записи в него. Обычно таким правом обладает только администратор.

В главе 31, "Расширение возможностей MySQL", будет рассказываться о написании собственных функций для MySQL.

CREATE INDEX

Инструкция CREATE INDEX добавляет **индекс к** заданной **таблице:**

```
CREATE [UNIQUE | FULLTEXT] INDEX имя
    ON таблица (столбец [(длина)], ...)
```

То **же** самое можно **сделать с** помощью **инструкции** ALTER TABLE.

CREATE TABLE

Инструкция CREATE TABLE предназначена для создания таблиц. Это, наверное, одна из наиболее сложных SQL-инструкций. Общий ее формат таков:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] имя
      [(спецификация, ...)]
      [опция...]
      [[IGNORE I REPLACE] запрос]
```

Флаг TEMPORARY задает создание временной таблицы, существующей в течение текущего сеанса. По завершении сеанса таблица удаляется. Временным таблицам можно присваивать имена других таблиц, делая последние временно недоступными.

Спецификатор IF NOT EXISTS подавляет вывод сообщений об ошибках в случае, если таблица с указанным именем уже существует. Имени таблицы может предшествовать имя базы данных, отделенное точкой. Если это не сделано, таблица будет создана в базе данных, которая установлена по умолчанию.

Чтобы задать имя таблицы с пробелами, необходимо заключить его в обратные кавычки, например `` `player list` ``. То же самое нужно будет делать во всех ссылках на таблицу, поскольку пробелы используются для разделения идентификаторов.

Разрешается создавать таблицы без столбцов, однако в большинстве случаев спецификация хотя бы одного столбца все же присутствует. Спецификации столбцов и индексов приводятся в круглых скобках и разделяются запятыми. Формат спецификации следующий:

```
имя тип
[NOT NULL | NULL]
[DEFAULT значение]
[AUTO_INCREMENT]
[PRIMARY KEY]
[ссылка]
```

Типы столбцов рассматривались в главе 11, "Типы столбцов и индексов". Спецификация типа включает название типа и его размерность.

По умолчанию столбцы принимают значения NULL. Спецификатор NOT NULL запрещает подобное поведение.

У любого столбца есть значение по умолчанию. Если оно не указано, программа MySQL выберет его самостоятельно. Для столбцов, принимающих значения NULL, значением по умолчанию будет NULL, для строковых столбцов — пустая строка, для целочисленных столбцов — нуль. Изменить эту установку позволяет предложение DEFAULT.

Поля-счетчики, создаваемые с помощью флага AUTO_INCREMENT, игнорируют значения по умолчанию, так как в них записываются порядковые номера. Тип счетчика должен быть беззнаковым целым. В таблице может присутствовать лишь одно поле-счетчик. Им не обязательно является первичный ключ.

В листинге 13.20 показан пример создания таблицы.

```
CREATE TABLE IF NOT EXISTS player (
      /* Столбцы */
      ID INT(11) UNSIGNED NOT NULL AUTO_INCREMENT,
      Nickname CHAR(8) NOT NULL,
      Password CHAR(8) NOT NULL,
      Rank FLOAT(4,2) NOT NULL DEFAULT 50.0,
      Region ENUM('North','South','East','West') NOT NULL,
      Team TINYINT(3) UNSIGNED NOT NULL,
```

```
    /* Индексы */
    PRIMARY KEY (ID),
    INDEX (Region, Rank),
    FOREIGN KEY (Team) REFERENCES team (ID)
)
```

Таблицы типа **MyISAM** и **InnoDB** следят за тем, какие значения генерируются счетчиком, поэтому в случае удаления какой-либо записи ее порядковый номер не будет использован повторно. В таблицах других типов порядковый номер вычисляется путем прибавления единицы к максимальному значению столбца. Если из таблицы удалить все записи, нумерация начнется с единицы.

Чтобы активизировать работу **счетчика,** необходимо вставить в таблицу строку, в которой соответствующее поле равно 0 или NULL. Функция LAST_INSERT_ID(), описанная в главе 12, "Встроенные функции", позволяет узнать последнее значение, сгенерированное счетчиком.

Спецификация PRIMARY KEY позволяет назначить столбец первичным ключом. При этом для столбца будет создан индекс.

В конце определения столбца может стоять предложение REFERENCES, синтаксис которого такой же, как и в описанном ниже предложении FOREIGN KEY, но подобный тип ограничений не поддерживается в MySQL. В версии 3.23 это предложение анализируется синтаксически и отбрасывается. Предполагается, что определения внешних ключей будут сохраняться в версии 4.0.

В круглых скобках задаются спецификации не только столбцов, но также ограничений и индексов. Об индексах рассказывалось в главе 11, "Типы столбцов и индексов", а возможные ограничения описаны ниже.

CHECK (выражение)

Эта спецификация поддерживается для совместимости с другими СУБД, но не несет никакой смысловой нагрузки в MySQL.

[CONSTRAINT имя] FOREIGN KEY имя (столбец, ...) [ссылка]

Эта спецификация тоже не играет никакой роли в MySQL, существуя лишь в целях внешней совместимости. Имеет смысл включать ее в инструкцию CREATE TABLE в качестве "документации" к схеме базы данных. В **принципе,** предложение FOREIGN KEY помечает группу столбцов как зависящую от набора столбцов другой таблицы, формируя связь между **таблицами,** но в MySQL правильность этой связи не контролируется.

Спецификация ссылки имеет следующий вид:

```
REFERENCES таблица [(столбец,...)] [MATCH FULL | MATCH PARTIAL]
        [ON DELETE правило] [ON UPDATE правило]
```

Синтаксис правил удаления и обновления таков:

```
{RESTRICT | CASCADE | SET NULL | NO ACTION I SET DEFAULT}
```

FULLTEXT [INDEX] [имя] (столбец [(длина)], ...)

Эта спецификация задет полнотекстовый индекс для указанных столбцов.

{KEY | INDEX} [имя] (столбец [(длина)], ...)

Эта спецификация задет индекс для указанных столбцов.

PRIMARY KEY (столбец [(длина)], ...)

Эта спецификация делает группу из одного или нескольких столбцов первичным ключом. Первичный ключ не может содержать дублирующиеся значения и значения NULL. У каждой таблицы есть не более чем один первичный ключ.

UNIQUE [INDEX] [имя] (столбец [(длина)], ...)

Эта спецификация накладывает на группу столбцов ограничение уникальности.

Табличные опции

После спецификаций столбцов и индексов может стоять произвольное число опций. Они разделяются пробелами, а не запятыми. В листинге 13.21 демонстрируется задание четырех опций.

```
CREATE TABLE team (
     /* Столбцы */
     ID TINYINT(3) UNSIGNED NOT NULL AUTO_INCREMENT,
     Name CHAR(16) NOT NULL,

     /* Индексы */
     PRIMARY KEY (ID)
)
TYPE=MYISAM
AUTO_INCREMENT=100
AVG ROW LENGTH=9 MAX ROWS=20
```

TYPE = тип

Опция TYPE задает формат хранения таблицы. По умолчанию таблицы имеют тип MyISAM. Для таких таблиц поддерживается большинство функциональных возможностей, за исключением транзакций. Список всех возможных типов представлен в табл. 13.2. Типы HEAP, ISAM, MERGE и MylSAM доступны всегда. Поддержка остальных типов включается на этапе компиляции. Характеристики каждого типа рассматриваются в главе 24, "Физическое хранение данных".

Таблица 13.2. Типы таблиц

Тип	*Описание*
BDB	Таблицы этого типа поддерживают транзакции благодаря библиотеке функций Berkeley DB
Berkeley_db	Синоним типа BDB
HEAP	Таблицы этого типа хранятся в памяти
InnoDB	Таблицы этого типа поддерживают транзакции благодаря библиотеке функций Innobase
ISAM	Этот формат использовался старыми версиями MySQL
MERGE	Это коллекция таблиц MylSAM, интерпретируемых как одно целое
MYISAM	Это стандартный тип таблиц

AUTO_INCREMENT=начальное_значение

Эта опция задает начальное значение поля-счетчика. Она доступналишь для таблиц типа MylSAM.

AVG_ROW_LENGTH = длина

Эта опция помогает программе MySQL создавать указатели записей. Комбинация опций AVG_ROW_LENGTH и MAX_ROWS определяет объем дискового пространства, занимаемый таблицей, и, следовательно, длину указателей.

CHECKSUM = {0 | 1}

Если эта опция включена, программа MySQL будет хранить контрольную сумму каждой записи. Это помогает осуществлять контроль ошибок с помощью инструкции CHECK TABLE. По умолчанию опция отключена. Кроме того, она доступна только для таблиц типа MylSAM.

COMMENT = комментарий

С помощью этой опции таблицу можно снабдить комментарием. Длина комментария не должна превышать 60 символов, но в MySQL версии 4.0 предел возрастет до 255 символов.

MAX_ROWS = число_строк

Эта опция сообщает программе MySQL о максимальном числе записей, которое планируется хранить в таблице. Это нежесткий предел, а лишь подсказка, на основании которой программа выделяет таблице дисковое пространство. Чем меньше данное значение, тем короче будут указатели записей, что ускорит их поиск.

MIN_ROWS = число_строк

Эта опция задает предполагаемое минимальное число записей, которое планируется хранить в таблице.

PACK_KEYS = {0 | 1}

Если эта опция **включена,** программа MySQL будет осуществлять сжатие всех индексов. Обычно сжимаются только индексы столбцов CHAR и VARCHAR, когда их размерность начинает превышать 8 байтов. Это позволяет экономить дисковое пространство при наличии в столбцах большого количества дублирующихся значений.

PASSWORD = пароль

Пароль используется для шифрования определения таблицы, но в открытых версиях MySQL эта возможность не поддерживается. Она доступналишь в коммерческих версиях программы.

DELAY_KEY_WRITE = {0 | 1}

Эта опция заставляет программу MySQL откладывать обновление индексов до того момента, когда таблица будет закрыта. По умолчанию ключи обновляются при каждом изменении. Данная опция доступна только для таблиц типа MyISAM.

ROW_FORMAT = DEFAULT | DYNAMIC | STATIC | COMPRESSED

Эта опция задает формат хранения записей и на момент написания книги еще не реализована.

RAID_TYPE = {1 | STRIPED | RAID0}
RAID_CHUNKS = число_блоков
RAID_CHUNKSIZE = размер

Опции семейства RAID задаются все вместе. Поддержка функций RAID должна быть включена на этапе компиляции. В версии 3.23.36 поддерживается лишь тип `STRIPED`, а остальные два типа определены как его синонимы.

Таблица, для которой установлены эти опции, будет распределена на несколько файлов. Опция RAID_CHUNKS указывает число файлов. Опция RAID_CHUNKSIZE задает размер каждого файла в килобайтах. Когда первый файл заполняется, создается следующий файл. Такая технология позволяла обходить существовавшие когда-то ограничения на размеры файлов, что не актуально в современных операционных системах.

UNION = (таблица, ...)

Опция UNION задает слияние перечисленных таблиц (листинг 13.22).

```
CREATE TABLE marchSales (
     ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
     Item INT,
     Price DECIMAL(6,2)
);
CREATE TABLE aprilSales (
     ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
     Item INT,
     Price DECIMAL(6,2)
);
CREATE TABLE totalSales (
     ID INT NOT NULL,
     Item INT,
     Price DECIMAL(6,2),
     KEY(ID)
) TYPE=MERGE UNION=(marchSales, aprilSales);
```

Запрос на выборку

В конце инструкции CREATE TABLE может находиться инструкция SELECT. Результаты запроса на выборку будут занесены в создаваемую таблицу. Если в самой инструкции CREATE TABLE отсутствуют спецификации столбцов, то вид создаваемой таблицы будет соответствовать таблице результатов запроса. В противном случае столбцы результатов запроса будут добавлены к определенным ранее столбцам.

Флаги IGNORE и REPLACE определяют порядок обработки дублирующихся записей, извлекаемых инструкцией SELECT.

Незаметные изменения

Программа MySQL меняет определения некоторых столбцов. При этом не выдается никаких предупреждений, но если выполнить инструкцию DESCRIBE, можно будет увидеть изменения.

Например, столбцы типа TIMESTAMP должны иметь размерность 6, 8, 12 или 14 символов, поэтому нечетное значение размерности в интервале от 7 до 13 будет преобразовано в ближайшее большее четное число. Кроме того, флаги NULL и NOT NULL для таких столбцов игнорируются, а значения NULL преобразуются в текущее время.

Столбцы типа CHAR и VARCHAR тоже могут подвергаться определенным изменениям, что объясняется особенностями их хранения. Столбцы типа VARCHAR размерностью менее четырех символов будут приведены к типу CHAR. Если в таблице содержатся строки переменной длины, то столбцы типа CHAR размерностью более трех символов приводятся к типу VARCHAR.

В главе 11, "Типы столбцов и индексов", говорилось о том, что у некоторых типов столбцов есть синонимы. Все эти синонимы будут автоматически преобразованы в эквивалентные им типы.

С помощью инструкции SHOW TABLES можно получить список таблиц базы данных, а с помощью инструкции DESCRIBE — просмотреть информацию о заданной таблице.

DELETE

Инструкция DELETE удаляет записи из таблицы. Ее синтаксис таков:

```
DELETE [LOW_PRIORITY] FROM таблица
    [WHERE условия]
    [LIMIT число_записей]
```

Флаг LOW_PRIORITY говорит о том, что операция удаления должна быть отложена до того момента, пока не завершатся все операции чтения таблицы. Предложение WHERE имеет такой же формат, как и в инструкции SELECT.

С помощью предложения LIMIT можно ограничить число удаляемых записей. Это позволяет разбивать длинные операции удаления на ряд более мелких операций, которые проще контролировать. Например, в листинге 13.23 удаляется по 10 записей за раз, ведь всего могут существовать сотни пользователей, которые не регистрировались в системе за прошедший месяц, к тому же, операция сравнения дат тоже занимает определенное время.

```
DELETE LOW_PRIORITY FROM user
 WHERE LastLogin < (NOW() - INTERVAL 30 DAY)
 LIMIT 10
```

Если предложения WHERE и LIMIT отсутствуют, будут удалены все записи таблицы. То же самое делает инструкция TRUNCATE, но она выполняется гораздо быстрее. При удалении записей подобным образом программа MySQL не сообщит об их числе. Если эта информация важна, задайте предложение WHERE, условие отбора которого всегда истинно.

Программа MySQL лишь помечает удаляемую строку как освобожденную. Пока она не будет затерта новой строкой, ее данные останутся на диске. Это повышает производительность за счет менее экономного использования дискового пространства. Инструкция OPTIMIZE TABLE удаляет неиспользуемые строки и восстанавливает правильный формат таблицы.

DESCRIBE

Инструкция DESCRIBE возвращает таблицу, содержащую описание одного или нескольких столбцов заданной таблицы. Общий формат инструкции таков:

```
{DESCRIBE | DESC} таблица [столбец | условие_отбора]
```

Для ленивых: вместо слова DESCRIBE можно указывать его сокращенную форму DESC.

В простейшем случае задается только таблица. В результате будет выдано описание каждого ее столбца. Имени таблицы может предшествовать имя базы данных.

Допускается указывать конкретный столбец или шаблон имени. Шаблон заключается в кавычки и имеет такой же вид, как и в операторе LIKE (см. главу 10, "Типы данных, переменные и выражения"). Аналогичные результаты выдает инструкция SHOW COLUMNS.

В листинге 13.24 приведены результаты инструкции DESCRIBE. В первых двухколонках указаны имя и тип каждого столбца. В колонке "Null" будет стоять YES, если столбец допускает значения NULL. Для столбцов, являющихся частью первичного ключа, в четвертой колонке будет стоять PRI. Если же столбец входит в состав другого индекса, то в этой колонке будет указано MUL. В пятой колонке отображается значение по умолчанию. В последней колонке приводится дополнительная информация о столбце. В частности, здесь указывается, является ли столбец полем-счетчиком.

```
mysql> DESCRIBE mysql.user;
+-----------------+-----------------+----+-----+---------+-------+
| Field           | Type            |Null| Key | Default | Extra |
+-----------------+-----------------+----+-----+---------+-------+
| Host            | char(60) binary |    | PRI |         |       |
| User            | char(16) binary |    | PRI |         |       |
| Password        | char(16) binary |    |     | N       |       |
| Select_priv     | enum('N','Y')   |    |     | N       |       |
| Insert_priv     | enum('N','Y')   |    |     | N       |       |
| Update_priv     | enum('N','Y')   |    |     | N       |       |
| Delete_priv     | enum('N','Y')   |    |     | N       |       |
| Create_priv     | enum('N','Y')   |    |     | N       |       |
| Drop_priv       | enum('N','Y')   |    |     | N       |       |
| Reload_priv     | enum('N','Y')   |    |     | N       |       |
| Shutdown_priv   | enum('N','Y')   |    |     | N       |       |
| Process_priv    | enum('N','Y')   |    |     | N       |       |
| File_priv       | enum('N','Y')   |    |     | N       |       |
| Grant_priv      | enum('N','Y')   |    |     | N       |       |
| References_priv | enum('N','Y')   |    |     | N       |       |
| Index_priv      | enum('N','Y')   |    |     | N       |       |
| Alter_priv      | enum('N','Y')   |    |     | N       |       |
+-----------------+-----------------+----+-----+---------+-------+
17 rows in set (0.01 sec)
```

DROP DATABASE

Инструкция DROP DATABASE удаляет базу данныхиз системы:

DROP DATABASE [IF EXISTS] *имя*

Вместе с базой данных удаляются все таблицы и вся хранимая информация. Спецификация IF EXISTS подавляет вывод сообщения об ошибке, выдаваемого в случае, если указанная база данных не существует.

Базы данных реализованы в виде каталогов, содержащих файлы данных и индексов. Инструкция DROP DATABASE удаляет в заданном каталоге все файлы, созданные программой MySQL. Другие файлы остаются. Если каталог пуст, он тоже удаляется. Таким образом, можно удалить все таблицы, но сама база данных останется, если в ней есть посторонние файлы. Удаление соответствующего каталога файловой системы тоже приводит к удалению базы данных.

Таблицы типа InnoDB нужно удалять вручную.

DROP FUNCTION

Инструкция DROP FUNCTION удаляет из памяти код функции, загруженной с помощью инструкции CREATE FUNCTION:

```
DROP FUNCTION имя
```

DROP INDEX

Инструкция DROP INDEX удаляет индекс таблицы, выполняя для этого инструкцию ALTER TABLE:

```
DROP INDEX имя ON таблица
```

DROP TABLE

Инструкция DROP TABLE удаляет все файлы, относящиеся к таблице. Она имеет следующий синтаксис:

```
DROP TABLE [IF EXISTS] таблица[, таблица ...] [RESTRICT | CASCADE]
```

Спецификация IF EXISTS подавляет вывод сообщения об ошибке, выдаваемого в случае, если заданная таблица не существует. Можно указывать несколько имен таблиц, разделяя их запятыми.

Флаги RESTRICT и CASCADE игнорируются в MySQL версии 3.23. Они предназначены для выполнения сценариев, созданных в других СУБД.

В листинге 13.25 демонстрируется удаление двух таблиц: `message` и log.

```
DROP TABLE IF EXISTS message, log
```

EXPLAIN

Инструкция EXPLAIN описывает способ выполнения указанного запроса или же эмулирует инструкцию DESCRIBE для заданной таблицы:

```
EXPLAIN {запрос | таблица}
```

В качестве запроса разрешается вводить любую допустимую инструкцию SELECT. В результате будет выдана таблица с описанием индексов, используемых для выполнения операции объединения. Благодаря этому можно узнать, влияют ли индексы на повышение производительности объединений. Подробнее об оптимизации запросов рассказывается в главе 26, "Оптимизация". В листинге 13.26 показаны типичные результаты, выдаваемые инструкцией EXPLAIN.

```
mysql> EXPLAIN
    ->     SELECT *
    ->       FROM user, db
    ->      WHERE user.Host = db.Host;
+-------+------+---------------+------+---------+------+------+-------------+
| table | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+-------+------+---------------+------+---------+------+------+-------------+
| user  | ALL  | PRIMARY       | NULL |    NULL | NULL |    5 |             |
| db    | ALL  | PRIMARY       | NULL |    NULL | NULL |    2 | where used  |
+-------+------+---------------+------+---------+------+------+-------------+
2 rows in set (0.16 sec)
```

Таблица, возвращаемая инструкцией EXPLAIN, содержит одну строку для каждой таблицы в порядке их упоминания. В первой колонке приводится имя таблицы, во второй — тип объединения. Возможны следующие типы: ALL, const, eq_ref, index, range, ref, system.

Тип ALL означает полное сканирование таблицы. Обычно для такой таблицы требуется индекс.

Тип const означает, что в таблице есть одна совпадающая запись, а тип system указывает на таблицу, состоящую из одной записи. В обоих случаях программа MySQL может хранить запись в памяти в течение всего времени выполнения запроса, что приведет к повышению производительности.

Тип eq_ref означает ссылку по равенству, т.е. для каждой комбинации записей предыдущих таблиц из данной таблицы извлекается одна запись. В большинстве случаев это наилучший тип с точки зрения производительности.

Тип index означает, что сканированию подлежит весь индекс. Это почти так же плохо, как и тип ALL, но чуть быстрее, поскольку количество элементов индекса обычно меньше, чем число записей в таблице.

Тип range указывает на то, что условию отбора может удовлетворять подмножество записей и каждую из них придется сканировать на каждом проходе.

Тип ref означает, что условию отбора будет удовлетворять подмножество записей, но для ссылки на них используется индекс. Это имеет место в том случае, когда адресуется индексируемый столбец, допускающий дублирующиеся значения.

В колонке possible_keys перечислены индексы, которые могут быть использованы для повышения эффективности запроса. Здесь может стоять NULL.

В колонке key показаны задействованные индексы. Здесь тоже может стоять NULL. В колонке key_len сообщается о том, сколько частей составного ключа используется.

В колонке ref приводится список столбцов или констант, с помощью которых осуществлялся отбор записей. В колонке rows отображается число записей, участвующих в запросе.

В колонке Extra приводится дополнительная информация об объединении. Ключевое слово Distinct говорит о том, что MySQL перестает сканировать записи при обнаружении первого совпадения. Сообщение Not exists означает, что программа оптимизировала левое внешнее объединение, пропустив последующие операции сканирования для совпавшей комбинации. Сообщение range checked for each record выдается, когда

ни один из индексов не устроил программу полностью, т.е. на каждом этапе сканирования используются разные индексы. Сообщение Using `filesort` указывает на необходимость дополнительного прохода для сортировки таблицы результатов. Сообщение Using index означает, что информация о столбцах извлекалась только из индексов, т.е. не понадобилось искать сами записи. Сообщение Using temporary говорит о необходимости создания временной таблицы. Если же в колонке Extra содержится сообщение Where used, значит, для отбора записей было использовано предложение WHERE.

FLUSH

Инструкция FLUSH очищает внутренние кэш-буферы MySQL, используемые для ускорения запросов. После имени инструкции можно указывать произвольное число имен буферов, разделенных запятыми:

FLUSH *буфер*[, *буфер* ...]

Существуют пять очищаемых буферов: HOSTS, LOGS, PRIVILEGES, TABLES и STATUS. Ниже описаны спецификации буферов.

HOSTS

Это кэш имен компьютеров и адресов Internet. Когда компьютер меняет свой IP-адрес, данный буфер нужно очистить. Это также позволит заново установить соединения, которые ранее были отменены из-за превышения лимита ошибок.

LOGS

Очистка данного буфера приводит к закрытию и повторному открытию всех журнальных файлов. Нумерованные файлы получат новые номера.

PRIVILEGES

Очистка данного буфера приведет к повторной загрузке таблиц привилегий в базу данных `mysql`. Это необходимо делать после ручного редактирования таблиц. Инструкции GRANT и REVOKE не требуют очистки буфера.

STATUS

Очистка этого буфера приводит к сбросу большинства значений, сообщаемых инструкцией SHOW TABLE STATUS. Данное действие производит глобальный эффект.

TABLES [имя, ...]

Если после имени TABLES не указан список таблиц, будут закрыты все таблицы, в противном случае — лишь те, что перечислены.

TABLES WITH READ LOCK

Данная спецификация означает закрытие всех таблиц и их блокирование с помощью инструкции LOCK TABLES. Это полезно, когда требуется выполнить резервное копирование всей базы данных. Снять блокировки можно с помощью инструкции UNLOCK TABLES.

GRANT

Инструкция GRANT предоставляет **указанным** пользователям **требуемые привилегии, создавая учетную запись** пользователя **в случае** необходимости. **Общий** формат **инструкции** таков:

```
GRANT тип [(столбец, ...)][, тип [(столбец, ...)] ...]
    ON таблица
    TO пользователь[@узел] [IDENTIFIED BY пароль]
        [, пользователь [IDENTIFIED BY пароль] ...]
    [WITH GRANT OPTION]
```

Возможные **типы привилегий перечислены в табл. 13.3. Привилегии выдаются в** зависимости от контекста: глобально, **для** конкретной **базы данных, таблицы или** столбца. Некоторые **привилегии имеют смысл лишь в** определенном контексте. **К примеру, для** столбцов допустимы **привилегии** INSERT, SELECT **и** UPDATE. **К таблицам и базам данных применимы эти же привилегии, плюс** ALTER, CREATE, DELETE, DROP **и** INDEX. **Следующие привилегии являются** глобальными: FILE, PROCESS, RELOAD и SHUTDOWN.

Тип	*Описание*
ALL [PRIVILEGES]	Пользователю предоставляются все права, кроме права **самостоятельной** передачи привилегий
ALTER	Пользователь может вводить инструкцию ALTER TABLE
CREATE	Пользователь может создавать таблицы и базы данных
DELETE	Пользователь может вводить инструкцию DELETE
DROP	Пользователь может удалять таблицы и базы данных
FILE	Пользователь получает доступ к файлам на локальном диске и может выполнять репликацию
INDEX	Пользователь может добавлять и удалять индексы
INSERT	Пользователь может вводить инструкцию INSERT
PROCESS	Пользователь может просматривать список активных потоков и удалять потоки
REFERENCES	Никак не интерпретируется в версии 3.23
RELOAD	Пользователь может очищать кэш-буферы
SELECT	Пользователь может создавать запросы к таблицам
SHUTDOWN	Пользователь может останавливать работу сервера
UPDATE	Пользователь может вводить инструкцию UPDATE
USAGE	Пользователь не имеет никаких прав и может лишь **регистрироваться** на сервере

Предложение ON и список столбцов задают контекст привилегии. Таблица может идентифицироваться по собственному имени либо в сочетании с именем базы данных. Метасимвол * соответствует всем объектам диапазона. Например, запись `store.*` означает выдачу привилегий для всех таблиц базы данных `store` (листинг 13.27). Отдельный символ * соответствует всем таблицам текущей базы данных. В случае записи `*.*` привилегии являются глобальными.

```
GRANT SELECT, INSERT, UPDATE, DELETE
   ON store.*
   TO gwb@'%.whitehouse.gov'
 WITH GRANT OPTION
```

Имя пользователя — это строка длиной не более 16 символов. Если вслед за этим именем стоят символ @ и сетевое имя, то привилегия применима только тогда, когда пользователь регистрируется на сервере с указанного компьютера. Имена пользователя и компьютера необходимо заключать в кавычки, когда в них присутствуют метасимволы. Например, запись `'leon@%.php.net'` обозначает пользователя `leon`, подключающегося с любого компьютера домена `php.net`. Вместо сетевого имени может стоять и IP-адрес. Если же узел вообще не указан, то информация о том, какому узлу принадлежит пользователь, не будет учитываться.

Предложение IDENTIFIED BY задает пароль пользователя. Если учетная запись пользователя уже существует, его пароль меняется. Задать пароль можно также с помощью инструкции SET.

Спецификация WITH GRANT OPTIONS разрешает пользователю предоставлять аналогичные привилегии другим пользователям.

Инструкция `SHOW GRANTS` выводит список привилегий, которыми владеет пользователь.

INSERT

Инструкция INSERT добавляет записи в таблицу. Ее синтаксис таков:

```
 INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
 [INTO] таблица [(столбец, ...)]
{VALUES (значение,...),(...),... | запрос |
        SET столбец=значение,...}
```

Можно задавать строки в виде набора литеральных значений или в виде таблицы результатов запроса. В первом случае порядок следования значений соответствует порядку столбцов, указанному в определении таблицы либо здесь же, в круглых скобках после имени таблицы. Предложение SET позволяет явно указывать, к какому столбцу относится конкретное значение.

Флаг LOW_PRIORITY говорит о том, что операция вставки должна быть отложена до того момента, пока не будут закончены все операции чтения. Во время самой вставки на таблицу накладывается жесткая блокировка. Поскольку таблицы типа MyISAM допускают одновременные вставки, использовать флаг LOW_PRIORITY нежелательно.

Клиентская программа блокируется до момента завершения инструкции INSERT, что не всегда удобно. Флаг DELAYED отменяет такую установку, помещая инструкцию INSERT в очередь и немедленно возвращая управление программе. Этот флаг применим лишь к таблицам типа **MyISAM**.

Флаг IGNORE подавляет вывод сообщений об ошибках, выдаваемых в случае обнаружения дубликатов. Обычно, если вставляемая запись нарушает целостность **первичного** ключа или ограничение уникальности, отменяется вся операция вставки. При наличии флага IGNORE дублирующаяся запись будет отброшена, и операция **продолжится**. Для замены дубликатов воспользуйтесь инструкцией REPLACE.

Тип вводимого значения должен соответствовать типу столбца, которому оно сопоставлено. Список, приводимый после имени таблицы, может содержать имена столбцов в произвольном порядке. Столбцам, не указанным в списке, будут присвоены значения по умолчанию. Например, в **поле-счетчик** записывается следующее целое число.

После ключевого слова VALUES в скобках приводится набор значений, разделенных запятыми. Разрешается в одном предложении VALUES вводить значения сразу для нескольких записей, что запрещено в стандарте (листинг 13.28).

```
INSERT IGNORE
  INTO team (Name)
VALUES ('Bulldogs'), ('Saints'), ('Giants')
```

В списке значений могут присутствовать выражения. В состав выражений могут входить ссылки на вставляемые столбцы, но лишь на те из них, которые упоминаются раньше. В листинге 13.29 демонстрируется вставка записи с одновременным вычислением текущего года и среднего значений четырех столбцов.

```
INSERT
  INTO performance (FiscalYear, Q1, Q2, Q3, Q4, Mean)
VALUES (NOW(), 0.15, 0.07, 0.13, 0.24, (Q1+Q2+Q3+Q4)/4)
```

В предложении SET указывается список имен столбцов и соответствующих им значений. В этом случае список столбцов после имени таблицы не нужен (листинг 13.30).

```
INSERT IGNORE
  INTO team
   SET Name='Bulldogs'
```

С помощью подчиненной инструкции SELECT можно отобрать группу записей для вставки в таблицу (листинг 13.31). Стандарт языка SQL запрещает ссылаться на одну и ту же таблицу в главной и подчиненной инструкции.

```
INSERT IGNORE
  INTO team (Name)
  SELECT Name
    FROM oldteam o, activelist a
   WHERE o.ID = a.Team
     AND a.Active='Y'
```

KILL

Инструкция KILL разрывает соединение с базой данных. Ей передается идентификатор потока:

KILL *поток*

Все потоки пронумерованы. Инструкция SHOW PROCESSLIST отображает список активных потоков. Обычно пользователь имеет право уничтожать только свои собственные потоки, но с помощью привилегии PROCESS ему можно разрешить удалять любые потоки.

LOCK TABLES

С помощью инструкции LOCK TABLES можно запретить другим потокам осуществлять запись в ту или иную таблицу. Синтаксис инструкции таков:

```
LOCK TABLES таблица [AS псевдоним] {READ [LOCAL] |
                                    [LOW_PRIORITY] WRITE}
[, таблица [AS псевдоним] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
   ...]
```

Инструкция LOCK TABLE ожидает бесконечно долго, пока требуемые блокировки не будут получены. В ней указывается список имен таблиц, разделенных запятыми. После каждого имени задается тип блокировки: READ (нежесткая) или WRITE (жесткая). Нежесткая блокировка запрещает запись в таблицу всем потокам, включая тот, кому она принадлежит. Жесткая блокировка разрешает доступ к таблице для чтения и записи только потоку-владельцу.

В листинге 13.32 показан пример блокирования двух таблиц. Из таблицы team извлекается максимальное значение поля Score, которое заносится в переменную @max. Далее это значение вставляется в таблицу record, после чего обе таблицы разблокируются.

```
LOCK TABLES team t READ, record r WRITE;
SELECT @max:=MAX(Score)
  FROM team t;
INSERT INTO record VALUES (NOW(), @max);
UNLOCK TABLES;
```

Для таблицы может быть задан псевдоним, и пока блокировка не снята, таблице нельзя назначать в запросах другие псевдонимы. Если таблица участвует в запросе под несколькими псевдонимами, ее придется заблокировать определенное число раз под соответствующими именами.

Флаг LOCAL разрешает выполнять инструкции INSERT, не приводящие к конфликтам. Но в этом случае файлы, содержащие табличные данные, становятся доступными для записи, из-за чего появляется угроза нарушения целостности данных.

Флаг LOW_PRIORITY делает приоритет запросов на получение жестких блокировок более низким, чем у запросов на получение нежестких блокировок. В этом случае жесткая блокировка будет предоставлена только тогда, когда нет отложенных нежестких блокировок. Обычно же запросы первого типа имеют более высокий приоритет.

Для снятия блокировок предназначена инструкция UNLOCK TABLES. Кроме того, блокировки снимаются при завершении потока или вводе другой инструкции LOCK TABLES. Подробнее о блокировках рассказывалось в главе 9, "Транзакции и параллельные вычисления". С блокировками связаны также функции GET_LOCK() и RELEASE_LOCK(), описанные в главе 12, "Встроенные функции".

LOAD DATA INFILE

Инструкция LOAD DATA INFILE читает данные из текстового файла и вставляет их в указанную таблицу. Ниже показан общий формат этой инструкции:

```
LOAD DATA [LOW_PRIORITY] [LOCAL] INFILE путь [IGNORE | REPLACE]
    INTO TABLE таблица
    [FIELDS [TERMINATED BY разделитель]
        [[OPTIONALLY] ENCLOSED BY ограждающий_символ]
        [ESCAPED BY управляющий_символ]]
    [LINES TERMINATED BY признак_конца_строки]
    [IGNORE число_строк LINES]
    [(столбец, ...)]
```

Флаг LOW_PRIORITY говорит о том, что процесс импорта данных должен быть отложен, пока не завершатся все операции чтения таблицы. Если отсутствует флаг LOCAL, файл открывается на сервере, в противном случае — в клиентской файловой системе. Для открытия файла на сервере следует иметь привилегию FILE.

Обычно необходимо указывать полное путевое имя файла. Неполные путевые имена вычисляются относительно каталога данных MySQL. Как правило, это каталог /usr/local/var. Файл, не имеющий путевого имени, открывается в каталоге базы данных, которая задана по умолчанию, например /usr/local/var/test.

В случае импорта записей, нарушающих целостность первичного ключа или ограничение уникальности, будут выдаваться сообщения об ошибке, подавляемые при наличии флага IGNORE или REPLACE. Флаг IGNORE приводит к отбрасыванию неправильных записей, а флаг REPLACE означает замену существующих записей. Флаг LOCAL подразумевает наличие флага IGNORE, поскольку сервер не может запретить клиенту посылать данные.

В листинге 13.33 демонстрируется загрузка содержимого файла /tmp/teams.txt в таблицу team.

```
LOAD DATA INFILE '/tmp/teams.txt'
     INTO TABLE team
     FIELDS TERMINATED BY '\t'
         OPTIONALLY ENCLOSED BY '"'
         ESCAPED BY '\\'
     LINES TERMINATED BY '\r\n'
     IGNORE 1 LINES
      (Name, ID)
```

По умолчанию программа MySQL ожидает, что поля файла разделяются символами табуляции, а специальные символы защищаются от интерпретации с помощью обратной косой черты. Предложение FIELDS позволяет изменять эти установки.

В спецификации TERMINATED BY указывается разделитель полей. Спецификация ENCLOSED BY задает символ (обычно кавычку), который используется для дополнительного обособления значений. Флаг OPTIONALLY не играет никакой роли.

Спецификация ESCAPED BY задает символ, который будет отменять специальное назначение управляющих символов. Ограждающие символы разрешается просто удваивать. Например, если все поля файла заключаются в двойные кавычки, то для вставки кавычки в поле можно записать "" либо \", что одно и то же.

Каждое поле входного файла считается строкой. Строки легко преобразуются в числа, но если тип столбца задан как ENUM или SET, то такое преобразование не будет правильным. Следовательно, элементы перечислений и множеств должны иметь строковый тип.

По умолчанию строки заканчиваются символом новой строки, выбор которого диктуется операционной системой. В UNIX это символ перевода строки. С помощью спецификации LINES TERMINATED BY можно задать другой конечный символ. Если в спецификации LINES TERMINATED BY указана пустая строка, а в спецификации FIELDS TERMINATED BY — нет, конец поля будет означать конец строки.

Если требуется импортировать файл с полями фиксированного размера, нужно задать разделители полей и строк пустыми. В этом режиме предполагается, что размеры полей соответствуют спецификациям столбцов, а сами поля никак не отделяются друг от друга. Например, для столбца типа CHAR(8) будут запрошены ближайшие 8 символов файла. Этот режим не подходит для столбцов типа TEXT и BLOB, а также для файлов, содержащих многобайтовые символы.

С помощью предложения IGNORE...LINES можно пропустить требуемое число строк файла. Это позволяет импортировать файлы, содержащие заголовки данных.

Если данные загружаются лишь в некоторые столбцы, то в конце инструкции должен быть задан список столбцов. Он необходим также в том случае, когда порядок полей входного файла не соответствует порядку столбцов таблицы. Пропущенным столбцам будут присвоены стандартные значения.

По умолчанию значения NULL записываются как \N. Если используются ограничивающие кавычки, то значения NULL вводятся в явном виде, причем без кавычек (с кавычками это будет воспринято как строка 'NULL'). То же самое справедливо и для случая, когда управляющий символ не задан.

Инструкция SELECT позволяет записывать табличные данные в файл. Для этого в ней предусмотрено предложение INTO.

LOAD TABLE

Инструкция LOAD TABLE копирует таблицу с главного сервера на подчиненный сервер:

```
LOAD TABLE имя FROM MASTER
```

OPTIMIZE TABLE

Инструкция OPTIMIZE TABLE исправляет содержимое табличного файла. Ее формат таков:

```
OPTIMIZE TABLE таблица[, таблица ...]
```

По мере вставки и удаления записей таблицы становятся все более **фрагментированными.** MySQL хранит удаленные записи на случай повторного использования, поэтому иногда новые записи разбиваются на два и более фрагмента. Инструкция OPTIMIZE TABLE уничтожает неиспользуемые записи и соединяет разделенные строки. Она также выполняет сортировку индексов.

Обычно лишь таблицы типа **MyISAM** и BDB поддерживают оптимизацию. Можно сконфигурировать программу MySQL на поддержку оптимизации таблиц других типов, но такая оптимизация реализуется путем перестройки таблицы, как если бы была выполнена инструкция ALTER TABLE. Более того, в настоящее время для таблиц типа BDB инструкция OPTIMIZE TABLE эквивалентна инструкции ANALYZE TABLE.

PURGE MASTER LOGS

Инструкция PURGE MASTER LOGS предназначена для удаления старых журнальных файлов на главном сервере. Ее синтаксис таков:

```
PURGE MASTER LOGS TO 'имя'
```

Инструкция удаляет заданный журнальный файл и все предыдущие файлы в последовательном порядке. Эту инструкцию можно вводить даже тогда, когда подчиненный сервер читает журнальные файлы в процессе репликации. Если инструкция попытается удалить используемый файл, ее выполнение аварийно завершится. С помощью инструкции SHOW STATUS можно узнать, какие файлы читаются подчиненными серверами в данный момент. Инструкция SHOW LOGS выдает список журнальных файлов главного сервера.

RENAME TABLE

Инструкция RENAME TABLE **меняет имя** одной **или** нескольких **таблиц:**

```
RENAME TABLE таблица TO имя[, таблица TO имя]...
```

Она выполняется в атомарном режиме. Ошибки, возникающие в процессе прохождения списка, приводят к отмене предыдущих изменений. Имени таблицы может предшествовать имя базы данных, что позволяет данной инструкции перемещать таблицы между базами данных (листинг 13.34). Правда, для этого базы данных должны располагаться на одном физическом диске.

```
RENAME TABLE
    store1.item TO store2.item_temp,
    store2.item TO store1.item,
    store2.item_temp TO store2.item
```

REPAIR TABLE

Инструкция REPAIR TABLE восстанавливает поврежденные таблицы. Ее формат таков:

```
REPAIR TABLE таблица[, таблица ...] [QUICK] [EXTENDED]
```

Если указан флаг QUICK, будут восстанавливаться только индексы. При наличии флага EXTENDED индекс будет создаваться по одной записи за раз, что в некоторых случаях приводит к созданию более качественного индекса.

REPLACE

Инструкция REPLACE аналогична инструкции INSERT, за исключением того, что дублирующиеся записи заменяют собой существующие. Общий формат инструкции таков:

```
REPLACE [LOW_PRIORITY | DELAYED]
 [INTO] таблица [(столбец, ...)]
{VALUES (значение,...),(...),... | запрос \
        SET столбец=значение,...}
```

Инструкция INSERT аварийно завершается, если вставляемая строка конфликтует с первичным ключом или ограничением уникальности. Инструкция REPLACE в подобном случае перезаписывает существующую строку. В остальном обе инструкции эквивалентны.

Если строка вставляется без перезаписи, программа MySQL сообщает о том, что запрос затронул нуль строк. В случае перезаписи сначала удаляется старая строка, а потом вставляется новая, поэтому считается, что были затронуты две строки.

RESET MASTER

Инструкция RESET MASTER удаляет все двоичные журнальные файлы на главном сервере и начинает процесс регистрации событий заново, создавая файл с номером 0. Одновременно с этим нужно повторно проинициализировать подчиненные серверы. Данная инструкция не принимает никаких аргументов.

RESET SLAVE

Инструкция RESET SLAVE заставляет подчиненный сервер начать процесс репликации заново. Она также не требует аргументов.

RESTORE TABLE

Инструкция RESTORE TABLE **восстанавливает таблицу из резервной** копии, которая **была** создана **с** помощью **инструкции** BACKUP TABLE. **Синтаксис инструкции** таков:

```
RESTORE TABLE таблица[, таблица ...] FROM путь
```

Файлы с расширением .frm **и** .MYD копируются **в** соответствующий **каталог базы данных. Если таблица уже существует, выдается** сообщение об ошибке. **Из-за** необходимости перестройки индексов выполнение **инструкции** может **занять** некоторое **время.**

REVOKE

Инструкция REVOKE отменяет **привилегии, выданные ранее инструкцией** GRANT. **Ее** формат таков:

```
REVOKE тип [(столбец, ...)][, тип [(столбец, ...)] ...]
    ON таблица
  FROM пользователь[, пользователь ...]
```

Типы привилегий были перечислены в табл. 13.3. Привилегии выдаются в определенном контексте: глобально, **для базы данных, таблицы или** столбца. **Инструкция** REVOKE должна соответствовать этому контексту. **Привилегии с** более **узким или** более широким контекстом **не затрагиваются.**

ROLLBACK

Инструкция ROLLBACK отменяет **все изменения, сделанные в** ходе **текущей транзакции. Об** этом рассказывалось **в главе 9, "Транзакции и параллельные вычисления". Транзакции** поддерживаются **не для всех** типов **таблиц.**

SELECT

Инструкция SELECT **предназначена для извлечения** информации **из таблиц. Она имеет следующий синтаксис:**

```
SELECT
    [DISTINCT | DISTINCTROW | ALL]
    [HIGH_PRIORITY]
    [SQL_BIG_RESULT I SQL_SMALL_RESULT]
    [SQL_BUFFER_RESULT]
    [STRAIGHT_JOIN]
    выражение[, выражение ...]
    [INTO {OUTFILE I DUMPFILE} файл опции]
    [FROM таблица[, таблица]
        [WHERE условие]
        [GROUP BY столбец[, столбец]]
        [HAVING условие]
        [ORDER BY столбец [ASC | DESC][, столбец [ASC | DESC]]]
        [LIMIT [смещение,] число_строк]
        [PROCEDURE процедура] ]
```

Инструкция вычисляет выражения, указанные в списке возвращаемых столбцов. В выражения могут входить литералы, переменные, функции и столбцы. Последние относятся к таблицам, перечисленным в предложении FROM. Результатом запроса будет таблица извлеченных записей, содержащая в качестве заголовка названия столбцов.

На столбцы ссылаются по имени. Если возникает неоднозначность, необходимо также указывать имя таблицы и базы данных. Полное имя столбца состоит из имени базы данных, имени таблицы и короткого имени столбца, разделенных точками. Таблицам могут быть назначены псевдонимы в предложении FROM, чтобы полные имена столбцов получались короче.

Каждому выражению в списке возвращаемых столбцов тоже может быть присвоен псевдоним с помощью предложения AS. Этот псевдоним появится в заголовке возвращаемой таблицы. Благодаря псевдониму на выражение можно ссылаться в других предложениях инструкции SELECT, например GROUP BY. Если псевдоним представляет собой несколько слов с пробелами, требуются кавычки (листинг 13.35).

```
SELECT NOW() AS 'Current Time'
```

Псевдоним отдельного столбца может упоминаться в любой части инструкции SELECT. Псевдонимы выражений запрещены в предложении WHERE.

Флаг DISTINCT указывает на необходимость удаления дублирующихся записей из таблицы результатов. В данном случае дубликатами считаются такие записи, которые имеют одинаковые значения во всех столбцах таблицы результатов. Речь не идет о проверке первичного ключа или ограничения уникальности. Например, если запросить значения поля "Возраст"таблицы сотрудников, то наверняка некоторые значения будут одинаковыми, хотя в исходной таблице все записи различаются по идентификатору.

DISTINCTROW — это синоним флага DISTINCT. Флаг ALL разрешает появление дубликатов и установлен по умолчанию.

Флаг HIGH_PRIORITY означает принудительное выполнение запроса, даже если имеется отложенная инструкция UPDATE. Обычно операции обновления выполняются в первую очередь.

Флаг SQL_BIG_RESULT заставляет программу MySQL созда вать в случае необходимости временные таблицы на диске. Это требуется для запросов к большому числу таблиц или запросов, возвращающих большую таблицу результатов, особенно если ее нужно сортировать при наличии флага DISTINCT или предложения ORDER BY. Флаг SQL_SMALL_RESULT говорит о том, что временные таблицы должны создаваться в памяти. В новых версиях MySQL этот флаг не нужен.

Флаг SQL_BUFFER_RESULT заставляет предварительно помещать результаты во временную таблицу, а не сразу же посылать их клиенту. Это позволяет быстрее освобождать таблицы, участвующие в запросе.

Флаг STRAIGHT_JOIN говорит о том, что таблицы должны включаться в объединение в том порядке, в котором они перечислены в предложении FROM.

В предложении INTO задается файл, куда должна быть записана таблица результатов. Полный формат этого предложения таков:

```
INTO {OUTFILE | DUMPFILE} файл
    [FIELDS [TERMINATED BY разделитель]
        [[OPTIONALLY] ENCLOSED BY ограждающий_символ]
        [ESCAPED BY управляющий_символ]]
    [LINES TERMINATED BY признак_конца_строки]
```

Поскольку **запись в файл** осуществляется **на сервере,** необходимо **иметь привилегию** FILE. Более того, **запрещается перезаписывать существующий файл. Файл,** созданный **в режиме** OUTFILE, можно **будет загрузить с** помощью **инструкции** LOAD DATA INFILE **(см. выше). Режим** DUMPFILE **предназначен для записи** столбцов **типа** BLOB или TEXT. **В** этом **режиме** запрос должен возвращать одну **запись с одним-единственным** полем. **Управляющие** символы **в** данном **случае не** используются.

В предложении FROM указываются таблицы, из которых извлекаются данные. Здесь может стоять либо список таблиц, разделенных запятыми, либо одна из показанных ниже спецификаций объединения:

```
таблица [CROSS] JOIN таблица
таблица STRAIGHT_JOIN таблица
таблица INNER JOIN таблица
    [{ON условие | USING (столбец, ...)}]
таблица {LEFT | RIGHT} [OUTER] JOIN таблица
    [{ON условие I USING (столбец, ...)}]
таблица NATURAL [{LEFT | RIGHT} [OUTER]] JOIN таблица
```

В операции объединения могут участвовать таблицы стандартной базы данных, таблицы заданной *базы* данных, а также виртуальная таблица, созданная в левой части объединения. В последнем случае создается цепочка объединений. Вспомните из главы 5, "Реляционная модель", что результатом объединения двух таблиц будет новая таблица. В **принципе,** она существует лишь до тех пор, пока выполняется **инструкция** SELECT, но вданном контексте на нее можно ссылаться как на настоящую таблицу, т.е. справа от нее может стоять оператор объединения.

Синтаксис табличной ссылки таков:

```
[база_данных.]таблица AS псевдоним
    [USE INDEX (индекс, ...)] [IGNORE INDEX (индекс, ...)]
```

Перед именем таблицы может стоять имя базы данных. Предложение AS задает псевдоним таблицы. Этот псевдоним можно использовать во всех остальных предложениях инструкции, в частности в операциях объединения, в условиях отбора, в списке возвращаемых столбцов и т.д. Предложение USE INDEX заставляет программу MySQLиспользовать указанные индексы, если это возможно, а предложение IGNORE INDEX, наоборот, запрещает некоторые индексы. Эти подсказки позволяют **повысить** производительность запроса.

Операция CROSS JOIN создает произведение таблиц. В этом случае каждая строка левой таблицы по очереди объединяется с каждой строкой правой таблицы и **включается** в таблицу произведения. Операция STRAIGHT_JOINприводит **к** тем же результатам.

В листинге 13.36 выполняется ***самообъединение*** таблицы, т.е. создается произведение двух копий одной и той же таблицы. Если в исходной таблице 4 строки, то в **таблице** произведения их будет 16.

```
SELECT t1.Name, t2.Name
  FROM team t1 JOIN team t2
```

Операция INNER JOIN включает в таблицу результатов только те строки таблицы произведения, которые удовлетворяют определенному критерию. Это называется *внутренним объединением*. Предложение USING задает столбцы, включаемые в объединение. Эти столбцы должны называться одинаково в обеих таблицах. В листинге 13.37 таблица игроков (player) объединяется с таблицей команд (team) по столбцу идентификатора команды (TeamID).

```
SELECT player.Name, team.Name
  FROM player INNER JOIN team USING (TeamID)
```

Предложение ON заменяет предложение WHERE и может содержать любые критерии сравнения строк двух объединяемых таблиц. Чаще всего таблицы объединяются по равенству двух столбцов, как показано в листинге 13.38. Обратите внимание на различные имена столбцов.

```
SELECT player.Name, team.Name
  FROM player INNER JOIN team
    ON (player.Team = team.ID)
```

В левое внешнее объединение (LEFT JOIN) включаются все записи внутреннего объединения плюс строки первой таблицы, которые не имеют связи ни с одной строкой второй таблицы. Правое внешнее объединение (RIGHT JOIN) строится так же, но относительно правой, а не левой таблицы. Строки, которым не найдено соответствие, дополняются значениями NULL. Ключевое слово OUTER является необязательным. Оно предназначено для сохранения совместимости с другими СУБД.

Естественное объединение (NATURAL JOIN) характерно тем, что сравниваются только одноименные столбцы двух таблиц, причем выполняется их проверка на равенство. Обычно это действительно самый "естественный" тип объединения, так как не нужно указывать ни имена столбцов, ни условие отбора. По умолчанию создается внутреннее естественное объединение, но с помощью ключевых слов LEFT и RIGHT его можно сделать левым или правым внешним объединением.

Некоторые предложения инструкции SELECT доступны лишь тогда, когда присутствует предложение FROM. Они выполняются в строго определенном порядке: сначала – WHERE, затем – GROUP BY, потом – HAVING и, наконец – ORDER BY.

Предложение WHERE задает условия, в соответствии с которыми из таблицы произведения отбираются строки. Если в предложении FROM была указана одна таблица, то условие отбора является критерием фильтрации ее строк. Если же выполняется операция объединения, то в предложении WHERE указываются правила сопоставления столбцов объединяемых таблиц. В листинге 13.39 создается то же самое объединение, что и в листинге 13.38, но на этот раз с использованием предложения WHERE.

```
SELECT player.Name, team.Name
  FROM player, team
 WHERE player.Team = team.ID
```

Аргументом предложения WHERE является булево выражение. В него может входить произвольное число выражений сравнения, объединяемых с помощью логических операторов. В листинге 13.40 показана более сложная форма предложения WHERE.

```
SELECT p.Name, t.Name
  FROM player p, team t
 WHERE p.Team = t.ID
   AND (p.Position = 'Catcher'
    OR  p.Position = 'Pitcher')
```

Предложение GROUP BY содержит правило группировки строк по одному или нескольким столбцам. Это позволяет применять к столбцам статистические функции (см. главу 12, "Встроенные функции"). Статистические функции разрешены в списке возвращаемых столбцов и в предложении HAVING, но не в предложении WHERE, поскольку оно вычисляется раньше, чем GROUP BY, т.е. до того, как формируются группы записей.

Запрос, показанный в листинге 13.41, группирует записи по идентификатору команды и возвращает число питчеров в каждой команде при условии, что таковых более девяти.

```
SELECT t.Name AS 'Team', COUNT(*) AS 'Pitchers'
  FROM player p, team t
 WHERE p.Team = t.ID
   AND p.Position = 'Pitcher'
 GROUP BY t.ID
HAVING Pitchers > 9
 ORDER BY Pitchers DESC
```

В предложении GROUP BY можно ссылаться на столбцы по имени или по номеру. Столбцы нумеруются слева направо, а нумерация начинается с единицы. Разрешается также группировать записи по столбцам, не указанным в предложении SELECT. Такие столбцы, естественно, не имеют номера.

В предложении HAVING указывается такое же булево выражение, как и в предложении WHERE. Оно задает условия отбора групп. Поскольку предложение HAVING проверяется после предложений WHERE и GROUP BY, в нем можно использовать псевдонимы столбцов и статистические функции. Разрешается ссылаться только на те столбцы, которые указаны в списке возвращаемых столбцов.

Предложение ORDER BY задает правило сортировки строк возвращаемой таблицы. Разрешается сортировать результаты запроса по любому элементу списка возвращаемых столбцов. Как и в случае предложения GROUP BY, на столбцы можно ссылаться по номерам.

По умолчанию столбцы сортируются по возрастанию, т.е. флаг ASC указывать не обязательно. Для сортировки по убыванию предназначен флаг DESC.

Предложение LIMIT позволяет отобрать из возвращаемой таблицы требуемое подмножество строк. Если в качестве аргумента указано одно число, возвращаются строки с первой по заданную. Если присутствуют два числа, то первое определяет начальный номер строки, а второе — число возвращаемых строк.

Предложение PROCEDURE предназначено для вызова процедуры, которая выполняется над таблицей результатов запроса перед тем, как эта таблица посылается клиенту. В MySQL есть лишь одна такая процедура: analyse() (см. главу 12, "Встроенные функции").

Для совместимости с Oracle разрешается ссылаться на виртуальный столбец _rowid (псевдоним первичного ключа). Еще один виртуальный столбец — auto_increment_column — позволяет находить последнюю вставленную строку таблицы, содержащей столбец-счетчик. Соответствующая проверка выглядит так:

```
WHERE auto_increment_column IS NULL
```

SET

Инструкция SET меняет значения опций и переменных. Ее синтаксис таков:

```
SET [OPTION] параметр=значение[, параметр=значение]...
```

Устанавливаемые значения действительны лишь в течение текущего сеанса и по его завершении сбрасываются. Ниже описаны возможные варианты спецификаций.

@переменная = значение

Эта спецификация предназначена для задания значений переменных. О переменных рассказывалось в главе 10, "Типы данных, переменные и выражения".

AUTOCOMMIT = {0 | 1}

Опция AUTOCOMMIT определяет, должна ли программа MySQL немедленно фиксировать изменения, вносимые запросами, или необходимо дожидаться завершения транзакции. О транзакциях рассказывалось в главе 9, "Транзакции и параллельные вычисления". По умолчанию данная опция включена.

CHARACTER SET {имя | DEFAULT}

Эта опция задает используемый набор символов (листинг 13.42). Ключевое слово DEFAULT восстанавливает стандартный набор. В MySQL входит лишь один набор

символов — cp1251_koi8, но можно создавать и собственные наборы, как описано в главе 31, "Расширение возможностей MySQL".

```
SET CHARACTER SET cp1251_koi8
```

Обратите внимание на то, что, в отличие от других опций, здесь не используется знак равенства.

INSERT_ID = число

Эта опция задает следующее значение поля-счетчика.

LAST_INSERT_ID = число

Эта опция задает значение, возвращаемое функцией LAST_INSERT_ID() (см. главу 12, "Встроенные функции").

PASSWORD = PASSWORD(пароль)

Эта опция позволяет текущему пользователю задать свой пароль. Зашифрованный пароль хранится в файле mysql.user.

PASSWORD FOR пользователь = PASSWORD(пароль)

С помощью этой опции можно задать пароль произвольного пользователя. Для этого необходимо иметь право записи в базу данных mysql. Пользователи указываются так же, как и в инструкции GRANT.

SQL_AUTO_IS_NULL = {0 | 1}

Эта опция задает использование виртуального столбца auto_increment_column. Если таблица содержит поле-счетчик, то последняя добавленная в нее строка будет содержать значение NULL в данном столбце (листинг 13.43).

```
SELECT *
  FROM team
 WHERE auto_increment_column IS NULL
```

Эта опция включена по умолчанию.

SQL_BIG_SELECTS = {0 | 1}

Если эта опция отключена, то инструкции SELECT со слишком большим числом объединений не будут выполняться. Предельное число объединений хранится в серверной переменной max_join_size.

SQL_BIG_TABLES = {0 | 1}

По умолчанию временные таблицы хранятся в памяти. Но если данная опция включена, то все временные таблицы будут принудительно выгружены на диск (листинг 13.44).

```
SET SQL_BIG_TABLES=1
```

SQL_BUFFER_RESULT = {0 | 1}

Как и в случае одноименного флага инструкции SELECT, эта опция заставляет записывать результаты запросов во временные таблицы с целью ускорения доступа к ним.

SQL_LOG_BIN = {0 | 1}

Эта опция включает или отключает журнальную регистрацию на главном сервере.

SQL_LOG_OFF = {0 | 1}

Если задать эту опцию равной 1, программа перестанет посылать данные в стандартный журнальный файл. Чтобы выполнить подобное изменение, пользователь должен иметь привилегию PROCESS.

SQL_LOG_UPDATE = {0 | 1}

Если задать эту опцию равной 0, программа перестанет посылать данные в файл регистрации обновлений. Чтобы выполнить подобное изменение, пользователь должен иметь привилегию PROCESS.

SQL_LOW_PRIORITY_UPDATES = {0 | 1}

Эта опция отключена поумолчанию. Если ее включить, все запросы на изменение значений столбцов начнут получать более низкий приоритет, чем инструкции SELECT. В этом случае инструкции DELETE, INSERT, LOCK TABLES WRITE и UPDATE будут блокироваться до завершения всех операций чтения.

SQL_MAX_JOIN_SIZE = {размер | DEFAULT}

Эта опция позволяет задать максимальное число объединений, допустимых в запросе. Если значение опции будет отличаться от того, что установлено по умолчанию, описанная выше опция SQL_BIG_SELECTS будет отключена.

SQL_QUOTE_SHOW_CREATE = {0 | 1}

По умолчанию инструкция SHOW CREATE TABLE помещает кавычки вокруг имен столбцов и таблиц. Если задать данную опцию равной нулю, кавычки выводиться не будут.

SQL_SAFE_MODE = {0 | 1}

Когда эта опция включена, все запросы на обновление и удаление, в которых не указан первичный ключ, отменяются. Подобный режим предназначен для новичков, которые способны ненароком ввести запрос, затрагивающий все записи таблицы.

SQL_SELECT_LIMIT = {предел | DEFAULT}

Эта опция ограничивает число записей, возвращаемых запросами на выборку. Предложение LIMIT инструкции SELECT переопределяет данную установку. По умолчанию может возвращаться неограниченное число записей.

SQL_SLAVE_SKIP_COUNTER = число_позиций

Эта опция заставляет подчиненный сервер пропустить указанное число позиций в журнальном файле главного сервера. Перед тем как устанавливать данное значение, необходимо отключить репликацию.

TIMESTAMP = {метка_времени | DEFAULT}

Эта опция задает текущее время сеанса, что бывает полезно при восстановлении измененных записей.

SET TRANSACTION

Инструкция SET TRANSACTION задает уровень изоляции транзакции. Ее формат таков:

```
SET [GLOBAL | SESSION] TRANSACTION LEVEL
    [READ UNCOMMITTED |READ COMMITTED | REPEATABLE READ |
     SERIALIZABLE]
```

По умолчанию инструкция затрагивает только ближайшую транзакцию. Если указан флаг GLOBAL, изменения коснутся всех транзакций, выполняемых на сервере. Флаг SESSION обозначает все транзакции данного сеанса.

Уровень изоляции определяет, к каким данным, с которыми работает транзакция, смогут получить доступ другие потоки. В режиме READ UNCOMMITTED транзакция может читать данные из таблиц напрямую, не проверяя, окончательные это данные или нет. Другим потокам разрешено менять содержимое этих таблиц, не дожидаясь завершения транзакции. Таким образом, в ходе транзакции можно выполнить один и тот же запрос дважды подряд и получить разные результаты.

В режиме READ COMMITTED транзакция работает только с окончательными данными. Изменения, вносимые другими потоками, не отразятся в запросе на выборку до тех пор, пока не будут зафиксированы. Тем не менее по-прежнему существует возможность того, что один и тот же запрос, выполненный дважды в ходе одной транзакции, выдаст разные результаты, так как между первым и вторым запросом другой поток может успеть выполнить собственную транзакцию.

В режиме REPEATABLE READ любая строка, читаемая или обновляемая в ходе транзакции, не может быть изменена другим потоком. Транзакция блокирует все строки, к которым обращается. Таким образом, значения, извлекаемые инструкцией

SELECT, никогда не меняются в течение транзакции. Но возможно появление строк-призраков, т.е. во время транзакции можно увидеть строку, добавленную другой транзакцией.

В режиме SERIALIZABLE транзакции принудительно выполняются друг за другом, последовательно. Несколько потоков могут начинать транзакции одновременно, но если окажется, что две транзакции пытаются обновить одну и ту же строку, одна из них будет объявлена проигравшей в тупиковой ситуации и отменена. Это наивысший уровень целостности данных.

SHOW COLUMNS

Инструкция SHOW COLUMNS возвращает описание столбцов таблицы. Она имеет следующий синтаксис:

```
SHOW [FULL] {COLUMNS I FIELDS}
FROM таблица [FROM база_данных] [LIKE шаблон]
```

Флаг FULL задает выдачу для каждого столбца информации о привилегиях. Предложение LIKE позволяет отобрать из таблицы только те столбцы, имена которых соответствуют указанному шаблону. Синтаксис строки шаблона был описан в главе 10, "Типы данных, переменные и выражения". Аналогичные результаты выдает инструкция DESCRIBE.

```
mysql> SHOW COLUMNS FROM team;
+--------+---------------------+------+-----+---------+----------------+
| Field  | Type                | Null | Key | Default | Extra          |
+--------+---------------------+------+-----+---------+----------------+
| ID     | tinyint(3) unsigned |      | PRI | NULL    | auto_increment |
| Name   | varchar(16)         |      |     |         |                |
+--------+---------------------+------+-----+---------+----------------+
2 rows in set (0.00 sec)
```

SHOW CREATE TABLE

Инструкция SHOW CREATE TABLE возвращает описание запроса, который необходим для создания указанной таблицы. Синтаксис инструкции таков:

```
SHOW CREATE TABLE таблица
```

В листинге 13.46 показаны результаты, выдаваемые в случае таблицы mysql.db. Обратите внимание на то, что имена столбцов заключены в обратные кавычки. Если установить опцию SQL_QUOTE_SHOW_CREATE, кавычки не будут отображаться, но они необходимы, когда есть имена столбцов с пробелами.

```
mysql> SHOW CREATE TABLE mysql.db \G
*************************** 1. row ***************************
       Table: db
Create Table: CREATE TABLE `db` (
  `Host` char(60) binary NOT NULL default '',
  `Db` char(64) binary NOT NULL default '',
  `User` char(16) binary NOT NULL default '',
  `Select_priv` enum('N','Y') NOT NULL default 'N',
  `Insert_priv` enum('N','Y') NOT NULL default 'N',
  `Update_priv` enum('N','Y') NOT NULL default 'N',
  `Delete_priv` enum('N','Y') NOT NULL default 'N',
  `Create_priv` enum('N','Y') NOT NULL default 'N',
  `Drop_priv` enum('N','Y') NOT NULL default 'N',
  `Grant_priv` enum('N','Y') NOT NULL default 'N',
  `References_priv` enum('N','Y') NOT NULL default 'N',
  `Index_priv` enum('N','Y') NOT NULL default 'N',
  `Alter_priv` enum('N','Y') NOT NULL default 'N',
  PRIMARY KEY (`Host`,`Db`,`User`),
  KEY `User` (`User`)
) TYPE=MYISAM COMMENT='Database privileges'
1 row in set (0.01 sec)
```

SHOW DATABASES

Инструкция SHOW DATABASES возвращает список баз данных, существующих на сервере:

SHOW DATABASES [LIKE *шаблон]*

Пример инструкции показан в листинге 13.47. В предложении LIKE задается **шаблон** имен баз данных (см. главу 10, "Типы данных, переменные и выражения").

```
mysql> SHOW DATABASES LIKE '%mysql';
+--------------------+
| Database (%mysql)  |
+--------------------+
| coremysql          |
| mysql              |
+--------------------+
2 rows in set (0.01 sec)
```

SHOW GRANTS

Инструкция SHOW GRANTS выводит список привилегий, предоставленных заданному пользователю:

SHOW GRANTS FOR *пользователь*

Имя пользователя можно дополнить именем узла, как показано в листинге 13.48.

```
mysql> SHOW GRANTS FOR root@localhost;
+-----------------------------------------------------------------------+
| Grants for root@localhost                                             |
+-----------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

SHOW INDEX

Инструкция SHOW INDEX возвращает информацию об индексах таблицы. Ее синтаксис таков:

```
SHOW {INDEX | KEYS}
FROM таблица [FROM база_данных]
```

Результатом этого запроса будет таблица, столбцы которой перечислены в табл. 13.4. Каждый компонент составного индекса будет представлен в таблице результатов отдельной строкой.

Столбец	*Описание*
Table	Имя таблицы
Non_unique	Указание на то, допускаются ли дублирующиеся значения
Key_name	Имя индекса
Seq_in_index	Порядковый номер для компонентов составного индекса
Column_name	Индексируемый столбец
Collation	Порядок сортировки
Cardinality	Среднестатистическое количество уникальных элементов индекса
Sub_part	Число индексируемых символов в столбце с частичной индексацией
Packed	Признак того, является ли индекс сжатым
Comment	Указание на то, имеет ли индекс тип FULLTEXT

SHOW LOGS

Инструкция SHOW LOGS возвращает информацию о журнальных файлах:

```
SHOW [MASTER] LOGS
```

Таблица результатов содержит три столбца. В столбце File отображается путь к файлу, а в столбце Type — тип файла, например BDB. Значение столбца Status определяет статус файла: FREE или IN USE.

SHOW PROCESSLIST

Инструкция SHOW PROCESSLIST выдает список соединений с текущей базой данных:

```
SHOW [FULL] PROCESSLIST
```

Если текущий пользователь имеет привилегию PROCESS, будет показан список всех соединений, поддерживаемых на сервере. В противном случае учитываются лишь соединения текущего пользователя. Флаг FULL говорит о том, что для каждого соединения необходимо показать полный текст запроса, выполняемого в настоящий момент. По умолчанию отображаются только 100 первых символов запроса.

SHOWSTATUS

Инструкция SHOW STATUS возвращает статистическую информацию о сервере. Ее синтаксис таков:

```
SHOW [MASTER I SLAVE] STATUS [LIKE шаблон]
```

Флаги MASTER и SLAVE указывают на то, что информация выдается соответственно о главном и подчиненном серверах. В предложении LIKE задается шаблон имен выводимых переменных. Список переменных приведен в табл. 13.5.

Переменная	*Описание*
Aborted_clients	Количество неправильно закрытых соединений
Aborted_connects	Количество неудавшихся попыток подключиться к серверу
Bytes_received	Количество байтов, полученных от клиентов
Bytes_sent	Количество байтов, посланных клиентам
Connections	Количество попыток подключиться к серверу
Created_tmp_disk_tables	Число временных таблиц, созданных на диске
Created_tmp_files	Число временных файлов, созданных демоном mysqld

Переменная	*Описание*
Created_tmp_tables	Число временных таблиц, созданных в памяти
Delayed_errors	Количество инструкций INSERT DELAYED, вызвавшихкакую-тоошибку
Delayed_insert_threads	Число потоков, в которых выполняются инструкции INSERT DELAYED
Delayed_writes	Количество строк, записанных инструкциями INSERT DELAYED
Flush_commands	Число инструкций FLUSH
Handler_delete	Количество удаленных строк
Handler_read_first	Количество операций чтения первого элемента любого индекса
Handler_read_key	Количество операций чтения индексируемых строк
Handler_read_next	Количество операций чтения следующего элемента индекса
Handler_read_prev	Количество операций чтения предыдущего элемента индекса
Handler_read_rnd	Количество операций чтения фиксированного элемента индекса
Handler_read_rnd_next	Количество операций чтения следующей записи из файла данных
Handler_update	Число запросов на обновление
Handler_write	Число запросов на добавление
Key_blocks_used	Количество используемых блоков в индексном буфере
Key_reads	Количество операций чтения индексных блоков с диска
Key_read_requests	Количество запросов на чтение индексных блоков из буфера
Key_writes	Количество операций записи индексных блоков надиск
Key_write_requests	Количество запросов на запись индексных блоков в буфер

Переменная	*Описание*
Max_used_connections	Максимальное число одновременных соединений
Not_flushed_delayed_rows	Количество строк, которые должны быть записаны инструкциями INSERT DELAYED
Not_flushed_key_blocks	Количество измененных индексных блоков, ожидающих записи на диск
Opened_tables	Число открывавшихся таблиц
Open_files	Число открытых файлов
Open_streams	Число открытых потоков
Open_tables	Число открытых таблиц
Questions	Общее число запросов
Select_full_join	Число операций объединения, в которых не использовался индекс
Select_full_range_join	Число операций объединения, в которых приходилось отбирать диапазон строк второй таблицы
Select_range	Число операций объединения, в которых приходилось отбирать диапазоны строк первой таблицы
Select_range_check	Число операций объединения, в которых не использовался индекс, Но после каждой строки пришлось проверять правильность ключа
Select_scan	Число операций объединения, в которых пришлось сканировать первую таблицу
Slave_open_temp_tables	Число временных таблиц, открытых на подчиненном сервере
Slave_running	Статус подчиненного сервера
Slow_launch_threads	Число потоков, которые слишком долго запускались
Slow_queries	Число запросов, которые слишком Долго выполнялись
Sort_merge_passes	Число операций слияния, которые пришлось выполнить при сортировке
Sort_range	Число операций сортировки, в которых пришлось создавать диапазоны строк
Sort_rows	Число отсортированных строк

Переменная	*Описание*
Sort_scan	Число операций сортировки, в которых пришлось сканировать таблицу
Table_locks_immediate	Число табличных блокировок, которые были поставлены немедленно
Table_locks_waited	Число табличных блокировок, которые не были поставлены немедленно
Threads_cached	Число потоков в кэш-буфере
Threads_connected	Текущее число соединений
Threads_created	Количество созданных потоков
Threads_running	Количество выполняющихся потоков
Uptime	Количество секунд, прошедших с момента запуска сервера

SHOW TABLE STATUS

Инструкция SHOW TABLE STATUS выводит статусную информацию о таблицах. Ее синтаксис таков:

SHOW TABLE STATUS [FROM *база_данных*] [LIKE шаблон]

В предложении LIKE задается шаблон имен таблиц. Список столбцов возвращаемой таблицы приведен в табл. 13.6. Описание каждой таблицы базы данных занимает одну строку.

Столбец	*Описание*
Auto_increment	Следующее значение поля-счетчика
Avg_row_length	Средняя длина записи
Check_time	Время последнего выполнения инструкции CHECK TABLE
Comment	Комментарии, указанные при создании таблицы
Create_options	Параметры создания таблицы
Create_time	Время создания таблицы
Data_free	Число выделенных, но неиспользуемых байтов
Data_length	Длина файла данных

Столбец	*Описание*
Index_length	Длина индексного файла
Max_data_length	Максимальная длина файла данных
Name	Имя таблицы
Rows	Число записей в таблице
`Row_format`	Формат хранения записей: сжатый, динамический или фиксированный
Type	Тип таблицы
`Update_time`	Время последнего обновления таблицы

SHOW TABLES

Инструкция SHOW TABLES возвращает список таблиц, существующих в указанной базе данных:

SHOW [OPEN] TABLES [FROM *база_данных*] [LIKE *шаблон]*

По умолчанию отображаются все таблицы текущей базы данных. Флаг OPEN ограничивает список только открытыми таблицами. В этом случае во втором столбце указывается, сколько раз таблица **кэшировалась** и использовалась (листинг 13.49).

```
mysql> SHOW OPEN TABLES FROM mysql;
+----------------------+--------------------+
| Open_tables_in_mysql | Comment            |
+----------------------+--------------------+
| user                 | cached=1, in_use=0 |
| func                 | cached=1, in_use=0 |
| host                 | cached=1, in_use=0 |
| columns_priv         | cached=1, in_use=0 |
| db                   | cached=1, in_use=0 |
| tables_priv          | cached=1, in_use=0 |
+----------------------+--------------------+
6 rows in set (0.00 sec)
```

SHOW VARIABLES

Инструкция SHOW VARIABLES возвращает список серверных переменных:

SHOW VARIABLES [LIKE *шаблон]*

Подробное описание серверных переменных приведено в табл. 13.7.

Переменная	*Описание*
ansi_mode	Будет равна ON, если включен режим ANSI (описан в главе 14, "Утилиты командной строки" при рассмотрении fleMOHamysqld)
back_log	Максимальное число необслуженных запросов на подключение
basedir	Базовый каталог (по умолчанию— /usr/local)
bdb_cache_size	Размер кэша для таблиц типа BDB
bdb_home	Каталог для таблиц типа BDB (по умолчанию — /usr/local/var)
bdb_logdir	Каталог журнальных файлов для таблиц типа BDB
bdb_log_buffer_size	Размер буфера журнальной регистрации для таблиц типа BDB
bdb_max_lock	Максимальное число блокировок для таблиц типа BDB
bdb_shared_data	Будет равна ON, если для таблиц типа BDB включен режим совместного использования данных
bdb_tmpdir	Временный каталог для таблиц типа BDB
bdb_version	Версия СУБД Berkeley DB
binlog_cache_size	Размер буфера для двоичного журнала транзакций
character_set	Набор символов по умолчанию
character_sets	Поддерживаемые наборы символов
concurrent_insert	Будет равна ON, если для таблиц типа MyISAM разрешены одновременные операции вставки
connect_timeout	Время простоя, по истечении которого запрос на подключение будет отменен
datadir	Каталог табличных данных (по умолчанию— /usr/local/var)

Переменная	*Описание*
delayed_insert_limit	Число строк, которые могут быть записаны инструкцией INSERT DELAYED, прежде чем снова будут разрешены операции чтения
delayed_insert_timeout	Время, отводимое на выполнение инструкции INSERT DELAYED
delayed_queue_size	Число строк, поставленных в очередь на отложенную запись
delay_key_write	Будет равна ON, если для инструкций CREATE TABLE включена опция DELAY_KEY_WRITE
flush	Будет равна ON, если таблицы должны регулярно закрываться в целях синхронизации и освобождения ресурсов
flush_time	Интервал времени между операциями закрытия таблиц
have_bdb	Будет равна ON, если поддерживаются таблицы типа BDB
have_gemini	Будет равна ON, если поддерживаются таблицы типа Gemini
have_innodb	Будет равна ON, если поддерживаются таблицы типа InnoDB
have_isam	Будет равна ON, если поддерживаются таблицы типа ISAM
have_raid	Будет равна ON, если поддерживаются таблицы типа RAID
have_ssl	Будет равна ON, если для канала связи между клиентом и сервером поддерживается протокол шифрования SSL
init_file	Имя сценария, выполняемого при запуске сервера
innodb_data_file_path	Имя используемого в данный момент файла InnoDB

Переменная	*Описание*
innodb_data_home_dir	Каталог для файлов InnoDB
innodb_flush_log_at_trx_commit	Будет равна ON, если журнальные файлы InnoDB закрываются при завершении транзакции
innodb_log_archive	Будет равна ON, если журнальные файлы InnoDB архивируются
innodb_log_arch_dir	Каталог для архивов журнальных файлов InnoDB
innodb_log_group_home_dir	Каталог для журнальных файлов InnoDB
interactive_timeout	Допустимое время простоя в интерактивном сеансе
join_buffer_size	Размер буфера для операций объединения, в которых не используются индексы
key_buffer_size	Размер индексного буфера, используемого всеми соединениями
language	Каталог, содержащий тексты сообщений об ошибках
large_files_support	Будет равна ON, если поддерживаются большие файлы
locked_in_memory	Будет равна ON, если демон mysqld запущен с флагом --memlock
log	Будет равна ON, если включена журнальная регистрация всех запросов
log_bin	Будет равна ON, если включена двоичная журнальная регистрация
log_slave_updates	Будет равна ON, если регистрируются обновления подчиненного сервера
log_update	Будет равна ON, если включена регистрация обновлений
long_query_time	Число секунд, по истечении которого выполняющийся запрос считается медленным

Переменная	*Описание*
lower_case_table_names	Будет равна 1, если имена всех таблиц принудительно переводятся в нижний регистр
low_priority_updates	Будет равна ON, если запросы на обновление имеют более низкий приоритет, чем запросы на выборку
max_allowed_packet	Максимальный размер пакета в байтах
max_binlog_cache_size	Максимальный размер двоичного журнала транзакций
max_binlog_size	Предельный размер в байтах, по достижении которого произойдет ротация двоичных журнальных файлов
max_connections	Максимальное число соединений
max_connect_errors	Максимальное число ошибок подключения, по достижении которого доступ к узлу блокируется
max_delayed_threads	Максимальное число потоков, в которых выполняются инструкции INSERT DELAYED
max_heap_table_size	Максимальный размер таблиц типа HEAP
max_join_size	Максимальное число строк, которое может быть получено в результате объединения таблиц
max_sort_length	Максимальное число байтов, используемое при сортировке столбцов типа BLOB и TEXT
max_tmp_tables	Максимальное число временных таблиц
max_user_connections	Максимальное число соединений для одного пользователя
max_write_lock_count	Число блокировок записи, после которого необходимо разрешить блокировки чтения

Переменная	*Описание*
myisam_max_extra_sort_file_size	Максимальная разница в размерах временного файла, используемого для создания индекса, и индексного кэша
myisam_max_sort_file_size	Максимальный размер временного файла, используемого для создания индекса; в случае превышения этого предела будет использован индексный буфер
myisam_recover_options	Параметры восстановления таблиц
myisam_sort_buffer_size	Размер буфера, используемого для создания индексов таблиц типа MyISAM
net_buffer_length	Ожидаемая длина запросов, посылаемых клиентами
net_read_timeout	Предельное время ожидания следующих данных в ходе операции чтения
net_retry_count	Число попыток восстановить прерванную операцию чтения
net_write_timeout	Предельное время ожидания записи требуемого блока данных
open_files_limit	Максимальное число файловых дескрипторов, используемых демоном mysqld
pid_file	Файл, в котором хранится идентификатор процесса mysqld
port	Порт для приема запросов на подключение
protocol_version	Версия коммуникационного протокола, используемого сервером MySQL
query_buffer_size	Начальный размер буфера запросов
record_buffer	Размер буфера, используемого при сканировании таблиц
safe_show_database	Будет равна ON, если пользователям не разрешено просматривать базы данных, для доступа к которым у них нет привилегий

Переменная	*Описание*
server_id	Идентификатор сервера
skip_locking	Будет равна ON, если используются внутренние блокировки
skip_networking	Будет равна ON, если соединения устанавливаются только локально
skip_show_database	Будет равна ON, если для просмотра списка баз данных пользователю необходимо иметь привилегию PROCESS
slow_launch_time	Число секунд, по истечении которого запуск потока будет считаться медленным
socket	UNIX-сокет, используемый для подключения к серверу (обычно /tmp/mysql.sock)
sort_buffer	Размер буфера, используемого для сортировки записей
table_cache	Максимальное число открытых таблиц
table_type	Типтаблиц по умолчанию
thread_cache_size	Число потоков, хранимых в кэше для повторного использования
thread_concurrency	Указание на число одновременных потоков (только в Solaris)
thread_stack	Размер стека потоков
timezone	Часовой пояс, в котором работает сервер
tmpdir	Каталог временных файлов
tmp_table_size	Максимальный размер временной таблицы, по достижении которого она записывается на диск
transaction_isolation	Уровень изоляции потоков
version	Версия сервера
wait_timeout	Период простоя, по истечении которого соединение закрывается

SLAVE

Инструкция SLAVE управляет репликацией подчиненного сервера:

```
SLAVE [START | STOP]
```

TRUNCATE

Инструкция TRUNCATE удаляет все записи из таблицы и имеет следующий синтаксис:

```
TRUNCATE TABLE таблица
```

Функционально она напоминает инструкцию DELETE, но реализована по-другому: она удаляет и воссоздает таблицу. Следовательно, эффект инструкции TRUNCATE необратим. Если она вызывается в ходе транзакции, последняя завершается до выполнения инструкции. Подобный способ очистки таблицы работает быстрее, чем когда используется инструкция DELETE.

UNLOCK TABLES

Инструкция UNLOCK TABLE снимает все установленные блокировки:

```
UNLOCK TABLES
```

Она отменяет действия, произведенные инструкцией LOCK TABLES.

UPDATE

Инструкция UPDATE изменяет значения столбцов таблицы. Ее синтаксис таков:

```
UPDATE [LOW_PRIORITY] [IGNORE] таблица
   SET столбец=значение[, столбец=значение, ...]
[WHERE условие]
[ORDER BY столбец [ASC I DESC][, столбец [ASC | DESC]]]
[LIMIT предел]
```

Флаг LOW_PRIORITY говорит о том, что операции обновления должны **быть** отложены до **тех** пор, пока **не завершатся все** операции **чтения. Флаг** IGNORE **заставляет** программу **MySQL** игнорировать **изменения,** приводящие **к** конфликтам. Спорные строки останутся нетронутыми.

За раз можно обновить только одну таблицу, но произвольное число столбцов. Имена столбцов и присваиваемые им значения приводятся в предложении SET. Предложение WHERE содержит условие отбора обновляемых записей. В листинге 13.50 отбирается одна строка, идентифицируемая по первичному ключу.

```
UPDATE team
   SET Name='Wildcats'
 WHERE ID=5
```

Можно ссылаться на текущее значение обновляемого столбца. Например, в листинге 13.51 к текущему значению столбца Customers товара с идентификатором 3 прибавляется число 2.

```
UPDATE store
   SET Customers = Customers + 2, LastUpdate = NOW()
 WHERE StoreID=3
```

Предложение ORDER BY задает порядок сортировки отбираемых записей. Предложение LIMIT заставляет программу MySQL обновлять лишь указанный диапазон строк, соответствующих условию отбора. Благодаря этим предложениям можно разбить большую операцию обновления на несколько маленьких.

USE

Инструкция USE задает базу данных, используемую по умолчанию:

```
USE база_данных
```

В последующих инструкциях все таблицы, имена которых приведены в коротком формате, будут считаться принадлежащими к этой базе данных. Просмотреть список имеющихся баз данных можно с помощью инструкции SHOW DATABASES.

УТИЛИТЫ КОМАНДНОЙ СТРОКИ

В этой главе.

Переменные среды
Конфигурационные файлы
Полный список утилит

Глава 14

та глава содержит описание утилит **командной строки**, входящих в стандартный дистрибутив MySQL. Большинство утилит представляет собой двоичные Исполняемые файлы. Некоторые являются сценариями интерпретатора команд. В основном они доступны для всех платформ, хотя есть и исключения.

В главе 3, "Взаимодействие с MySQL", уже описывалась основная клиентская утилита `mysql`, но это лишь верхушка айсберга. Имеются полезные утилиты, позволяющие проверять и восстанавливать таблицы, создавать резервные копии базы данных, не прерывая работу сервера, и т.д.

Переменные среды

Все утилиты MySQL основаны на библиотеке клиентских функций, поэтому у них есть ряд общих установок. Эти установки хранятся в переменных среды и дублируются в некоторых аргументах командной строки.

MYSQL_DEBUG

Эта переменная задает параметры отладки. Она эквивалентна опции `--debug` некоторых утилит.

MYSQL_PWD

Эта переменная может хранить стандартный пароль, используемый при регистрации на сервере. Помните о **том**, что пароль не защищен от просмотра другими пользователями. Данная переменная эквивалентна опции `--password`.

MYSQL_TCP_PORT

Эта переменная хранит стандартный номер порта, через который осуществляется подключение к серверу MySQL. Она эквивал ентна опции `--port`.

MYSQL_UNIX_PORT

Эта переменная хранит путевое имя UNIX-сокета, используемого для подключения к серверу MySQL. Она эквивалентна опции `--socket`.

TMPDIR

Эта переменная может хранить путевое имя каталога временных файлов. Она эквивалентна опции `--tmpdir`.

USER

Эта переменная может хранить стандартное имя пользователя, указываемое при подключении к серверу MySQL. Она особенно полезна в среде Win32, где пользователь не всегда указывает свое имя при входе в систему.

Конфигурационные файлы

При запуске программа MySQL ищет конфигурационные файлы в нескольких каталогах. Анализ файлов осуществляется в определенном порядке. В UNIX общесистемные установки хранятся в файле `/etc/my.cnf`, а в Win32 — в файле `my.ini` в системном каталоге. В UNIX вслед за общесистемным файлом будет проверен одноименный файл в каталоге данных, а затем — файл `.my.cnf` в начальном каталоге пользователя. В Win32 вторым проверяется файл `my.cnf` в корневом каталоге диска C, а за ним — одноименный файл в каталоге данных.

Таким образом, в UNIX файлы проверяются в таком порядке:

```
/etc/my.cnf
/usr/local/mysql/data/my.cnf
/home/leon/.my.cnf
```

В Window NT порядок будет следующим:

```
c:\winnt\my.ini
c:\my.cnf
c:\mysql\data\my.cnf
```

Ни один из вышеперечисленных файлов не является обязательным, но если они все же присутствуют, то каждый последующий файл перекрывает установки предыдущего. Более того, переопределяются также значения, заданные в переменных среды.

Структура конфигурационного файла проста. Символ `#` объявляет начало комментария: весь текст до конца текущей строки будет проигнорирован интерпретатором. Границы наборов опций помечаются заголовками в квадратных скобках. Названия за-

головков соответствуют именам клиентских программ, а специальный заголовок [client] обозначает блок, общий для всех клиентов. Блок серверных опций называется [server].

Опции, задаваемые в конфигурационных файлах, называются так же, как и опции командной строки, только перед ними не ставятся дефисы. Знак равенства отделяет имя опции от присваиваемого ей значения (в командной строке операция присваивания может записываться иначе).

В листинге 14.1 приведен пример конфигурационного файла, включенного в стандартный дистрибутив. Перечисленные ниже утилиты могут иметь в нем свои записи: myisamchk, mysqlcheck, mysqlshow, myisampack, mysql, mysql.server, mysqladmin, mysqld, mysqld_multi, mysqldump, mysqlimport, safe_mysqld. Кроме того, утилита mysqld_multi создает отдельные заголовки для групп серверов.

```
[client]
port=3306

[mysqld]
port=3306,
default-character-set=latinl
set-variable = key_buffer=16M
set-variable = max_allowed_packet=1M
set-variable = thread_stack=128K
set-variable = flush_time=1800

[mysqldump]
quick
set-variable = max_allowed_packet=16M

[mysql]
no-auto-rehash
```

Полный список утилит

Ниже описаны все ключевые утилиты командной строки, имеющиеся в MySQL.

comp_err

По умолчанию сообщения об ошибках хранятся в каталоге /usr/local/share/mysql. Для каждого поддерживаемого языка создается подкаталог с txt-файлом и sys-файлом. Утилита comp_err компилирует текстовый файл в формат, понимаемый программой MySQL. Формат вызова утилиты таков:

```
comp_err
    [-? I -I]
    [-V]
    входной_файл выходной_файл
```

isamchk

Эта утилита восстанавливает и оптимизирует табличные файлы, хранящиеся в старом формате ISAM. Более современный ее аналог — утилита `myisamchk`.

make_binary_distribution

Этот сценарий создает двоичный дистрибутив. Если программа MySQL компилировалась из исходных файлов, сценарий make_binary_distribution будет находиться в подкаталоге `scripts`. Результатом работы сценария будет файл с расширением `.tar.gz`, предназначенный для инсталляции программы MySQL в аналогичной операционной системе.

msql2mysql

Этот сценарий пытается преобразовать исходный файл, содержащий обращения к базе данных mSQL, в эквивалентный файл MySQL. В нем используется утилита replace, с помощью которой функции mSQL заменяются своими аналогами из MySQL. Сценарий `msql2mysql` расположен в каталоге `scripts` и в качестве аргумента принимает имя преобразуемого файла.

my_print_defautls

Эта утилита анализирует заданные конфигурационные файлы и возвращает список содержащихся в них опций в формате, предназначенном для использования в командной строке. Синтаксис вызова утилиты my_print_defaults таков:

```
my_print_defaults
     [--config-file=файл | --defaults-file=файл I -c файл]
     [--extra-file=файл | --defaults-extra-file=файл | -e файл]
     [--help I -?]
      [--no-defaults I -n]
      [--version | -V]
     группа [группа]...
```

Эту утилиту удобно вызывать в сценариях, где требуется определить стандартные установки. Пример ее работы показан в листинге 14.2.

```
# my_print_defaults --config-file=/etc/my.cnf mysqld
--default-character-set=latin1
--set-variable=key_buffer=16M
--set-variable=max_allowed_packet=1M
--set-variable=thread_stack=128K
--set-variable=flush_time=1800
```

--config-file=файл (--defaults-file=файл) (-c)

Эта опция задает требуемый конфигурационный файл (по умолчанию анализируется стандартный файл).

--extra-file=файл (--defaults-extra-file=файл) (-e)

Эта опция задает конфигурационный файл, который читается после файла глобальных установок, но перед любым конфигурационным файлом, находящимся в начальном каталоге текущего пользователя.

-help (-?)

При наличии этой опции выдается лишь справочная информация об утилите `my_print_defaults`.

--no-defaults (-n)

При наличии этой опции возвращается пустая строка.

--version (-V)

При наличии этой опции возвращается лишь информация о версии утилиты.

myisamchk

Утилита myisamchk проверяет, восстанавливает и оптимизирует таблицы формата MyISAM. Она может принимать целый ряд опций и требует указания как минимум одной таблицы:

```
myisamchk
     [--analyze | -a]
     [--backup | -B]
     [--character-sets-dir=каталог]
     [--check | -c]
     [--check-only-changed | -C]
     [--data-file-length=длина | -D длина]
     [--debug[=конфигурация] | -# конфигурация]
     [--description | -d]
     [--extend-check | -e]
     [--fast | -F]
     [--force | -f]
     [--help | -?]
     [--information | -i]
     [--keys-used=битовое_поле | -k битовое_поле]
     [--medium-check | -m]
     [--no-symlinks | -l]
     [--quick | -q]
     [--read-only | -T]
     [--recover | -r]
     [--safe-recover | -o]
     [--set-auto-increment[=число] \ -A [число]]
     [--set-character-set=имя]
```

```
[--set-variable переменная=значение \ -O переменная=значение]
[--silent I -s]
[--sort-index | -S]
[--sort-records=индекс | -R индекс]
[--sort-recover | -n]
[--tmpdir=каталог | -t каталог]
[--unpack | -u]
[--update-state | -U]
[--verbose   -v]
[--version   -V]
[--wait | -w]
таблица[.MYI] [таблица[.MYI]] ...
```

При ссылке на таблицы разрешается использовать подстановочные знаки, чтобы не нужно было перечислять все таблицы. Расширение `.MYI` можно не указывать. В листинге 14.3 показано, как проверить все таблицы всех имеющихся баз данных.

```
myisamchk /usr/local/var/*/*.MYI
```

Если ни одна опция не указана, таблицы проверяются в режиме `--check`.

Утилита myisamchk должна вызываться в каталоге данных сервера. Она работает с `MYI`-файлами, в которых хранятся индексы таблиц MyISAM. Поскольку утилита может модифицировать эти файлы, необходимо заблокировать соответствующие таблицы или вообще остановить сервер. В противном случае любая "посторонняя" попытка записи в таблицу приведет к ее повреждению.

Вместо утилиты myisamchk можно воспользоваться инструкциями CHECK TABLE, OPTIMIZE TABLE или REPAIR TABLE. Все они вызываются из среды `mysql`.

Утилита myisamchk работает в одном из трех режимов: проверка, оптимизация или восстановление. В первом случае утилита лишь сообщает о наличии ошибок и может пометить таблицу как поврежденную, но никаких изменений в нее не вносится. Исключение делается только для опции `--force`, которая заставляет утилиту переключиться в режим восстановления в случае обнаружения какой-либо ошибки.

В режиме оптимизации улучшаются индексы. При наличии опции `--analyze` утилита проверяет ключи таблицы и значения, хранящиеся в ее столбцах. Это позволяет модулю оптимизации находить наилучшие способы объединения таблиц.

В режиме восстановления утилита пытается исправлять все ошибки. Обычно опции `--recover` вполне достаточно.

--analyze (-a)

Эта опция заставляет утилиту оптимизировать ключи таблицы, что позволит эффективнее выполнять табличные объединения.

--backup (-B)

Эта опция заставляет утилиту создавать резервные копии табличных файлов. К имени файла добавляется метка времени, а расширение меняется на `.BAK`.

--character-sets-dir=каталог

Эта опция задает каталог, в котором хранятся файлы наборов символов.

--check (-c)

Эта опция заставляет утилиту проверить таблицу на предмет наличия в ней ошибок. Если обнаруживается хотя бы одна ошибка, таблица помечается как поврежденная.

--check-only-changed (-C)

Эта опция заставляет утилиту проверять только те таблицы, которые изменились с момента последней проверки.

--data-file-length=длина (-D длина)

Эта опция задает максимальную длину восстанавливаемого файла данных.

--debug[=конфигурация] (-# конфигурация)

Эта опция заставляет утилиту записывать отладочную информацию в указанный журнальный файл. Поддержка данной опции должна быть включена в утилиту на этапе компиляции. По умолчанию данные записываются в файл `/tmp/client.trace`. Подробнее о формате журнальных файлов рассказывается в главе 24, "Физическое хранение данных".

--description (-d)

При наличии этой опции возвращается информация о заданной таблице.

--extend-check (-e)

Эта опция заставляет утилиту выполнить самую тщательную проверку таблиц. К данному средству следует прибегать в последнюю очередь, поскольку утилита `myisamchk` будет выполняться достаточно долго. Она проверит все ключи строка за строкой, используя индексный буфер. Следовательно, нужно предварительно увеличить размер буфера до максимально возможной величины. Если эта опция используется в режиме восстановления, будет выполнено полное восстановление всех возможных строк, включая те, которые были законно удалены.

-fast (-F)

Эта опция заставляет утилиту проверять только те таблицы, которые не были правильно закрыты.

--force (-f)

В режиме проверки эта опция заставляет утилиту переключиться в режим восстановления, как только будет обнаружена какая-либо ошибка. В режиме восстановления эта опция заставляет утилиту игнорировать временные файлы.

--help (-?)

При наличии этой опции возвращается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--information (-i)

При наличии этой опции выдается информация о каждой проверяемой таблице.

--keys-used=битовое_поле (-k битовое_поле)

Эта опция заставляет утилиту активизировать одни ключи и дезактивировать другие. Поскольку каждая операция вставки связана с обновлением индексов, дезактивация их приводит к ускорению операции. Впоследствии можно восстановить индекс с помощью инструкции `--recover`.

В качестве битового поля указывается целое число. Первая битовая цифра задает первый индекс, вторая — второй и т.д. Например, чтобы активизировать первый и четвертый индексы и отключить все остальные, нужно задать аргумент 9 (двоичная форма — 1001).

--medium-check (-m)

При наличии этой опции выполняется более детальная проверка таблиц, чем в случае опции `--check`, хотя и не такая тщательная, как в случае опции `--extended-check`.

--no-symlinks (-l)

Эта опция запрещает утилите проверять содержимое символических ссылок.

--quick (-q)

Эта опция заставляет утилиту восстанавливать только индексы и пропускать файлы данных. Если указана опция `-qq`, файлы данных будут обновляться только в случае обнаружения дубликатов.

--read-only (-T)

При наличии этой опции утилита `myisamchk` будет искать ошибки, но не помечать таблицу как поврежденную в случае их обнаружения. Это позволит не прерывать работу других пользователей на время проверки таблицы.

--recover (-r)

Эта опция заставляет утилиту устранять обнаруживаемые ошибки. В зависимости от определения таблицы возможен либо режим `--safe-recover`, либо `--sort-recover`.

--safe-recover (-o)

В этом режиме исправляется ряд ошибок, которые невозможно устранить в режиме `--sort-recover`. Кроме того, используется индексный буфер, поэтому желательно максимально увеличить его размер. В данном режиме восстановление таблицы происходит медленнее, чем в режиме `--sort-recover`, так что сначала лучше выполнить "быстрое" восстановление.

--set-auto-increment[=число] (-A [число])

Эта опция задает следующее значение поля-счетчика, если оно еще небыло использовано.

--set-character-set=имя

Эта опция задает используемый набор символов.

--set-variable переменная=значение (-O переменная=значение)

С помощью этой опции можно задавать значения некоторых переменных утилиты `myisamchk`. Список переменных и их стандартных значений приведен в табл. 14.1.

Переменная	*Значение по умолчанию*
decode_bits	9
key_buffer_size	520192
read_buffer_size	262136
sort_buffer_size	2097144
sort_key_blocks	16
write_buffer_size	262136

При восстановлении таблицы в режиме `--sort-recover` используется буфер сортировки. В этом случае операция восстановления выполняется достаточно быстро. Но бывают ситуации, когда утилита myisamchk вынуждена использовать индексный буфер. Например, этого требуют опции `--safe_recover` и `--extended_check`. Кроме того, индексный буфер задействуется, когда размер временного файла, используемого при сортировке индекса, становится в два раза больше, чем размер индексного файла. Таким образом, благодаря индексному буферу сокращается объем требуемого дискового пространства, но вместе с тем снижается производительность.

--silent (-s)

Эта опция приводит к сокращению объема выдаваемой информации. При наличии опции `-ss` будет возвращаться еще меньше информации.

--sort-index (-S)

Эта опция заставляет утилиту сортировать индексы, что приведет к ускорению операций сканирования таблиц.

--sort-records=индекс (-R индекс)

Эта опция заставляет утилиту сортировать строки файла данных по указанному номеру индекса, что приведет к ускорению операций поиска по диапазону значений.

--sort-recover (-n)

Эта опция заставляет утилиту использовать буфер сортировки даже в том случае, если размер временных таблиц окажется очень большим. В данном режиме, в отличие от режима `--safe-recover`, утилита не может обрабатывать дублирующиеся значения в столбцах, на которые наложено ограничение уникальности.

--tmpdir=каталог (-t каталог)

Эта опция задает каталог временных файлов.

--unpack (-u)

Эта опция заставляет утилиту `myisamchk` распаковать таблицу, упакованную утилитой `myisampack`.

--update-state (-U)

Эта опция заставляет утилиту обновить индексный файл, включив в него информацию о последней проверке.

--verbose (-v)

Эта опция переводит утилиту в "многословный" режим, в котором выдается больше информации, чем обычно. Существуют также опции `-vv` и `-vvv`, которые еще больше повышают информативность выходных данных. Их удобно применять совместно с опциями `--description` и `--extended-check`.

--version (-V)

При наличии этой опции выдается лишь информация о версии утилиты.

--wait (-w)

Эта опция заставляет утилиту дождаться разблокирования таблицы.

myisamlog

Утилита myisamlog **сканирует файл** `myisam.` log **и возвращает из него требуемую** информацию. **Синтаксис** вызова **утилиты** таков:

```
myisamlog
     [-?iruvDIPV]
     [-c предел]
     [-f число_файлов]
     [-F путь]
     [-o смещение]
     [-R]
     [-w файл]
     [-p число_компонентов]
     [журнальный_файл [таблица ...]]
```

-? (-I)

При наличии этой опции возвращается лишь справочная информация об утилите.

-c предел

Эта опция ограничивает вывод указанным числом команд.

-D

При наличии этой опции утилита проверяет, скомпилирован ли демон `mysqld` в режиме включения отладочной информации.

-F путь

Эта опция задает используемое путевое имя.

-f файлы

Эта опция задает максимальное число открытых файлов.

-o смещение

Эта опция заставляет утилиту начать сканирование файла с указанного смещения.

-P

При наличии этой опции возвращается информация о процессах.

-p компоненты

Эта опция заставляет утилиту удалять заданное число компонентов из путевых имен.

-R

Эта опция заставляет утилиту отображать позицию записи.

-r

При наличии этой опции будет выдана статистика операций восстановления.

-U

При наличии этой опции будет выдана статистика операций обновления.

-V

Эта опция заставляет утилиту выдавать развернутую информацию. Если указана опция **-vv**, отчет будет еще более подробным.

-V

При наличии этой опции возвращается информация о версии утилиты.

-w

При наличии этой опции будет выдана статистика операций записи в файлы.

myisampack

Эта утилита создает сжатые таблицы, доступные только для чтения. Синтаксис ее вызова таков:

```
myisampack
    [--backup | -b]
    [--debug[=конфигурация] | -# конфигурация]
    [--force | -f]
    [--join=таблица | -j таблица]
    [--help | -?]
    [--packlength=длина | -p длина]
    [--silent | -s]
    [--temp_dir=каталог | -T каталог]
    [--test | -t]
    [--verbose | -v]
    [--version | -V]
    [--wait | -w]
    файл[.MSI]...
```

После **сжатия таблицы не забудьте** перестроить **индексы с** помощью **утилиты** myisamchk. **Команда** `myisamchk --unpack` позволяет распаковать **сжатую таблицу.**

--backup (-b)

При наличии этой опции будут созданы резервные копии таблиц, имеющие расширение `.OLD`.

--debug[=конфигурация] (-# конфигурация)

Эта опция заставляет утилиту записывать отладочную информацию в указанный журнальный файл. Поддержка данной опции должна быть включена в утилиту на этапе компиляции. По умолчанию данные записываются в файл `/tmp/client.trace`. Подробнее о формате журнальных файлов рассказывается в главе 24, "Физическое хранение данных".

--force (-f)

Эта опция заставляет утилиту сжимать файл даже в том случае, если это приведет к увеличению его размера. Временные файлы (имеют расширение .TMD) также игнорируются. Они создаются и удаляются в процессе сжатия таблиц, но могут остаться в системе, если операция внезапно прерывается.

--join=таблица (-j таблица)

При наличии этой опции все входные таблицы будут объединены в одну новую таблицу, при условии, что они имеют одинаковую структуру.

--help (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--packlength=длина (-p длина)

Эта опция задает размер указателей на записи. Аргумент определяет число байтов и должен быть равен 1, 2 или 3. Обычно утилита сама находит правильный размер указателей. Иногда она сообщает о том, что можно использовать более короткий индекс.

--silent (-s)

Эта опция подавляет вывод информационных сообщений. Будут выдаваться только сообщения об ошибках.

--temp_dir=каталог (-T каталог)

Эта опция задает каталог для временных файлов.

-test (-t)

Эта опция переводит утилиту в тестовый режим, в котором таблицы в действительности не сжимаются.

--verbose (-v)

При наличии этой опции будет выдаваться информация о том, как протекает процесс сжатия таблиц.

--version (-V)

При наличии этой опции будет выдана лишь информация о версии утилиты.

--wait (-w)

Эта опция заставляет утилиту дождаться разблокирования таблиц.

mysql

Утилита mysql **является,** очевидно, **самым важным** инструментом пользователя **MySQL. Она** позволяет посылать **базам данных запросы в** интерактивном **и пакетном режимах.** Формат **ее** вызова таков:

```
mysql
     [--batch | -B]
     [--character-sets-dir=каталог]
     [--compress I -C]
      [--database=имя | -D имя]
      [--debug-info | -T]
     [--debug[=конфигурация] \ -# конфигурация]
    [--default-character-set=файл]
    [--defaults-extra-file=файл]
     [--defaults-file=файл]
     [--enable-named-commands | -G]
     [--execute=команда I -e команда]
    [--force | -f]
     [--help | -?]
     [--host=узел | -h узел]
    [--html | -H]
     [--ignore-space | -i]
      [--no-auto-rehash | -A]
     [--no-defaults]
     [--no-named-commands | -g]
     [--no-pager] .
     [--notee]
      [--one-database I -o]
      [--pager[=команда]
     [--password[=пароль] I -p[пароль]]
      [--pipe I -W]
       --port=порт | -P порт]
       --print-defaults
       --quick | -q]
       --raw | -r]
       --safe-updates[=максимум] \ -U максимум]
       --set-variable переменная=значение \ -O переменная=значение]
      [--silent I -s]
```

можно пользоваться клавишами управления курсором для перемещения по набранному тексту (стрелки влево/вправо) и для вызова команд из перечня ранее введенных (стрелки вверх/вниз). Имеется также множество специальных команд, активизируемых при нажатии управляющих клавиш, но их рассмотрение выходит за рамки данной книги (соответствующую информацию можно найти по адресу *http://cnswww.cns.cwru. edu/~chet/readline/rltophtml*).

Версия утилиты `mysql` для платформы Win32 не поддерживает библиотеку `readline`, так как она не является частью Windows. Но есть другая утилита, которая называется `mysqlc` и работает с библиотекой readline, входящей в пакет `cygwin`. Эта утилита будет рассмотрена ниже.

В пакетном режиме утилита mysql принимает инструкции по каналу. Например, команда, показанная в листинге 14.4, зарегистрирует пользователя admin на сервере MySQL и выполнит инструкции, содержащиеся в файле `build.sql`, как если бы эти инструкции были введены в командной строке. Прежде чем сценарий начнет выполняться, программа попросит пользователя ввести пароль.

```
mysql --user=admin --password < build.sql
```

Переменные

Помимо упомянутых в начале главы переменных среды утилита mysql проверяет значения двух других переменных: `MYSQL_HISTFILE` и MYSQL_HOST.

MYSQL_HISTFILE

По умолчанию перечень введенных команд хранится в файле `.mysql_history`, находящемся в начальном каталоге пользователя. Переменная среды MYSQL_HISTFILE позволяет задать другое местоположение файла.

MYSQL_HOST

В этой переменной хранится адрес стандартного узла. Если она не задана, утилита будет подключаться к локальному узлу.

Опции

--batch(-B)

Эта опция переводит утилиту mysql в пакетный режим. Поля результатов запроса будут разделяться символами табуляции, а не так, как в интерактивном режиме. Данные, представленные в таком формате, удобнее передавать по каналу другой программе. Клавиши управления курсором в пакетном режиме не работают.

Если SQL-инструкции передаются утилите mysql посредством оператора канала (|) или оператора переадресации (<), то утилита автоматически переходит в пакет-

ный режим и опция `--batch` не нужна. Если же опция указана, но инструкции не введены, утилита перейдет в интерактивный режим, правда, с некоторыми особенностями. В частности, приглашение интерпретатора команд не отобразится, хотя утилита будет ожидать ввод команд.

--character-sets-dir=каталог

Эта опция задает каталог, где хранятся файлы наборов символов. Процесс создания таких файлов описан в главе 31, "Расширение возможностей MySQL". По умолчанию программа MySQL работает с набором Latin, известным как ISO8859-1. Эту установку можно изменить на этапе компиляции программы или же с помощью опции `--default-character-set`.

--compress (-C)

Эта опция заставляет утилиту посылать серверу данные в сжатом виде, что бывает удобно, когда соединение с сервером является очень медленным.

--database=имя (-D имя)

Эта опция задает базу данных, с которой будет вестись работа в текущем сеансе. То же самое делает инструкция USE. Имя базы данных можно также указать в качестве последнего аргумента утилиты `mysql`.

--debug[=конфигурация] (-# конфигурация)

Эта опция заставляет утилиту записывать отладочную информацию в указанный журнальный файл. Поддержка данной опции должна быть включена в утилиту на этапе компиляции. По умолчанию данные записываются в файл `/tmp/client.trace`. Подробнее о формате журнальных файлов рассказывается в главе 24, "Физическое хранение данных".

--debug-info (-T)

Эта опция заставляет утилиту выдавать отладочную информацию при завершении работы. В листинге 14.5 показан отчет о коротком сеансе.

```
User time 0.01, System time 0.00
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 72, Physical pagefaults 187, Swaps 0
Blocks in 0 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 0, Involuntary context switches 0
```

--default-character-set=файл

Эта опция меняет стандартный набор символов, установленный на этапе компиляции программы MySQL. Обычно таковым является набор Latin (ISO8859-1). Указываемый файл должен находиться в каталоге, который был задан с помощью опции `--character-sets-dir`. Этот файл представляет собой не просто коллекцию символов. В нем описаны правила сортировки алфавита и трансляции каждого символа в верхний и нижний регистры.

--defaults-extra-file=файл

Данная опция заставляет утилиту прочитать дополнительный конфигурационный файл. Это происходит после анализа файла глобальных установок.

--defaults-file=файл

Эта опция позволяет переопределить стандартный конфигурационный файл.

--enable-named-commands (-G)

Эта опция разрешает использовать длинные версии команд интерпретатора (так называемые *именованные команды)*. Если именованные команды запрещены, нужно либо пользоваться их короткими аналогами, либо вводить их в отдельной строке, оканчивающейся точкой с запятой. Если же такие команды разрешены, необходимо внимательно следить за тем, чтобы случайно не активизировать одну из них. Это может произойти, если часть SQL-инструкции начинается с именованной команды.

Рассмотрим многострочную инструкцию, показанную в листинге 14.6. Здесь имя столбца source совпадает с названием стандартной команды. Если именованные команды разрешены, утилита `mysql` ошибочно воспримет вторую строку как запрос на выполнение файла, а не как часть списка возвращаемых столбцов.

```
SELECT id,
source, body
    FROM news;
```

--execute=команда (-e команда)

Эта опция заставляет утилиту выполнить указанную инструкцию, которая должна быть заключена в одинарные кавычки (листинг 14.7).

```
mysql --execute='SELECT * FROM user' -u leon -p -B freetrade
```

Показанная инструкция отобразит содержимое таблицы `user` в пакетном режиме.

--force (-f)

Эта опция заставляет утилиту продолжать обработку входных данных даже в случае обнаружения ошибки. Имеет смысл применять ее в пакетном режиме при обработке длинных файлов.

--help (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--host=узел (-h узел)

Эта опция задает узел, к которому будет пытаться подключиться утилита. По умолчанию связь устанавливается с локальным узлом. В качестве аргумента может присутствовать доменное имя или IP-адрес.

--html (-H)

Эта опция заставляет утилиту отображать результаты запроса в формате HTML. Применять ее имеет смысл лишь в пакетном режиме.

--ignore-space (-i)

Эта опция заставляет утилиту игнорировать пробелы, следующие за именем функции. Обычно за именем функции должна идти открывающая скобка.

--no-auto-rehash (-A)

Если в имеющейся версии `mysql` применяется библиотека `readline`, то можно пользоваться функцией автоматического дополнения имен таблиц и столбцов. При нажатии клавиши табуляции интерпретатор предложит свой вариант дополнения имени. Опция `--no-auto-rehash` запрещает утилите загружать список всех имен, благодаря чему утилита загружается немного быстрее. Режим автодополнения можно включить позднее с помощью команды rehash (\r).

--no-defaults

Эта опция запрещает утилите анализировать конфигурационный файл.

--no-named-commands (-g)

Эта опция запрещает именованные команды — длинные версии команд, начинающихся с обратной косой черты. Такая установка принята по умолчанию. Противоположное действие выполняет опция `--enable-named-commands`.

--no-pager

Эта опция применима только в отношении UNIX-версий утилиты `mysql`. Она запрещает передавать выходные данные утилите постраничной разбивки.

--notee

Эта опция запрещает запись выходных данных в файл. Такая установка принята по умолчанию.

--one-database (-o)

Эта опция заставляет утилиту обновлять только стандартную базу данных. Подобный режим полезен при восстановлении базы данных на основании журналов обновлений, о чем рассказывается в главе 25, "Устранение последствий катастроф".

--pager[=команда]

Эта опция заставляет утилиту mysql посылать выходные данные указанной команде (обычно программе постраничной разбивки, например more).

--password[=пароль] (-p[пароль])

Эта опция задает пароль для подключения к серверу. При ее отсутствии пароль не посылается. Если же пароль не указан, программа попросит его ввести. Учтите, что пароль, вводимый в командной строке, могут увидеть другие пользователи, просматривающие список процессов. Обратите также внимание на то, что между названием короткой версии опции и паролем не нужен пробел.

--pipe (-W)

Эта опция заставляет утилиту взаимодействовать с именованными каналами, но они доступны лишь в том случае, когда клиент работает в Windows NT. В остальных системах взаимодействие с сервером осуществляется посредством протокола TCP.

--port=порт (-P порт)

Эта опция переопределяет стандартный номер порта, с которым работает утилита (по умолчанию — 3306).

--print-defaults

При наличии этой опции утилита отобразит список своих установок и завершит работу.

--quick (-q)

Эта опция заставляет утилиту не **кэшировать** результаты запроса, а отображать их строка за строкой. Она также отключает перечень ранее введенных команд. Учтите, что работа сервера замедлится, если вывод данных будет приостановлен программой постраничной разбивки.

--raw (-r)

Эта опция применяется в пакетном режиме. Она отменяет стандартную функцию преобразования непечатаемых символов в управляющие последовательности. Например, в интерактивном **режиме**, если ячейка содержит символ новой строки, он будет воспринят буквально, и текст начнется с новой строки. В пакетном режиме этого не произойдет, но отобразится код \n. Опция `--raw` влияет на символы табуляции (\t), новой строки (`\n`), обратной косой черты (`\\`), а также на первые 10 ASCII-символов (\0 до `\09`).

--safe-updates[=максимум] (-U максимум)

Эта опция позволяет защитить пользователей-новичков от случайного выполнения инструкции DELETE или UPDATE, затрагивающей всю таблицу. Запросы, в которых в предложении WHERE не указан первичный ключ, отклоняются. Если задан аргумент, будет изменено значение переменных SQL_SELECT_LIMIT и `SQL_MAX_JOIN_SIZE`. У данной опции есть синоним: `--i-am-a-dummy`.

--set-variable переменная=значение (-O переменная=значение)

Эта опция позволяет установить значения некоторых переменных, в частности `max_allowed_packet`, `net_buffer_length`, `select_limit` и `max_join_size`.

--skip-column-names (-N)

При наличии этой опции названия столбцов не включаются в результаты запроса.

--skip-line-numbers (-L)

Эта опция подавляет вывод номеров строк в сообщениях об ошибках. Изначально номера строк появляются только в пакетном режиме. Приведем пример сообщения об ошибке:

```
ERROR 1054 at line 2: Unknown column 'check' in 'field list'
```

В случае опции `--skip-line-numbers` строка будет выглядеть следующим образом:

```
ERROR 1054: Unknown column 'check' in 'field list'
```

--silent (-s)

Эта опция приводит к сокращению объема информации, возвращаемой клиенту. Например, не будет выдано приветственное сообщение, а результаты запросов не будут заключаться в рамки.

--socket=файл (-S файл)

Эта опция переопределяет стандартный сокет, используемый для подключения к серверу. Обычно файл сокета называется `/tmp/mysql.sock`.

--table (-t)

Эта опция заставляет утилиту выдавать результаты запроса в стандартном формате. Значения ячеек будут отделяться друг от друга пробелами и символами вертикальной черты. В пакетном режиме столбцы разделяются символами табуляции без какого бы то ни было дополнительного оформления.

--tee=файл

Эта опция заставляет утилиту записывать выходные данные в указанный файл и доступна только в UNIX-версии утилиты.

--unbuffered (-n)

Эта опция заставляет утилиту очищать выходной буфер после каждого запроса. Такой режим удобен при передаче данных по каналу другой программе.

--user=имя (-u имя)

Эта опция задает имя пользователя, указываемое при регистрации на сервере. По умолчанию берется имя текущего пользователя, под которым он зарегистрировался в локальной системе.

--verbose (-v)

Эта опция заставляет утилиту выдавать более подробные сообщения. Опцию можно указывать несколько раз подряд, чтобы получать все более детальный отчет о работе утилиты.

--version (-V)

При наличии этойопции будет выдана лишь информация о версии утилиты.

-vertical (-E)

Эта опция заставляет утилиту выдавать результаты запроса в вертикальном виде. Значения каждого столбца будут размещаться в отдельной строке.

--wait (-w)

При наличии этой опции утилита будет пытаться подключиться к серверу до тех пор, пока очередная попытка не увенчается успехом.

Команды

В интерактивном режиме утилита `mysql` реагирует на SQL-инструкции и на специальные команды. Инструкции могут занимать несколько строк, а символ новой строки воспринимается как обычный символ пробела. Приглашение интерпретатора команд обычно выглядит так: `mysql>`. После первой строки многострочной инструкции выдается приглашение `->`. Если начать строковый литерал, а затем нажать <Enter>, утилита сообщит об **этом**, отобразив в приглашении `'>` или `">`. Конец строки помечается точкой с запятой.

Команды — это специальные ключевые слова, распознаваемые интерпретатором. Их можно записывать двумя способами: в виде буквы с предшествующей обратной косой чертой (короткая форма) или в виде слова *(именованная команда)*. Короткие команды могут присутствовать где угодно, а именованные команды по умолчанию должны начинать строку.

clear (\c)

Эта команда отменяет выполнение текущей инструкции, удаляя ее из буфера. Используйте команду `clear`, если при вводе многострочной инструкции вдруг обнаружилось, что в предыдущей строке содержится ошибка.

connect [база_данных [узел]] (\r [база_данных [узел]])

Эта команда заставляетутилиту повторно подключиться к серверу. Можно указать новую базу данных и новый узел.

edit (\e)

Эта команда вызывает внешний редактор для ввода текста запроса. Имя редактора содержится в переменной среды EDITOR. Если часть запроса уже введена, она будет доступна для редактирования. При выходе из редактора будет вновь вызван интерпретатор mysql. Данная команда недоступна в Win32.

ego (\G)

Эта команда заставляет утилиту выполнить текущую инструкцию и отобразить ее результаты в вертикальном стиле (см. описание опции `--vertical`).

exit (\q)

Эта команда завершает сеанс работы с интерпретатором команд.

до (\д)

Эта команда заставляет утилиту выполнить текущую инструкцию.

help (\h, \?)

Эта команда отображает справочную информацию о доступных командах.

nopager (\n)

Эта команда запрещает передавать результаты запросов программе постраничной разбивки.

pager [команда] (\P команда)

Эта команда заставляет утилиту `mysql` посылать результаты запросов указанной программе постраничной разбивки. То же самое делает описанная выше опция `--pager`.

print (\p)

Эта команда отображает текущую инструкцию, содержащуюся в буфере.

quit (\q)

Это синоним команды `exit`.

rehash (\#)

Эта команда включает режим автоматического дополнения имен.

source файл (\. файл)

Эта команда заставляет утилиту выполнить инструкции, содержащиеся в указанном текстовом файле.

status (\s)

Эта команда возвращает информацию о клиенте и сервере (листинг 14.8).

```
mysql> status

mysql Ver 11.15 Distrib 3.23.37, for pc-linux-gnu (i586)

Connection id:          13
Current database:
Current user:           root@localhost
```

```
Current pager:          stdout
Using outfile:        ' '
Server version:         3.23.37
Protocol version:       10
Connection:             Localhost via UNIX socket
Client characterset:    latin1
Server characterset:    latin1
UNIX socket:            /tmp/mysql.sock
Uptime:                 1 day 4 hours 52 min 18 sec

Threads: 5  Questions: 299  Slow queries: 0  Opens: 153
Flush tables: 1  Open tables: 0  Queries per second avg: 0.003
```

tee [файл] (\T [файл])

Эта команда заставляет утилиту посылать выходные данные не только на экран, но и в файл. Данный режим выключается командой `notee`. Если файл не указан, берется предыдущее путевое имя.

use база_данных (\и база_данных)

Эта команда позволяет сменить базу данных.

mysql_install_db

Этот сценарий создает стандартные таблицы привилегий в базе данных `mysql`. Обычно он запускается один раз при первой инсталляции программы MySQL. Если таблицы привилегий уже существуют, сценарий ничего не делает.

mysqlaccess

Этот **Perl-сценарий** проверяет **привилегии заданных узла,** пользователя **и базы данных. Синтаксис** его вызова таков:

```
mysqlaccess [узел [пользователь [база_данных]]]
    [--brief I -b]
    [--commit]
    [--copy]
    [--db=база_данных | -d база_данных]
    [--debug=уровень]
    [--help | -?]
    [--host=имя | -h имя]
    [--howto]
    [--old_server]
    [--password=пароль | -p пароль]
    [--plan]
    [--preview]
    [--relnotes]
    [  rhost=имя | -H имя]
```

```
[--rollback]
[--spassword=пароль | -P пароль]
[--superuser=имя | -U имя]
[--table | -t]
[--user=имя | -и]
[--version | -v]
```

Вместо имени узла, пользователя или базы данных может стоять метасимвол *****, обозначающий всю совокупность значений. Этот метасимвол необходимо взять в одинарные кавычки.

mysqladmin

Утилита mysqladmin выполняет ряд административных задач. Формат ее вызова таков:

```
mysqladmin
    [--character-sets-dir=каталог]
    [--compress | -C]
    [--debug[=конфигурация] | -# конфигурация]
    [--force | -f]
    [--help | -?]
    [--host=узел | -h узел]
    [--password[=пароль] | -p[пароль]]
    [--port=порт | -P порт]
    [--relative | -r]
    [--silent | -s]
    [--sleep=секунды | -i секунды]
    [--socket=файл | -S файл]
    [--user=имя | -u имя]
    [--verbose | -v]
    [--version | -V]
    [--vertical | -E]
    [--wait[=число_попыток] \ -w число_попыток]
    [create имя]
    [drop имя]
    [extended-status]
    [flush-hosts]
    [flush-logs]
    [flush-privileges]
    [flush-status]
    [flush-tables]
    [flush-threads]
    [kill процесс,процесс,...]
    [password новый_пароль]
    [ping]
    [processlist]
    [refresh]
    [reload]
    [shutdown]
    [start-slave]
    [status]
    [stop-slave]
    [variables]
    [version]
```

Утилита подключается к серверу и посылает ему одну или несколько команд, обычно разрешенных лишь администратору базы данных.

--character-sets-dir=каталог

Эта опция задает каталог, где хранятся файлы наборов символов. Процесс создания таких файлов описан в главе 31, "Расширение возможностей MySQL". По умолчанию программа MySQL работает с набором Latin, известным как ISO8859-1. Эту установку можно изменить на этапе компиляции программы или же с помощью опции `--default-character-set`.

--compress (-C)

Эта опция заставляет утилиту посылать серверу данные в сжатом виде, что бывает удобно, когда соединение с сервером является очень медленным.

--debug[=конфигурация] (-# конфигурация)

Эта опция заставляет утилиту записывать отладочную информацию в указанный журнальный файл. Поддержка данной опции должна быть включена в утилиту на этапе компиляции. По умолчанию данные записываются в файл `/tmp/client.trace`. Подробнее о формате журнальных файлов рассказывается в главе 24, "Физическое хранение данных".

--force (-f)

Эта опция заставляет утилиту продолжать обработку входных данных даже в случае обнаружения ошибки. При удалении базы данных подтверждение не потребуется.

--help (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--host=узел (-h узел)

Эта опция задает узел, к которому будет пытаться подключиться утилита. По умолчанию связь устанавливается с локальным узлом. В качестве аргумента может присутствовать доменное имя или IP-адрес.

--password[=пароль] (-p[пароль])

Эта опция задает пароль для подключения к серверу. При ее отсутствии пароль не посылается. Если же пароль не указан, программа попросит его ввести. Учтите, что пароль, вводимый в командной строке, могут увидеть другие пользователи, просматривающие список процессов. Обратите также внимание на то, что между названием короткой версии опции и паролем не нужен пробел.

--port=порт (-P порт)

Эта опция переопределяет стандартный номер порта, с которым работает утилита (по умолчанию — 3306).

--relative (-r)

Эта опция меняет формат вывода команды `extended-status` в режиме `--sleep`. Вместо подсчета итоговых сумм будут вычисляться разности между текущими и предыдущими значениями.

--silent (-s)

Эта опция сокращает объем информации, возвращаемой утилитой.

--sleep=секунды (-i секунды)

Эта опция заставляет утилиту непрерывно выполняться в течение указанного времени. В сочетании с одной из статусных команд она позволяет превратить терминальное окно в постоянно обновляющийся системный монитор.

--socket=файл (-S файл)

Эта опция переопределяет стандартный сокет, используемый для подключения к серверу. Обычно файл сокета называется `/tmp/mysql.sock`.

--user=имя (-u имя)

Эта опция задает имя пользователя, указываемое при регистрации на сервере. По умолчанию берется имя текущего пользователя, под которым он зарегистрировался в локальной системе.

--verbose (-v)

Эта опция заставляет утилиту выдавать более подробные сообщения. Опцию можно указывать несколько раз подряд, чтобы получать все более детальный отчет о работе утилиты.

--version (-V)

При наличии этой опции будет выдана лишь информация о версии утилиты.

--vertical (-E)

Эта опция заставляет утилиту выдавать результаты запроса в вертикальном виде. Значения каждого столбца будут размещаться в отдельной строке.

--wait (-w)

При наличии этой опции утилита будет пытаться подключиться к серверу до тех пор, пока очередная попытка не увенчается успехом.

create имя

Эта команда создает базу данных и является аналогом инструкции CREATE DATABASE.

drop имя

Эта команда удаляет базу данных и является аналогом инструкции DROP DATABASE.

extended-status

Эта команда возвращает расширенную информацию о состоянии сервера. Сообщаются значения различных переменных, а также некоторые статистические данные, например общее число подключений и суммарное время работы (листинг 14.9). Аналогичные данные выдает инструкция SHOW STATUS. При наличии опций `--sleep` и `--relative` отображаемая информация будет непрерывно обновляться.

```
# mysqladmin extended-status
+                                +       +
I Variable name                  | Value I
'                                '       '
  Aborted_clients                  0
  Aborted_connects                 0
  Bytes_received                   411
  Bytes_sent                       12630
  Connections                      19
  Created_tmp_disk_tables          0
  Created_tmp_tables               0
  Created_tmp_files                0
  Delayed_insert_threads           0
  Delayed_writes                   0
  Delayed_errors                   0
  Flush_commands                   1
  Handler_delete                   0
  Handler_read_first               0
  Handler_read_key                 0
  Handler_read_next                0
  Handler_read_prev                0
  Handler_read_rnd                 0
  Handler_read_rnd_next            0
  Handler_update                   0
  Handler_write                    0
  Key_blocks_used                  5
  Key_read_requests                0
  Key_reads                        0
  Key_write_requests               0
  Key_writes                       0
  Max_used_connections             2
  Not_flushed_key_blocks           0
  Not_flushed_delayed_rows         0
  Open_tables                      0
  Open_files                       0
  Open_streams                     0
```

```
| Opened_tables                 | 0     |
| Questions                     | 49    |
| Select_full_join              | 0     |
| Select_full_range_join        | 0     |
| Select_range                  | 0     |
| Select_range_check            | 0     |
| Select_scan                   | 0     |
| Slave_running                 | OFF   |
| Slave_open_temp_tables        | 0     |
| Slow_launch_threads           | 0     |
| Slow_queries                  | 0     |
| Sort_merge_passes             | 0     |
| Sort_range                    | 0     |
| Sort_rows                     | 0     |
| Sort_scan                     | 0     |
| Table_locks_immediate         | 0     |
| Table_locks_waited            | 0     |
| Threads_cached                | 0     |
| Threads_connected             | 18    |
| Threads_created               | 3     |
| Threads_running               | 1     |
| Uptime                        | 3648  |
+-------------------------------+-------+
```

flush-hosts

Эта команда выполняет инструкцию FLUSH HOSTS, которая очищает кэш имен компьютеров и адресов Internet. Когда компьютер меняет свой IP-адрес, данный буфер нужно очистить. Это также позволит заново установить соединения, которые ранее были отменены из-за превышения лимита ошибок.

flush-logs

Эта команда выполняет инструкцию FLUSH LOGS, которая закрывает и повторно открывает все журнальные файлы. Нумерованные файлы получат новые номера.

flush-privileges

Эта команда выполняет инструкцию FLUSH PRIVILEGES, которая повторно загружает таблицы привилегий в базуданных `mysql`. Это необходимо делать после ручного редактирования таблиц.

flush-status

Эта команда выполняет инструкцию FLUSH STATUS, которая сбрасывает большинство значений, сообщаемых инструкцией SHOW TABLE STATUS.

flush-tables

Эта команда выполняет инструкцию FLUSH TABLES, которая закрывает все таблицы.

flush-threads

Эта команда очищает буфер потоков.

kill процесс, [процесс,...]

Эта команда разрывает одно или несколько соединений. Получить список активных соединений можно с помощью команды `processlist`.

password новый_пароль

Эта команда меняет пароль администратора. Учтите, что пароль, вводимый в командной строке, могут увидеть другие пользователи, просматривающие список процессов.

ping

Эта команда посылает серверу **эхо-запрос**, проверяя, функционирует ли сервер.

processlist

Эта команда возвращает информацию о соединениях, установленных с сервером. Аналогичные данные можно получить с помощью инструкции SHOW PROCESSLIST. В первом столбце возвращаемой таблицы указываются идентификаторы соединений. Это те значения, которые необходимо передавать команде `kill`.

refresh

Эта команда закрывает таблицы и журнальные файлы.

reload

Эта команда повторно загружает таблицы привилегий.

shutdown

Эта команда останавливает сервер. Для его повторного запуска нужно вызвать демон `mysqld`. В среде Windows NT и Windows 2000 сервер запускается командой NET START `mysql`. В UNIX этой цели служит сценарий, располагаемый в каталоге `init.d`, например:

```
# /etc/init.d/mysql start
```

start-slave

Эта команда запускает подчиненный сервер.

status

Эта команда возвращает статусное сообщение, в котором указаны общее время работы сервера, число активных потоков, общее число запросов и среднее число запросов в секунду (рис. 14.10).

```
Uptime: 3914  Threads: 3  Questions: 63  Slow queries: 0  Opens: 0
Flush tables: 1  Open tables: 0  Queries per second avg: 0.016
```

stop-slave

Эта команда останавливает подчиненный сервер.

variables

Эта команда выполняет инструкцию SHOW VARIABLES (см. главу 13, "Инструкции SQL").

version

Эта команда возвращает информацию о версии сервера. Сюда же включаются и данные о его состоянии (листинг 14.11).

```
[root@red leon]# mysqladmin version
mysqladmin Ver 8.19 Distrib 3.23.37, for pc-linux-gnu on i586
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free
software,
and you are welcome to modify and redistribute it under the GPL
license

Server version          3.23.37
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /tmp/mysql.sock
Uptime:                 56 min 36 sec

Threads: 3  Questions: 42  Slow queries: 0  Opens: 0  Flush
tables: 1  Open tables: 12
Queries per second avg: 0.012
```

mysqlbinlog

Эта утилита преобразует записи двоичного журнального файла в текстовую форму или в SQL-инструкции. Синтаксис вызова утилиты таков:

```
mysqlbinlog
     [--help I -?]
     [--host=сервер | -h сервер]
     [--offset=записи | -o записи]
     [--password=пароль \ -p пароль]
     [--port=порт I -P порт]
```

```
[--position=байты | -j байты]
[--short-form | -s]
[--table=имя | -t имя]
[--user=имя | -и имя]
[--version | -V]
журнальный_файл [журнальный_файл]...
```

После списка опций следует перечень журнальных файлов. Результаты работы записываются в поток stdout.

--help (-?)

При наличии этой опции возвращается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--host=сервер (-h сервер)

Эта опция переопределяет стандартный сервер (по умолчанию — localhost).

--offset=записи (-o записи)

Эта опция заставляет утилиту пропустить указанное число записей файла.

--password=пароль (-p пароль)

Эта опция задает пароль для подключения к серверу.

--port=порт (-P порт)

Эта опция переопределяет стандартный номер порта для подключения к серверу.

--position=байты (-j байты)

Эта опция заставляет утилиту начать чтение файла с указанной позиции.

--short-form (-s)

При наличии этой опции утилита будет выдавать лишь информацию о запросах.

--table=имя (-t имя)

При наличии этой опции возвращается информация об указанной таблице.

--user=имя (-u имя)

Эта опция задает имя пользователя, указываемое при регистрации на сервере.

--version (-V)

При наличии этой опции возвращается лишь информация о версии утилиты.

mysqlbug

Сценарий mysqlbug посылает отчет о найденных ошибках команде разработчиков MySQL. В этот отчет должна быть включена тщательно собранная информация о сервере, включая его версию, название операционной системы и параметры конфигурации.

Данный сценарий автоматизирует процесс сбора информации, но описание ошибки необходимо ввести самостоятельно. С этой целью запускается текстовый редактор, где подсказывается, какие сведения следует указать. При выходе из редактора сообщение будет направлено в соответствующий список рассылки программы MySQL.

mysqlc

Это версия утилиты `mysql`, которая работает с библиотекой `readline` в среде Win32. Она требует наличия в системе DLL-файла пакета `cygwin`.

mysqld

Демон mysqld представляет собой сервер MySQL. Он работает в фоновом режиме, принимая запросы от клиентов. Синтаксис вызова демона таков:

```
mysqld
    [--ansi]
    [--basedir=каталог | -b каталог]
    [--bdb-home=каталог]
    [--bdb-lock-detect={DEFAULT | OLDEST | RANDOM | YOUNGEST}]
    [--bdb-logdir=каталог]
    [--bdb-no-recover]
    [--bdb-no-sync]
    [--bdb-shared-data]
    [--bdb-tmpdir=каталог]
    [--big-tables]
    [--bind-address=адрес]
    [--binlog-do-db=база_данных]
    [--binlog-ignore-db=база_данных]
    [--bootstrap]
    [--character-sets-dir=каталог]
    [--chroot=каталог]
    [--console]
    [--core-file]
    [--datadir=каталог | -h каталог]
    [--default-character-set=имя]
    [--default-table-type=тип]
    [--delay-key-write-for-all-tables]
    [--enable-locking]
    [--exit-info=[битовое_поле] | -T [битовое_поле]]
    [-- flush]
    [--gemini-full-recovery]
    [--gemini-lazy-commit]
    [--gemini-no-recovery]
    [--gemini-unbuffered-io]
    [--help | -?]
```

```
[--init-file=файл]
[--innodb_data_file_path=файл:размер[;файл:размер]]
[--innodb_data_home_dir=каталог]
[--innodb_flush_log_at_trx_commit={0 | 1}]
[--innodb_log_archive[=число]]
[--innodb_log_arch_dir=каталог]
[--innodb_log_group_home_dir=каталог]
[--install]
[--language=каталог | -L каталог]
[--log[=каталог] | -l [каталог]]
[--log-bin-index=каталог]
[--log-bin[=каталог]]
[--log-isam[=каталог]]
[--log-long-format]
[--log-slave-updates]
[--log-slow-queries[=каталог]]
[--log-update[=каталог]]
[--low-priority-updates]
[--master-connect-retry=секунды]
[--master-host=узел]
[--master-info-file=файл]
[--master-password=пароль]
[--master-port=порт]
[--master-user=пользователь]
[--memlock]
[--myisam-recover[=опция[,опция...]]]
[--new | -n]
[--old-protocol | -o]
[--pid-file=файл]
[--port=порт | -P порт]
[--remove]
[--replicate-do-db=база_данных]
[--replicate-do-table=база_данных.таблица]
[--replicate-ignore-db=база_данных]
[--replicate-ignore-table=база_данных.таблица]
[--replicate-rewrite-db=главный->подчиненный]
[--replicate-wild-do-table=шаблон]
[--replicate-wild-ignore-table=шаблон]
[--safe-mode]
[--server-id=идентификатор]
[--set-variable переменная=значение | -O переменная=значение]
[--skip-bdb]
[--skip-concurrent-insert]
[--skip-delay-key-write]
[--skip-gemini]
[--skip-grant-tables | -Sg]
[--skip-host-cache]
[--skip-innodb]
[--skip-locking]
[--skip-name-resolve]
[--skip-networking]
[--skip-new]
[--skip-show-database]
[--skip-slave-start]
[--skip-thread-priority]
```

```
--socket=файл]
--sql-bin-update-same]
--standalone]
--temp-pool]
--tmpdir=каталог | -t каталог]
--transaction-isolation=уровень]
--user=имя | -и имя]
--version | -V]
```

При работе в UNIX нужно пользоваться сценарием `safe_mysqld`, а не вызывать демон напрямую. Пользователи Windows NT и Windows 2000 должны установить демон в виде сервиса с помощью опции `--install`.

Большинство перечисленных ниже опций может присутствовать в конфигурационных файлах, о которых рассказывалось в начале главы. В этом случае перед именем опции не ставятся дефисы. Например, запись `--user=leon` в командной строке эквивалентна записи user=leon в группе `[mysqld]` файла `/etc/my.cnf`.

--ansi

Эта опция включает режим ANSI, в котором синтаксис SQL-инструкций немного меняется. В частности, между именами функций и открывающими скобками разрешаются пробелы. Это означает, что все имена функций становятся зарезервированными словами и их нельзя использовать в качестве имен таблиц или столбцов.

Тип столбца REAL преобразуется в тип FLOAT, а не DOUBLE. Для транзакций устанавливается уровень изоляции SERIALIZABLE. Оператор I I становится оператором конкатенации строк. Двойные кавычки запрещается использовать для выделения строковых литералов. Они теперь применяются вместо обратных кавычек для выделения имен баз данных, таблиц истолбцов.

Все эти изменения делают программу MySQL более совместимой со стандартом ANSI.

--basedir=каталог (-b каталог)

Эта опция задает основной каталог дистрибутива.

--bdb-home=каталог

Эта опция задает начальный каталог для файлов Berkeley DB.

--bdb-lock-detect=тип

Эта опция задает правило снятия взаимоблокировок с таблиц Berkeley DB. Аргумент *тип* может принимать одно из четырех значений: DEFAULT, OLDEST, RANDOM и YOUNGEST. Первое значение определяет стандартное правило, применяемое в библиотеке клиентских функций. В остальных случаях отменяется соответственно самая старая транзакция, произвольная транзакция или самая недавняя транзакция.

--bdb-logdir=каталог

Эта опция задает каталог для журнальных файлов Berkeley DB.

--bdb-no-recover

Эта опция подавляет восстановление таблиц Berkeley DB при запуске сервера.

--bdb-no-sync

Эта опция отключает режим синхронной очистки журнальных файлов Berkeley DB.

--bdb-shared-data

Эта опция разрешает многозадачный режим обслуживания таблиц Berkeley DB.

--bdb-tmpdir=каталог

Эта опция задает каталог для временных файлов модуля Berkeley DB.

--big-tables

Эта опция заставляет демон записывать временные таблицы на диск, а не хранить их в памяти. Программа MySQL самостоятельно определяет, когда следует "сбрасывать" таблицы на диск, поэтому данная опция считается устаревшей.

--bind-address=адрес

Эта опция задает IP-адрес сервера.

--binlog-do-db=база_данных

Эта опция заставляет сервер фиксировать обновления указанной базы данных только в двоичном журнале.

--binlog-ignore-db=база_данных

Эта опция заставляет сервер отключить для указанной базы данных двоичный журнал.

--bootstrap

Эта опция используется только инсталляционными сценариями.

--character-sets-dir=каталог

Эта опция задает каталог, в котором хранятся файлы наборов символов.

--chroot=каталог

Эта опция задает каталог, который после запуска сервера станет корневым.

--console

Эта опция заставляет демон держать консольное окно открытым (применима только в Windows-версиях демона).

--core-file

Эта опция позволяет демону создавать файл дампа в случае аварийного завершения.

--datadir=каталог (-h каталог)

Эта опция задает каталог, в котором будут храниться табличные данные. Для каждой базы данных создается отдельный подкаталог, содержащий файлы таблиц.

--default-character-set=имя

Эта опция задает стандартный набор символов.

--default-table-type=тип

Эта опция задает стандартный тип таблиц. Обычно по умолчанию принят тип `MYISAM`, но можно также задать тип ISAM, HEAP, `BDB`, Gemini и `InnoDB`. Все они описаны в главе 24, "Физическое хранение данных". При создании таблицы всегда можно явно указать ее тип.

--delay-key-write-for-all-tables

Эта опция запрещает очищать индексный буфер между операциями записи в таблицу.

--enable-locking

Эта опция включает режим блокирования файлов с использованием системной функции `lockd()`. В некоторых операционных системах, например в Linux, применять данную функцию не рекомендуется.

--exit-info=[битовое_поле] (-T [битовое_поле])

Эта опция переводит сервер в режим отладки. С помощью битового поля включаются различные опции, известные лишь разработчикам MySQL, поэтому применять данную опцию рядовому пользователю не рекомендуется.

--flush

Эта опция заставляет демон записывать изменения на диск после каждого запроса. Такой режим может оказаться полезен, если сервер часто "сбоит".

--gemini-full-recovery

Эта опция активизирует журнал восстановления таблиц Gemini (включена по умолчанию).

--gemini-lazy-commit

Эта опция ослабляет требование, в соответствии с которым журнал транзакций очищается после каждой инструкции COMMIT.

--gemini-no-recovery

Эта опция отключает журнал восстановления таблиц Gemini.

--gemini-unbuffered-io

Эта опция заставляет демон осуществлять запись в таблицы Gemini в обход системного буфера ввода-вывода.

--help (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--init-file=файл

Эта опция задает путь к файлу, содержащему SQL-инструкции. Данный файл выполняется при запуске сервера.

--innodb_data_file_path=файл:размер[;файл:размер]

Эта опция задает имена и размеры файлов данных, используемых таблицами InnoDB. Путь указывается относительно начального каталога InnoDB. Размер обычно выражается в мегабайтах (суффикс `M`). Например, запись `ibdata1:300M` указывает на файл размером 300 Мбайт. Спецификации файлов отделяются друг от друга символом точки с запятой.

--innodb_data_home_dir=каталог

Эта опция задает каталог для всех файлов InnoDB.

--innodb_flush_log_at_trx_commit={0 | 1}

Если эта опция равна нулю, то при завершении транзакции, связанной с изменением таблиц InnoDB, данные не будут немедленно записаны на диск.

--innodb_log_archive[=число]

Эта опция задает размер архива журнальных файлов InnoDB. Поскольку программа MySQL ведет свои собственные журналы обновлений, архивировать журналы InnoDB нет необходимости.

--innodb_log_arch_dir=каталог

Эта опция задает каталог, в котором будут храниться архивы журнальных файлов InnoDB.

--innodb_log_group_home_dir=каталог

Эта опция задает каталог для хранения журнальных файлов InnoDB.

--install

При наличии этой опции демон `mysqld` будет инсталлирован в виде сервиса в Windows NT и Windows 2000. В других операционных системах данная опция недоступна.

Именем сервиса будет MySQL. Сервис можно запускать и останавливать с помощью команды NET или команд панели управления сервисами.

--language=каталог (-L каталог)

Эта опция задает язык сообщений об ошибках (точнее, каталог, где хранятся сообщения на соответствующем языке). Файлы сообщений компилируются в файлы с расширением `.sys` с помощью утилиты `comp_err`.

--log[=каталог] (-l [каталог])

Эта опция включает регистрацию всех запросов на подключение к серверу и всех SQL-запросов. По умолчанию журнальный файл будет создан в каталоге данных и назван по имени узла с добавлением расширения `.` log, например `myserver.`log. Такой файл называется журналом запросов. О различных форматах журнальных файлов рассказывается в главе 24, "Физическое хранение данных".

--log-bin-index=каталог

Эта опция задает имя индексного файла для двоичного журнала. По умолчанию файл назван по имени узла с добавлением суффикса `-bin.`index, например `myserver-bin.index`.

--log-bin [=каталог]

Эта опция включает двоичный журнал, который используется при репликации. По умолчанию журнальный файл назван по имени узла с добавлением суффикса -bin, а в качестве расширения используется номер журнала, например `myserver-bin.001`.

--log-isam[=каталог]

Эта опция включает регистрацию всех действий, выполняемых над таблицами MyISAM. Данный режим применяется разработчиками MySQL при отладке программы и не представляет интереса для рядовых пользователей. По умолчанию создается журнальный файл `myisam.log`.

--log-long-format

При наличии этой опции в журнал обновлений будет записываться дополнительная информация. Кроме того, запросы, в которых не используются индексы, будут регистрироваться в журнале медленных запросов, если таковой ведется.

--log-slave-updates

Эта опция заставляет подчиненный сервер регистрировать обновления, получаемые от главного сервера, в двоичном журнале. По умолчанию данный режим отключен. Если подчиненный сервер является главным для другого сервера в цепочке, эту опцию нельзя использовать.

--log-slow-queries[=каталог]

Эта опция включает регистрацию медленных запросов, т.е. таких, которые выполняются дольше определенного времени. Соответствующий лимит задан в переменной long_query_time.

--log-update[=каталог]

Эта опция включает регистрацию обновлений. По умолчанию журнальный файл назван по имени узла, а в качестве расширения используется номер журнала, например `myserver.001`.

--low-priority-updates

При наличии этой опции и нструкции DELETE, INSERT и UPDATE будут иметь более низкий приоритет, чем инструкции SELECT.

--master-connect-retry=секунды

Эта опция задает интервал времени, в течение которого можно пытаться повторно подключиться к главному серверу (по умолчанию — 60 с).

--master-host=узел

Эта опция задает доменное имя или IP-адрес главного сервера.

--master-info-file=файл

Эта опция переопределяет стандартное имя файла, используемого подчиненным сервером для отслеживания своей позиции в журнальном файле главного сервера. По умолчанию файл называется `master.info`.

--master-password=пароль

Эта опция задает пароль, с помощью которого подчиненный сервер регистрируется на главном сервере.

--master-port=порт

Эта опция задает порт, прослушиваемый главным сервером в ожидании запросов на подключение (по умолчанию — 3306).

--master-user=пользователь

Эта опция задает имя пользователя, указываемое подчиненным сервером при регистрации на главном сервере.

--memlock

Эта опция заставляет операционную систему хранить исполняемый файл сервера в физической памяти и не выгружать его страницы на диск. Такой режим допустим лишь в техсистемах, где используется функция `mlockall()`.

--myisam-recover[=опция[,опция...]]

При наличии этой опции демон будет пытаться восстанавливать поврежденные таблицы в момент запуска сервера. Необходимо, чтобы была также установлена опция --skip-locking. Демон просканирует все неправильно закрытые или поврежденные таблицы, а затем выполнит действие, определяемое указанными опциями (табл. 14.2).

Опция	*Директива*
BACKUP	Создать резервную копию файла перед его восстановлением
DEFAULT	Ничего не делать
FORCE	Восстанавливать таблицу даже в том случае, если некоторые записи будут потеряны
QUICK	Не проверять строки таблиц, если отсутствуют удаленные блоки

--new (-n)

Эта опция предназначена для активизации новых, возможно небезопасных, функций.

--old-protocol (-o)

Эта опция заставляет сервер придерживаться старой версии клиентского протокола. Данное изменение затрагивает лишь клиентов более ранних версий, чем 3.20.28.

--pid-file=файл

Эта опция задает путевое имя файла, в котором хранится идентификатор процесса. Зная данный идентификатор, можно уничтожить серверный процесс.

--port=порт (-P порт)

Эта опция задает порт, прослушиваемый сервером в ожидании запросов на подключение (по умолчанию —3306).

--remove

Эта опция означает удаление сервиса MySQL из Windows. В других операционных системах данная опция недоступна.

--replicate-do-db=база_данных

Эта опция заставляет подчиненный сервер реплицировать только указанную базу данных. Если необходимо реплицировать группу баз данных, укажите несколько таких опций.

--replicate-do-table=база_данных.таблица

Эта опция заставляет подчиненный сервер реплицировать только указанную таблицу. Если необходимо реплицировать группу таблиц, укажите несколько таких опций. **Таблицы,** которые не были упомянуты, не реплицируются.

--replicate-ignore-db=база_данных

Эта опция запрещает серверу реплицировать указанную базу данных.

--replicate-ignore-table=база_данных.таблица

Эта опция запрещает серверу реплицировать указанную таблицу.

--replicate-rewrite-db=главный->подчиненный

Эта опция создает правило, по которому имя базы данных главного сервера транслируется в имя базы данных подчиненного сервера. Например, правило `replicate-rewrite-db=freetrade->store` означает, что база `freetrade` превратится в базу `store`.

--replicate-wild-do-table=шаблон

В этой опции с помощью метасимволов задаются имена таблиц, реплицируемых подчиненным сервером. Имени таблицы должно предшествовать имя *базы* данных, например `store.cat%`.

--replicate-wild-ignore-table=шаблон

В этой опции с помощью метасимволов задаются имена таблиц, которые не должны реплицироваться подчиненным сервером. Имени таблицы должно предшествовать имя базы данных, например `store.cat%`.

--safe-mode

Эта опция отключает некоторые функции оптимизации.

--server-id=идентификатор

Эта опция задает уникальный идентификатор сервера, требуемый для репликации.

--set-variable переменная=значение (-O переменная=значение)

Эта опция задает значение указанной серверной переменной. Список переменных приведен в табл. 14.3.

Переменная	*Описание*
back_log	Максимальное число не обслуженных запросов на подключение; по достижении этого предела сервер перестанет реагировать на запросы
bdb_cache_size	Размер кэша для таблиц Berkeley DB
bdb_log_buffer_size	Размер буфера журнальной регистрации для таблиц Berkeley DB
bdb_max_lock	Максимальное число блокировок для таблиц Berkeley DB
binlog_cache_size	Размер буфера для двоичного журнала
connect_timeout	Время простоя, по истечении которого запрос на подключение будет отменен
delayed_insert_limit	Число строк, которые могут быть записаны инструкцией INSERT DELAYED, прежде чем снова будут разрешены операции чтения
delayed_insert_timeout	Время, отводимое на выполнение инструкции INSERT DELAYED
delayed_queue_size	Число строк, поставленных в очередь на отложенную запись
flush_time	Интервал времени между операциями закрытия таблиц
gemini_connection_limit	Максимальное число пользователей, которые могут одновременно работать с таблицами Gemini
gemini_db_buffers	Число буферов таблиц Gemini, хранимых в кэше
gemini_io_threads	Число потоков ввода-вывода, связанных с таблицами Gemini
gemini_lock_table_size	Максимальное число блокировок таблиц Gemini (по умолчанию — 4096)
gemini_spin_retries	Максимальное число попыток повторно поставить блокировку на таблицу Gemini

Переменная	*Описание*
innodb_additional_mem_pool_size	Размер резидентного буфера, используемого модулем InnoDB для хранения внутренних структур данных
innodb_buffer_pool_size	Размер буфера, используемого модулем InnoDB для кэширования таблиц и индексов
innodb_file_io_threads	Число потоков InnoDB, осуществляющих операции файлового ввода-вывода
innodb_lock_wait_timeout	Интервал времени, в течение которого транзакция InnoDB ожидает снятия взаимоблокировки; по прошествии этого времени транзакция будет отменена
innodb_log_buffer_size	Размер буфера, используемого для записи журнальных файлов InnoDB
innodb_log_files_in_group.	Число журнальных файлов InnoDB, участвующих в цикле ротации
innodb_log_file_size	Размер журнальных файлов InnoDB
innodb_mirrored_log_groups	Число идентичных групп журнальных файлов
interactive_timeout	Допустимое время простоя в интерактивном сеансе
join_buffer_size	Размер буфера для операций объединения, в которых не используются индексы
key_buffer_size	Размер индексного буфера, используемого всеми соединениями
long_query_time	Число секунд, по истечении которого выполняющийся запрос считается медленным
lower_case_table_names	Будет равна 1, если имена всех таблиц принудительно переводятся в нижний регистр
max_allowed_packet	Максимальный размер пакета в байтах

Переменная	*Описание*
max_binlog_cache_size	Максимальный размер двоичного журналатранзакций
max_binlog_size	Предельный размер в байтах, по достижении которого произойдет ротация двоичных журнальных файлов
max_connections	Максимальное число соединений
max_connect_errors	Максимальное число ошибок подключения, по достижении которого доступ кузлублокируется
max_delayed_threads	Максимальное число потоков, в которых выполняются инструкции INSERT DELAYED
max_heap_table_size	Максимальный размер таблиц типа HEAP
max_join_size	Максимальное число строк, которое может быть получено в результате объединения таблиц
max_sort_length	Максимальное число байтов, используемых при сортировке столбцов типа BLOB и TEXT
max_tmp_tables	Максимальное число временных таблиц
max_user_connections	Максимальное число соединений для одного пользователя
max_write_lock_count	Число блокировок записи, после которого необходимо разрешить блокировки чтения
myisam_max_extra_sort_file_size	Максимальная разница в размерах временного файла, используемого для создания индекса, и индексного буфера для таблиц MyISAM; считается, что размеры таблиц заданы в мегабайтах
myisam_max_sort_file_size	Максимальный размер (в мегабайтах) временного файла, ис пользуемого для создания индекса таблицы MylSAM; в случае превышения этого предела будет использован индексный буфер

Переменная	*Описание*
myisam_sort_buffer_size	Размер буфера, используемого для создания индексов таблиц MyISAM
net_buffer_length	Ожидаемая длина запросов, посылаемых клиентами
net_read_timeout	Предельное время ожидания следующих данных в ходе операции чтения
net_retry_count	Число попыток восстановить прерванную операцию чтения
net_write_timeout	Предельное время ожидания записи требуемого блока данных
open_files_limit	Максимальное число файловых дескрипторов, используемых демоном mysqld
query_buffer_size	Начальный размер буфера запросов
record_buffer	Размер буфера, используемого при сканировании таблиц
slow_launch_time	Число секунд, по истечении которого запуск потока будет считаться медленным
sort_buffer	Размер буфера, используемого для сортировки записей
table_cache	Максимальное число открытых таблиц
thread_cache_size	Число потоков, хранимых в кэше для повторного использования
thread_concurrency	Указание на число одновременных потоков (только в Solaris)
thread_stack	Размер стека потоков
tmp_table_size	Максимальный размер временной таблицы, по достижении которого она записывается на диск
wait_timeout	Период простоя, по истечении которого соединение закрывается

--skip-bdb

Эта опция отключает поддержку таблиц Berkeley DB. Если использование таких таблиц не планируется, включите данную опцию, чтобы не расходовались лишние ресурсы.

--skip-concurrent-insert

Эта опция запрещает выполнять одновременные операции вставки данных в таблицы MyISAM. Она предназначена для целей отладки.

--skip-delay-key-write

При наличии этой опции запросы на отложенную запись ключей будут игнорироваться.

--skip-gemini

Эта опция отключает поддержку таблиц Gemini.

--skip-grant-tables (-Sg)

Эта опция означает запуск сервера в режиме, при котором всем пользователям предоставлены максимальные привилегии. Данный режим помогает администратору, забывшему свой пароль, вернуть контроль над сервером. Привилегии восстанавливаются с помощью команды `mysqladmin reload` или инструкции FLUSH PRIVILEGES.

--skip-host-cache

Эта инструкция отключает кэш доменных имен. Для поиска доменных имен будут посылаться запросы серверу DNS.

--skip-innodb

Эта опция отключает поддержку таблиц InnoDB. Если использование таких таблиц не планируется, включите данную опцию, чтобы не расходовались лишние ресурсы.

--skip-locking

Эта опция отключает системный режим блокирования таблиц. Она необходима в некоторых операционных системах.

--skip-name-resolve

Эта опция отключает режим преобразования IP-адресов в доменные имена.

--skip-networking

Эта опция запрещает соединения TCP/IP, разрешая лишь соединения, устанавливаемые посредством UNIX-сокетов.

--skip-new

Эта опция предназначена для отключения новых, возможно **нестабильных,** функций.

--skip-show-database

Эта опция говорит о том, что для выполнения инструкции `SHOW` DATABASES пользователи должны иметь привилегию PROCESS.

--skip-slave-start

Эта опция запрещает подчиненному серверу начинать процесс репликации в момент своего запуска. Репликацию можно инициировать вручную с помощью инструкции SLAVE START.

--skip-thread-priority

Эта функция отменяет приоритеты потоков.

--socket=файл

Эта функция задает путь к **сокету,** с помощью которого **устанавливаются** локальные соединения (по умолчанию — `/tmp/mysql.sock`).

--sql-bin-update-same

Эта опция заставляет главный сервер использовать для двоичного журнального файла и журнала обновлений одно и то же имя.

--standalone

Эта опция поддерживается только в Windows-версиях MySQL. Если она присутствует, демон не будет запускаться в виде сервиса.

--temp-pool

Эта опция запрещает демону назначать каждому новому временному файлу уникальное имя. Вместо этого будет создан небольшой пул имен. Данный прием позволяет избежать ошибки, присущей ядрам Linux вплоть до версии 2.4.3 (они неправильно выделяют память при создании группы файлов с разными именами).

--tmpdir=каталог (-t каталог)

Эта опция задает каталог для временных файлов.

--transaction-isolation=уровень

Эта опция задает стандартный уровень изоляции транзакций. Значение аргумента может быть таким: READ-COMMITTED, READ-UNCOMMITTED, `REPEATABLE-READ` и `SERIALIZABLE` (см. описание инструкции SET TRANSACTION в главе 13, "Инструкции SQL").

--user=имя (-u имя)

Эта опция задает имя пользователя, запускающего демон `mysqld`.

--version (-V)

При наличии этой опции возвращается лишь информация о версии утилиты.

mysqld-max

Это версия сервера MySQL, скомпилированная с включением всех возможных опций. Функционально она идентична демону mysqld.

mysqld-nt

Это версия сервера MySQL, скомпилированная для систем Windows NT и Windows 2000. В нее включена поддержка именованных каналов. Если при вызове программы не указано никаких опций, демон пытается запуститься в виде сервиса.

mysqld-opt

Это версия сервера MySQL, оптимизированная для процессоров Pentium. Ее рекомендуется применять в Windows 95 и Windows 98.

mysqld_multi

Эта утилита позволяет запускать несколько серверов MySQL одновременно. Синтаксис ее вызова таков:

```
mysqld_multi
     [--config-file=файл]
     [--example]
     [--help]
     [--log=файл]
      --mysqladmin=файл]
      --mysqld=файл]
      --no-log]
      --password=пароль]
      --tcp-ip]
      --user=имя]
      --version]
     {start|stop|report}    [группа[-группа]][,группа[-группа]]...
```

Утилита запускает несколько копий демона mysqld, каждую со своими конфигурационными параметрами. Наборы параметров **группируются** под специальными заголовками конфигурационного файла. Обычно установки сервера нах одятся в разделе `[mysqld]`. Если же запускается несколько серверов, то каждой группе параметров присваивается целочисленный **номер**, например `[mysqld15]`.

В листинге 14.12 показана конфигурация двух серверов. Первый из них хранит данные в подкаталоге var2 и прослушивает порт 3307. Второй сервер работает с ка-

талогом var3 и прослушивает порт 3308. В каждой из групп могут присутствовать любые параметры демона mysqld.

```
[mysqld_multi]
mysqld     = /usr/local/libexec/mysqld
mysqladmin = /usr/local/bin/mysqladmin
user       = multi_admin
password   = multipass
log        = /usr/local/var/multi.log

[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/var2/myserver.pid2
datadir    = /usr/local/mysql/var2
user       = jsmith

[mysqld3]
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/var3/myserver.pid3
datadir    = /usr/local/mysql/var3
user       = tjones
```

У каждого сервера должны быть свои сокет, порт, файл, хранящий идентификатор процесса, и каталог данных. Подробнее о запуске нескольких серверов рассказывается в главе 29, "Распределенные базы данных".

Утилита mysqld_multi способна запускать (директива start), останавливать (директива stop) и описывать (директива report) серверы, определяемые номером группы. Номера или их диапазоны разделяются запятыми. Пробелы в списке номеров недопустимы.

--config-file=файл

Эта опция задает путь к конфигурационному файлу. Формат стандартного файла был описан в начале главы. В UNIX он обычно называется /etc/my.cnf.

--example

При наличии этой опции утилита выдает лишь образец конфигурационного файла.

--help

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--log=файл

Эта опция задает путь к журнальному файлу (по умолчанию — /tmp/mysqld_multi.log).

--mysqladmin=файл

Эта опция задает путь к утилите `mysqladmin`, которая вызывается при выгрузке серверов.

--mysqld=файл

Эта опция задает путь к демону `mysqld`, который вызывается при запуске серверов. К демону можно обращаться непосредственно или с помощью сценария `safe_mysqld`.

--no-log

Эта опция отменяет журнальную регистрацию.

--password=пароль

Эта опция задает пароль, указываемый при запуске утилиты mysqladmin.

--tcp-ip

При наличии этой опции утилита будет взаимодействовать с сервером не через сокеты, а по протоколам TCP/IP.

--user=имя

Эта опция задает имя пользователя, указываемое при запуске утилиты mysqladmin. Учтите, что для всех серверов, упомянутых при вызове утилиты mysqld_multi, должны использоваться одинаковые имя пользователя и пароль. Поскольку у каждого сервера своя база данных `mysql`, нужно создать в ней учетную запись этого пользователя и предоставить ему привилегию на останов сервера.

--version

При наличии этойопции возвращается лишь информация о версии утилиты.

mysqldump

Утилита mysqldump извлекает информацию из указанной базы данных. Синтаксис ее вызова таков:

```
mysqldump
     [--add-drop-table]
     [--add-locks]
     [--all | -a]
     [--all-databases | -A]
     [--allow-keywords]
     [--character-sets-dir=каталог]
     [--complete-insert | -c]
     [--compress | -C]
     [--databases | -B]
     [--debug=[конфигурация] \ -# конфигурация]
     [--default-character-set=имя]
```

```
[--delayed-insert]
[--extended-insert | -e]
[--fields-enclosed-by=символ]
[--fields-escaped-by=символ]
[--fields-optionally-enclosed-by=символ]
[--fields-terminated-by=разделитель]
[--flush-logs I -F]
[-- force | -f]
[--help I -?]
[--host=узел | -h узел]
[--lines-terminated-by=разделитель]
[--lock-tables I -l]
[--no-create-db I -n]
[--no-create-info | -t]
[--no-data | -d]
[--opt]
[--password[=пароль] \ -p[пароль]]
[--port=порт | -P port]
[--quick |·-q]
[--quote-names]
[--set-variable переменная=значение \ -O переменная=значение]
[--socket=файл I -S файл]
[--tab=файл | -T файл]
[--tables]
[--user=имя | -u имя]
[--verbose   -v]
[--version   -V]
[--where=условия | -w условия]
база_данных [таблица]...
```

Эта утилита формирует SQL-инструкции, предназначенные для воссоздания указанных таблиц в другой базе данных. Полученные инструкции записываются в поток `stdout`. **Как минимум, это будут инструкции** CREATE TABLE и INSERT.

В стандартном режиме утилита `mysqldump` **принимает имя одной базы данных и необязательный список ее таблиц. Если ни одна таблица не указана, будет воссоздана вся база данных. С помощью опции** `--databases` **можно задать список баз данных, но тогда список таблиц будет игнорироваться.**

В листинге 14.13 демонстрируется воссоздание таблицы `db` **базы данных** `mysql`. **Результаты работы утилиты** mysqldump **записываются в файл** `dump.sql`. **В главе 25, "Устранение последствий катастроф", речь пойдет о применении утилиты** mysqldump **и других методах резервного копирования баз данных.**

```
[/tmp]# mysqldump --opt mysql db > dump.sql
[/tmp]# cat dump.sql
# MySQL dump 8.13
#
# Host: localhost Database: mysql
#
# Server version 3.23.39-log

#
```

```
# Table structure for table 'db'
#

DROP TABLE IF EXISTS db;
CREATE TABLE db (
  Host char(60) binary NOT NULL default '',
  Db char(64) binary NOT NULL default '',
  User char(16) binary NOT NULL default '',
  Select_priv enum('N','Y') NOT NULL default 'N',
  Insert_priv enum('N','Y') NOT NULL default 'N',
  Update_priv enum('N','Y') NOT NULL default 'N',
  Delete_priv enum('N','Y') NOT NULL default 'N',
  Create_priv enum('N','Y') NOT NULL default 'N',
  Drop_priv enum('N','Y') NOT NULL default 'N',
  Grant_priv enum('N', 'Y') NOT NULL default 'N',
  References_priv enum('N','Y') NOT NULL default 'N',
  Index_priv enum('N','Y') NOT NULL default 'N',
  Alter_priv enum('N','Y') NOT NULL default 'N',
  PRIMARY KEY (Host,Db,User),
  KEY User (User)
) TYPE=MyISAM COMMENT='Database privileges';

tt
# Dumping data for table 'db'
tt

LOCK TABLES db WRITE;
INSERT INTO db VALUES
('%','test','','Y','Y','Y','Y','Y','Y','N','Y','Y','Y'),('%',
'test\\_%','','Y','Y','Y','Y','Y','Y','N','Y','Y','Y'),(
'localhost','freetime','httpd','Y','Y','Y','Y','N','N','N','N',
'N','N');
UNLOCK TABLES;
```

--add-drop-table

Когда присутствует эта опция, перед каждой инструкцией CREATE TABLE будет вставляться инструкция DROP TABLE IF EXISTS.

--add-locks

Эта опция заставляет утилиту блокировать таблицы для записи перед вставкой строк. Перед каждой группой инструкций INSERT будет стоять инструкция LOCK TABLES, а после группы — инструкция UNLOCK TABLES.

--all (-a)

Эта опция требует от утилиты придерживаться синтаксиса SQL-инструкций, специфичного для MySQL. В результате полученный файл сценария может не поддерживаться другими серверами.

--all-databases (-A)

При наличии этой опции будут воссозданы все базы данных. Указывать списки баз данных и таблиц нет необходимости, так как в сценарий включаются соответствующие инструкции CREATE DATABASE.

--allow-keywords

При наличии этой опции имена столбцов, совпадающие с ключевыми словами, будут сопровождаться префиксом, соответствующим имени таблицы.

--character-sets-dir=каталог

Эта опция задает каталог, в котором хранятся файлы наборов символов.

--complete-insert (-c)

Эта опция заставляет утилиту включать в инструкции `INSERT` списки столбцов.

--compress (-C)

При наличии этой опции утилита будет сжимать данные, передаваемые от клиента к серверу.

--databases (-B)

Эта опция позволяет указывать список воссоздаваемых баз данных. Все аргументы, стоящие после списка опций, считаются именами баз данных, а не таблиц. Как и в случае опции `--all-databases`, в сценарий включаются инструкции CREATE DATABASE.

--debug=[конфигурация] (-#конфигурация)

Эта опция заставляет утилиту записывать отладочную информацию в указанный журнальный файл. Поддержка данной опции должна быть включена в утилиту на этапе компиляции. По умолчанию данные записываются в файл `/tmp/client.trace`. Подробнее о формате журнальных файлов рассказывается в главе 24, "Физическое хранение данных".

--default-character-set=имя

Эта опция меняет стандартный набор символов, установленный на этапе компиляции программы MySQL. Обычно таковым является набор Latin (ISO8859-1). Указываемый файл должен находиться в каталоге, который был задан с помощью опции `--character-sets-dir`. Этот файл представляет собой не просто коллекцию символов. В нем описаны правила сортировки алфавита и трансляции каждого символа в верхний и нижний регистры.

О создании наборов символов рассказывается в главе 31, "Расширение возможностей MySQL".

--delayed-insert

Эта опция заставляет утилиту добавлять к инструкциям INSERT ключевое слово DELAYED.

--extended-insert (-e)

Эта опция разрешает добавлять по несколько записей за раз.

--fields-enclosed-by=символ

Эта опция используется совместно с опцией `--tab` и задает символ, применяемый для выделения полей.

--fields-escaped-by=символ

Эта опция используется совместно с опцией `--tab` и задает символ, применяемый для отмены специального назначения управляющих символов.

--fields-optionally-enclosed-by=символ

Эта опция используется совместно с опцией `--tab` и задает символ, применяемый для выделения строковых полей.

--fields-terminated-by=разделитель

Эта опция используется совместно с опцией `--tab` и задает символ-разделитель полей.

--flush-logs (-F)

Эта опция заставляет утилиту очищать журнальные файлы перед тем, как начинать процесс генерации таблиц.

-force (-f)

Эта опция заставляет утилиту продолжать процесс генерации таблиц даже в случае возникновения ошибки.

-help (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--host=узел (-h узел)

Эта опция задает узел, к которому будет пытаться подключиться утилита. По умолчанию связь устанавливается с локальным узлом. В качестве аргумента может присутствовать доменное имя или IP-адрес.

--lines-terminated-by=разделитель

Эта опция используется совместно с опцией `--tab` и задает символ-разделитель строк.

--lock-tables (-l)

Эта опция заставляет утилиту блокировать все таблицы для чтения перед тем, как начинать процесс генерации таблиц. При выполнении инструкции LOCK TABLES указывается ключевое слово LOCAL, что позволяет выполнять одновременные операции вставки в таблицы MyISAM.

--no-create-db (-n)

Эта опция запрещает создавать инструкции CREATE DATABASE, даже если присутствует опция `--all-databases` или `--databases`.

--no-create-info (-t)

Эта опция запрещает создавать инструкции CREATE TABLE. Разрешаются лишь инструкции INSERT.

--no-data (-d)

Эта опция запрещает создавать инструкции INSERT. Разрешаются лишь инструкции CREATE TABLE.

--opt

Эта опция обозначает режим оптимальных установок. Активизируются следующие опции: `--quick`, `--add-drop-table`, `--add-locks`, `--extended-insert`, `--lock-tables`.

--password[=пароль] (-p[пароль])

Эта опция задает пароль для подключения к серверу. При ее отсутствии пароль не посылается. Если же пароль не указан, программа попросит его ввести. Учтите, что пароль, вводимый в командной строке, могут увидеть другие пользователи, просматривающие список процессов. Обратите также внимание на то, что между названием короткой версии опции и паролем не нужен пробел.

--port=порт (-P port)

Эта опция переопределяет стандартный номер порта, с которым работает утилита (по умолчанию — 3306).

--quick (-q)

Эта опция заставляет утилиту не кэшировать результаты своей работы, а отображать их строка за строкой.

--quote-names

При наличии этой опции имена таблиц и столбцов будут заключаться в обратные кавычки.

--set-variable переменная=значение (-O переменная=значение)

Эта опция позволяет установить значения переменных `max_allowed_packet` (определяет максимальный размер пакета) и `net_buffer_length` (определяет размер сетевого буфера).

--socket=файл (-S файл)

Эта опция переопределяет стандартный сокет, используемый для подключения к серверу. Обычно файл сокета называется `/tmp/mysql.sock`.

--tab=файл (-T файл)

При наличии этой опции утилита будет создавать для каждой таблицы два файла. Первый файл называется так же, как и таблица, и имеет расширение `.sql`. В нем содержится инструкция CREATE TABLE.

Второй файл называется аналогично, но имеет расширение `.txt`. В нем содержатся записи таблицы с символами табуляции в качестве разделителей полей. Записи разделяются символами новой строки. Именно такой файл создается инструкцией SELECT INTO. С помощью опций `--fields-enclosed-by`, `--fields-escaped-by`, `--fields-optionally-enclosed-by`, `--fields-terminated-by` и `--lines-terminated-by` можно изменить формат файла данных.

--tables

Эта опция отменяет опцию `--databases`.

--user=имя (-u имя)

Эта опция задает имя пользователя, указываемое при регистрации на сервере.

--verbose (-v)

Эта опция заставляет утилиту сопровождать свои действия комментариями.

--version (-V)

При наличии этой опции будет выдана лишь информация о версии утилиты.

--where=условия (-w условия)

Эта опция заставляет утилиту применять условия отбора к записям, включаемым в инструкцию INSERT. Не забудьте взять всю строку опции в двойные кавычки, чтобы избежать проблем с анализом аргументов командной строки.

В листинге 14.14 с помощью опции `--where` **из таблицы** `db` **отбираются только те записи,** которые относятся **к** пользователю `httpd`.

```
mysqldump "--where=user='httpd'" mysql db
```

mysqldumpslow

Этот **Perl-сценарий** отображает **на экране журнал медленных** запросов. **Синтаксис** его вызова **таков:**

```
mysqldumpslow
    [-a]
    [-d]
    [-g выражение]
    [-h узел]
    [-i сервер]
     -l]
     -n количество_цифр]
     -r]
     -s тип]
     -t число_запросов]
     -v]
    [журнальный_файл]
```

Если не указать путь к журнальному **файлу, сценарий** попытается самостоятельно **найти** его **на** основании установок конфигурационного **файла. Будут выведены дампы всех найденных** журналов данного **типа.**

-a

Эта опция запрещает группировать запросы по аргументам предложения WHERE. Обычно два запроса, отличающихся лишь литералами в предложении WHERE, объединяются водин.

-d

Эта опция **включает режим** отладки.

-g выражение

При наличии этой опции **будут учитываться лишь те инструкции,** которые соответствуют заданному регулярному **выражению.**

-h узел

Эта опция **задает имя узла, на** основании которого **будут** отбираться **журнальные файлы. В имени узла** могут присутствовать метасимволы.

-i сервер

Эта опция задает экземпляр сервера.

-l

Эта опция заставляет сценарий добавлять время блокировки к общему времени выполнения запроса.

-n количество_цифр

Эта опция заставляет сценарий считать одинаковыми все числа, в представлении которых содержится не менее указанного количества цифр.

-r

Эта опция изменяет порядок сортировки на обратный.

-s тип

Эта опция определяет, какого рода информацию сценарий будет искать в журнальном файле (табл. 14.4).

Тип	*Описание*
al	Средняя длительность блокировки
ar	Среднее число записей
at	Среднее время выполнения
l	Длительность блокировки
r	Число записей
t	Время выполнения

-t число_запросов

Эта опция задает число анализируемых записей журнального файла.

-V

Эта опция заставляет сценарий выдавать более подробные сообщения.

mysqlhotcopy

Этот Perl-сценарий создает копию активной базы данных. Синтаксис его вызова таков:

```
mysqlhotcopy
     [--allowold]
     [--checkpoint=таблица]
     [--debug]
     [--dryrun]
     [--flushlog]
     [--help | -?]
     [--keepold]
     [--method=имя]
     [--noindices]
     [--password=[пароль] | -p[пароль]]
     [--port=порт | -P порт]
     [--quiet | -q]
     [--regexp=шаблон]
     [--resetmaster]
     [--resetslave]
     [--socket=файл | -S файл]
     [--suffix=суффикс]
     [--user=имя | -u имя]
     база_данных[./[~]шаблон/] [каталог]
```

Сценарий блокирует все таблицы базы данных, а затем создает их образы в указанном каталоге. Можно отбирать таблицы, имена которых соответствуют или не соответствуют (метасимвол ~) регулярному выражению.

Этот сценарий требует наличия модуля DBI. Кроме того, его можно запускать только на сервере, т.е. **там,** где хранится база данных.

--allowold

Эта опция заставляет сценарий не удалять существующие резервные копии, а переименовывать их. В случае успешного завершения старые копии удаляются, иначе — восстанавливаются.

--checkpoint=таблица

Эта опция заставляет сценарий добавлять в контрольную таблицу запись о создании резервной копии каждой таблицы. Имени контрольной таблицы должно предшествовать имя базы данных, например `mylogs.checkpoint`. В этой таблице должны присутствовать как минимум следующие столбцы:

```
time_stamp TIMESTAMP NOT NULL
src VARCHAR(32)
dest VARCHAR(60)
msg VARCHAR(255)
```

--debug

При наличии этой опции будет включен режим отладки.

--dryrun

Эта опция заставляет сценарий сообщить о предполагаемых действиях, но не выполнять их.

--flushlog

При наличии этой опции сценарий выполнит инструкцию FLUSH LOGS после блокирования таблиц.

--help (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--keepold

Эта опция запрещает удалять существующие резервные копии по окончании работы.

--method=имя

Эта опция задает утилиту, используемую для копирования файлов. Аргумент *имя* может быть равен либо `cp`, либо `scp`.

--noindices

Эта опция запрещает копировать индексные файлы. Их можно восстановить с помощью утилиты `myisamchk`.

--password=[пароль] (-p[пароль])

Эта опция задает пароль для подключения к локальному серверу.

--port=порт (-P порт)

Эта опция задает номер порта для подключения к локальному серверу.

--quiet (-q)

Эта опция подавляет вывод на экран любых данных, кроме сообщений об ошибках.

--regexp=шаблон

Эта опция задает шаблон для выбора баз данных. Будет создана резервная копия каждой базы данных, имя которой соответствует шаблону.

--resetmaster

При наличии этой опции сценарий выполнит инструкцию RESET MASTER после блокирования таблиц.

--resetslave

При наличии этой опции сценарий выполнит инструкцию RESET SLAVE после блокирования таблиц.

--socket=файл (-S файл)

Эта опция задает путь к сокету, используемому для подключения к локальному серверу.

--suffix=суффикс

Эта опция задает суффикс имени архивного каталога (по умолчанию _copy).

--user=имя (-u имя)

Эта опция задает имя пользователя, указываемое при подключении к базе данных.

mysqlimport

Эта утилита представляет собой оболочку инструкции LOAD DATA INFILE. Синтаксис ее вызова таков:

```
mysqlimport
    [--character-sets-dir=каталог]
    [--columns=имя[,имя]... | -c имя[,имя]...]
    [--compress | -C]
    [--debug[=конфигурация] | -# конфигурация]
    [--delete | -d]
    [--fields-enclosed-by=символ]
    [--fields-escaped-by=символ]
    [--fields-optionally-enclosed-by=символ]
    [--fields-terminated-by=разделитель]
    [--force | -f]
    [--help | -?]
    [--host=узел | -h узел]
    [--ignore | -i]
    [--lines-terminated-by=разделитель]
    [--local | -L]
    [--lock-tables | -l]
    [--low-priority]
    [--password[=пароль] | -p[пароль]]
    [--port=порт | -P порт]
    [--replace | -r]
    [--silent | -s]
    [--socket=файл | -S файл]
    [--user=имя | -u имя]
    [--verbose -v]
    [--version -V]
    база_данных текстовый_файл [текстовый_файл]...
```

Утилита импортирует записи в таблицы. Необходимо указать базу данных и как минимум один текстовый файл. На основании имени файла утилита определяет имя таблицы, в которую импортируются данные.

--character-sets-dir=каталог

Эта опция задает каталог, в котором хранятся файлы наборов символов.

--columns=имя[,имя]... (-c имя[,имя]...)

Эта опция позволяет задать подмножество столбцов или переупорядочить их. Столбцам, имена которых отсутствуют в списке, будут присвоены значения по умолчанию. Строки импортируемого файла могут иметь слишком мало или слишком много полей, но утилита обычно справляется с этим. Пропущенным полям также присваиваются стандартные значения.

--compress (-C)

При наличии этой опции утилита будет сжимать данные, передаваемые от клиента к серверу.

--debug[=конфигурация] (-# конфигурация)

Эта опция заставляет утилиту записывать отладочную информацию в указанный журнальный файл. Поддержка данной опции должна быть включена в утилиту на этапе компиляции. По умолчанию данные записываются в файл `/tmp/client.trace`. Подробнее о формате журнальных файлов рассказывается в главе 24, "Физическое хранение данных".

--delete (-d)

При наличии этой опции таблица будет очищена перед импортом данных.

--fields-enclosed-by=символ

Эта опция задает символ, применяемый для выделения полей.

--fields-escaped-by=символ

Эта опция задает символ, применяемый для отмены специального назначения управляющих символов.

--fields-optionally-enclosed-by=символ

Эта опция задает символ, применяемый для выделения строковых полей.

--fields-terminated-by=разделитель

Эта опция задает символ-разделитель полей.

--force (-f)

Эта опция заставляет утилиту продолжать процесс заполнения таблиц даже в случае возникновения ошибки.

--help (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--host=узел (-h узел)

Эта опция задает узел, к которому попытается подключиться утилита. В качестве аргумента может присутствовать доменное имя или IP-адрес.

--ignore (-i)

Эта опция заставляет утилиту игнорировать входные записи, которые дублируют существующие записи.

--lines-terminated-by=разделитель

Эта опция задает символ-разделитель строк.

--local (-L)

Обычно текстовые файлы открываются в файловой системе сервера. При наличии данной опции файлы загружаются из клиентской системы.

--lock-tables (-l)

Эта опция заставляет утилиту блокировать все таблицы базы данных перед тем, как начинать процесс вставки записей.

--low-priority

Эта опция задерживает процедуру импорта до тех пор, пока не будут завершены все операции чтения таблиц.

--password[=пароль] (-p[пароль])

Эта опция задает пароль для подключения к серверу. При ее отсутствии пароль не посылается. Если же пароль не указан, программа попросит его ввести. Учтите, что пароль, вводимый в командной строке, могут увидеть другие пользователи, **просматривающие** список процессов. Обратите также внимание на то, что между названием короткой версии опции и паролем не нужен пробел.

--port=порт (-P порт)

Эта опция переопределяет стандартный номер порта, с которым работает утилита (по умолчанию — 3306).

--replace (-r)

Эта опция заставляет утилиту замещать существующие записи записями импортируемого файла.

--silent (-s)

Эта опция подавляет вывод на экран любых данных, кроме сообщений об ошибках.

--socket=файл (-S файл)

Эта опция переопределяет стандартный **сокет,** используемый для подключения к серверу. Обычно файл **сокета** называется `/tmp/mysql.sock`.

--user=имя (-u имя)

Эта опция задает имя пользователя, указываемое при регистрации на сервере.

--verbose (-v)

Эта опция **заставляет утилиту выдавать** более подробные комментарии.

--version (-V)

При наличии этой опции возвращается лишь информация о версии утилиты.

mysqlshow

Эта утилита представляет собой оболочку **инструкций** SHOW DATABASES, SHOW TABLES **и** SHOW COLUMNS. **Синтаксис ее вызова** таков:

```
mysqlshow
    [--character-sets-dir=каталог]
    [--compress I -C]
    [--debug[=конфигурация] I -# конфигурация]
    [--help | -?]
    [--host=узел | -h узел]
    [-- keys | -k]
    [--password[=пароль] | -p[пароль]]
    [--port=порт | -P порт]
    [--socket=файл | -S файл]
    [--status I -i]
    [--user=имя | -u файл]
     [--verbose I -v]
     [--version | -V]
    [база_данных [таблица [столбец]]]
```

Утилита возвращает информацию о **базах данных и таблицах. При** отсутствии аргументов **будет выдан** список **баз данных. Если** указано **имя базы данных, утилита вернет** список **содержащихся в ней таблиц. В имени таблицы могут** присутствовать **метасимволы** (`*`, `?`, `%`, `_`), что позволяет фильтровать список **таблиц.**

Если указана отдельная таблица, **будет выдана** информация о **ее** столбцах. **В спецификации** столбца тоже могут стоять метасимволы.

--character-sets-dir=каталог

Эта опция задает каталог, в котором хранятся файлы наборов символов.

--compress (-C)

При наличии этой опции утилита будет сжимать данные, передаваемые от клиента к серверу.

--debug[=конфигурация] (-# конфигурация)

Эта опция заставляет утилиту записывать отладочную информацию в указанный журнальный файл. Поддержка данной опции должна бытьвключена в утилиту на этапе компиляции. По умолчанию данные записываются в файл `/tmp/client.trace`. Подробнее о формате журнальных файлов рассказывается в главе 24, "Физическое хранениеданных".

--help (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются,

--host=узел (-h узел)

Эта опция задает узел, к которому попытается подключиться утилита. В качестве аргумента может присутствовать доменное имя или IP-адрес.

--keys (-k)

При наличии этойопции выдается информация об индексах таблиц.

--password[=пароль] (-p[пароль])

Эта опция задает пароль для подключения к серверу. При ее отсутствии **пароль** не посылается. Если же пароль не указан, программа попросит его ввести. Учтите, **что** пароль, вводимый в командной строке, могут увидеть другие пользователи, просматривающие список процессов. Обратите также внимание на то, что между названием короткой версии опции и паролем не нужен пробел.

--port=порт (-P порт)

Эта опция переопределяет стандартный номер порта, с которым работает утилита (по умолчанию — 3306).

--socket=файл (-S файл)

Эта опция переопределяет стандартный **сокет**, используемый для подключения к серверу. Обычно файл **сокета** называется `/tmp/mysql.sock`.

--status (-i)

При наличии этой опции выдается дополнительная информация о таблицах.

--user=имя (-u файл)

Эта опция задает имя пользователя, указываемое при регистрации на сервере.

--verbose (-v)

Эта опция заставляет утилиту выдавать более подробные комментарии.

--version (-V)

При наличии этой опции возвращается лишь информация о версии утилиты.

pack_isam

Эта утилита упаковывает таблицы ISAM точно так же, как это делает утилита `myisampack` в отношении таблиц MyISAM.

perror

Эта утилита возвращает описание числового кода ошибки. Формат ее вызова таков:

```
perror
    [--help I --info | -I I -?]
    [--silent | -s]
    [--verbose I -v]
    [--version | -V]
    код_ошибки [код_ошибки]...
```

С помощью утилиты perror можно преобразовывать коды ошибок в сообщения об ошибках (листинг 14.15).

```
# perror 1 2 3 4 5
Error code   1:  Operation not permitted
Error code   2:  No such file or directory
Error code   3:  No such process
Error code   4:  Interrupted system call
Error code   5:  Input/output error
```

--help (--info) (-I) (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

--silent (-s)

При наличии этой опции будет выдано лишь описание ошибки.

—verbose (-v)

Эта опция заставляет утилиту выдавать более подробные комментарии.

--version (-V)

При наличии этой опции возвращается лишь информация о версии утилиты.

replace

Эта утилита предназначена для замены строк в текстовых файлах. Ее синтаксис таков:

```
replaced
    [-I | -?]
    [-s]
    [-v]
    шаблон замена [шаблон замена]... [-- файл [файл]...]
```

Утилита меняет одну строку на другую. Можно задавать сразу несколько команд замены. По умолчанию текст читается из потока `stdin` и записывается в поток `stdout`, но опция — позволяет указывать список файлов для замены.

В табл. 14.5 описаны специальные коды, используемые в шаблонах.

Код	*Описание*
\^	Начало строки
\$	Конец строки
\Ь	Пробел, начало строки или конец строки

В листинге 14.16 обычные SQL-комментарии заменяются комментариями в стиле MySQL.

```
replace " --" " #"     /myscripts/*.sql
```

-I (-?)

При наличии этой опции выдается лишь описание синтаксиса командной строки, а все остальные опции игнорируются.

-S

Эта опция подавляет вывод на экран любых данных, кроме сообщений об ошибках.

-V

Эта опция заставляет утилиту выдавать более подробные сообщения.

safe_mysqld

Этот сценарий запускает демон mysqld и контролирует его перезапуск в случае сбоя. Именно такой способ вызова программы MySQL рекомендуется в UNIX-системах. Сценарий в первую очередь ищет утилиту mysqld-max, и только если она не найдена, запускается обычный демон mysqld. Сначала поиск ведется в текущем каталоге, а затем — в системных каталогах.

Все опции, указанные в командной строке, передаются демону MySQL. Параметры самого сценария safe_mysqld (перечислены ниже) задаются в конфигурационном файле.

defaults-extra-file=файл

Эта опция задает конфигурационный файл, который читается после файла глобальных установок, но перед любым конфигурационным файлом, находящимся в начальном каталоге текущего пользователя.

defaults-file=файл

Эта опция задает путевое имя глобального конфигурационного файла.

err-log=файл

Эта опция задает путевое имя журнала ошибок. В него включается информация о демоне mysqld, которую сам демон не смог сообщить, например сообщения о неудачных попытках запуска.

ledir=каталог

Эта опция задает каталог, в котором находится демон mysqld или mysqld-max.

mysqld-version=версия

Эта опция позволяет идентифицировать исполняемый файл сервера MySQL по суффиксу. Например, установка mysqld-version=max означает запуск файла mysqld-max. Пустая строка соответствует демону mysqld.

mysqld=версия

Эта опция задает имя исполняемого файла в каталоге, который указан в опции ledir.

no-defaults

Эта опция отключает конфигурационные файлы.

open-files-limit=число_файлов

Эта опция задает максимальное число файлов mysqld, которые могут быть открыты одновременно. Чтобы данная опция работала правильно, сценарий safe_mysqld должен быть запущен от имени пользователя root.

БИБЛИОТЕКА ФУНКЦИЙ ЯЗЫКА С

В этой главе…

Глава 15

В состав MySQL входит и библиотека функций языка C. Она предназначена для написания программ, взаимодействующих с сервером баз данных. В сочетании с библиотекой `mysqlclient` эта библиотека используется большинством утилит дистрибутива MySQL. Библиотеки других языков представляют собой оболочки библиотеки языка C.

Клиентская библиотека содержит также универсальные функции обработки массивов и строк, упрощающие взаимодействие с сервером. Их списки приведены в конце главы.

Типы данных

В клиентской библиотеке определен ряд типов данных. Большинство соответствующих структур объявлено в файле `include/mysql.h`. Существуют также функции, предназначенные для извлечения информации из этих структур.

MYSQL

Эта структура описывает сеанс подключения к серверу баз данных. Ее поля перечислены в табл. 15.1.

Поле	*Хранимая информация*
NET net	Параметры взаимодействия
`gptr connector_fd`	Дескриптор файла для протокола SSL
char *host	Адрес узла

Поле	*Хранимая информация*
char *user	Имя пользователя
char *passwd	Пароль
char *unix_socket	Путевое имя сокета
char *server_version	Версия сервера
char *host_info	Строка вида "Localhost via UNIX socket"
char *info	После выполнения запроса: строка вида "Rows matched: 2 Changed: 0 Warnings: O"
char *db	Имя стандартной базы данных
unsigned int port	Порт TCP/IP
unsigned int client_flag	Клиентские флаги
unsigned int server_capabilities	Параметры сервера
unsigned int protocol_version	Версия протокола
unsigned int field_count	Число полей в последней таблице результатов запроса
unsigned int server_status	Статус сервера
unsigned long thread_id	Идентификатор потока
my_ulonglong affected_rows	Число записей, участвовавших в последнем запросе
my_ulonglong insert_id	Идентификатор последней записи, при вставке которой было увеличено значение поля-счетчика
my_ulonglong extra_info	Значение, используемое утилитой mysqlshow
unsigned long packet_length	Длина пакета в байтах
enum mysql_status status	Одно из следующих значений: MYSQL_STATUS_READY, MYSQL_STATUS_GET_RESULT, MYSQL_STATUS_USE_RESULT

Поле	*Хранимая информация*
MYSQL_FIELD *fields	Массив с описаниями полей таблицы результатов
MEM_ROOT field_alloc	Буфер полей
my_bool free_me	True при вызове функции mysql_close()
my_bool reconnect	True, если соединение должно восстанавливаться в случае разрыва
struct st_mysql_options options	Различные опции, включая установки протокола SSL
char scramble_buff[9]	Случайная строка, посылаемая сервером при установлении соединения
struct charset_info_st *charset	Название стандартного набора символов
unsigned int server_language	Языксервера

MYSQL_DATA

Эта структура описывает данные, находящиеся в таблице результатов запроса (табл. 15.2).

Поле	*Хранимая информация*
my_ulonglong rows	Число строк в таблице результатов
unsigned int fields	Число столбцов в таблице результатов
MYSQL_ROWS *data	Записи таблицы
MEM_ROOT alloc	Резидентный буфер

MYSQL_FIELD

В этой структуре содержится информация о столбце таблицы (табл. 15.3). В структуру MYSQL входит массив структур MYSQL_FIELD, описывающих поля таблицы результатов.

Поле	*Хранимая информация*
char *name	**Имя** столбца
char *table	**Имя таблицы**
char *def	**Значение по** умолчанию
enum enum_field_types type	**Тип (табл.** 15.4)
unsigned int length	Размерность
unsigned int max_length	Максимальная размерность выбранного набора записей
unsigned int flags	Флаги
unsigned int decimals	Число десятичных значений

FIELD_TYPE_BLOB
FIELD_TYPE_DATE
FIELD_TYPE_DATETIME
FIELD_TYPE_DECIMAL
FIELD_TYPE_DOUBLE
FIELD_TYPE_ENUM
FIELD_TYPE_FLOAT
FIELD_TYPE_INT24
FIELD_TYPE_LONG
FIELD_TYPE_LONGLONG
FIELD_TYPE_LONG_BLOB
FIELD_TYPE_MEDIUM_BLOB
FIELD_TYPE_NEWDATE
FIELD_TYPE_NULL
FIELD_TYPE_SET
FIELD_TYPE_SHORT
FIELD_TYPE_STRING
FIELD_TYPE_TIME
FIELD_TYPE_TIMESTAMP

```
FIELD_TYPE_TINY
FIELD_TYPE_TINY_BLOB
FIELD_TYPE_VAR_STRING
FIELD TYPE YEAR
```

MYSQL_FIELD_OFFSET

В этой структуре хранится номер поля записи. Нумерация полей начинается с нуля.

MYSQL_RES

Эта структура описывает таблицу результатов запроса (табл. 15.5).

Поле	*Хранимая информация*
my_ulonglong row_count	Число строк в таблице результатов
unsigned int field_count	Число столбцов в таблице результатов
unsigned int current_field	Внутренний указатель на столбец
MYSQL_FIELD *fields	Массив с информацией о столбцах
MYSQL_DATA *data	Массив записей
MYSQL_ROWS *data_cursor	Указатель текущей записи
MEM_ROOT field_alloc	Резидентный буфер
MYSQL_ROW row	Описание текущей записи в случае режима небуферизованного чтения
MYSQL_ROW current_row	Запись, находящаяся в буфере
unsigned long *lengths	Массив длин текущих столбцов
MYSQL *handle	Дескриптор сеанса для режима небуферизованного чтения
my_bool eof	Маркер для функции mysql_fetch_row()

MYSQL_ROW

В этой структуре хранится содержимое одной записи таблицы.

MYSQL_ROWS

Эта структура описывает набор записей.

my_ulonglong

Это 64-разрядное целое число, используемое для подсчета количества записей, а также в качестве значения поля-счетчика. Оно определено как unsigned long long my_ulonglong.

Клиентские функции

В этом разделе описаны функции, используемые для взаимодействия с базой данных.

mysql_affected_rows()

Эта функция определяет количество строк, участвовавших в последнем запросе указанного сеанса. Ее прототип таков:

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Для операций обновления (UPDATE), удаления (DELETE) и вставки (INSERT) возвращается число измененных строк. Для операций выборки (SELECT) возвращается число строк в таблице результатов запроса.

В листинге 15.1 демонстрируются обновление нескольких записей и последующая проверка их числа с помощью функции mysql_affected_rows().

Листинг 15.1. Функция mysql_affected_

```
#include <stdio.h>
ttinclude<mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;
   char *query = "UPDATE book SET title=UPPER(title)";

   mysql_init(&mysql);

   /*
    ** Подключение к серверу.
    */
   if(!mysql_real_connect(&mysql,
                          "localhost",
                          "leon",
```

```
                        ""
                        "test", 0, NULL, 0))
    {
       fprintf(stderr,
               "Failed to connect to database. Error: %s\n",
               mysql_error(&mysql));
       exit();
    }

    /*
     ** Обновление нескольких записей.
     */
    if(mysql_query(&mysql, query))
    {
       fprintf(stderr, "Query failed: %s\n",
               mysql_error(&mysql));
    }
    else
    {
       printf("%d titles changed.",
              mysql_affected_rows(&mysql));
    }

    /*
     ** Закрытие соединения.
     */
    mysql_close(&mysql);
}
```

mysql_change_user()

Эта функция меняет пользователя и базу данных текущего сеанса. В случае успешного завершения возвращается нуль, иначе — код ошибки (перечислены далее в табл. 15.8). Прототип функции таков:

```
my_bool mysql_change_user(
    MYSQL  *mysql,
    const  char  *user,
    const  char *passwd,
    const  char *db)
```

Аргумент `mysql` — это описание соединения, возвращаемое функцией `mysql_real_connect()`. Значения аргументов user и password должны соответствовать реальному имени пользователя и паролю в таблицах привилегий. Последний аргумент является необязательным.

В листинге 15.2 функция `mysql_change_user()`, не разрывая соединение с сервером, отменяет какую-либо ассоциацию с пользователем или базой данных.

```
#include <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;

   mysql_init(&mysql);

   /*
    ** Подключение к серверу под видом обычного пользователя.
    */
   if(!mysql_real_connect(&mysql,
                          "localhost",
                          "leon",
                          "",
                          "test", 0, NULL, 0))
   {
      fprintf(stderr,
              "Failed to connect to database. Error: %s\n",
              mysql_error(&mysql));
      exit();
   }

   /*
    ** Удаление привязки к пользователю без разрыва соединения.
    */
   if(mysql_change_user(&mysql, NULL, NULL, NULL))
   {
      fprintf(stderr, "Failed to change user. Error: %s\n",
              mysql_error(&mysql));
   }

   /*
    ** Закрытие соединения.
    */
   mysql_close(&mysql);
}
```

mysql_character_set_name()

Эта функция возвращает название стандартного набора символов в данном сеансе:

```
const char * mysql_character_set_name(MYSQL *mysql)
```

Применение функции демонстрируется в листинге 15.3.

```
#include <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;

   mysql_init(&mysql);

   /*
    ** Подключение к серверу под видом обычного пользователя.
    */
   if(!mysql_real_connect(&mysql, "localhost", "leon", "",
                          "test", 0, NULL, 0»
   {
      fprintf(stderr,
              "Failed to connect to database. Error: %s\n",
             mysql_error(&mysql));
      exit ();
   }

   /*
    ** Выдача информации о сервере и клиенте.
    */
   printf("Character set: %s\n",
          mysql_character_set_name(&mysql));
   printf("Client Version: %s\n",
          mysql_get_client_info());
   printf("Host Info: %s\n",
          mysql_get_host_info(&mysql));
   printf("Protocol Version: %d\n",
          mysql_get_proto_info(&mysql));
   printf("Server Version: %s\n"",
          mysql_get_server_info(&mysql));
   printf("MySQL Info: %s\n",
          mysql_info(&mysql));
   printf("Thread Safe: %d\n",
          mysql_thread_safe());

   /*
    ** Закрытие соединения.
    */
   mysql_close(&mysql);
}
```

mysql_close()

Эта функция разрывает соединение с сервером и очищает память, связанную с сеансом:

void **mysql_close(MYSQL** *mysql)

mysql_connect()

Эта функция является устаревшим аналогом функции mysql_real_connect():

```
mysql_connect(
   MYSQL *mysql,
   const char *host,
   const char *user,
   const char *passwd)
```

Использовать ее не рекомендуется.

mysql_create_db()

Эта устаревшая функция создает базу данных, имя которой указано во втором аргументе:

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Использовать **эту функцию не** рекомендуется. **Лучше** посылать **инструкцию** CREATE DATABASE **с** помощью **функции** mysql_query().

mysql_data_seek()

Эта функция перемещает внутренний указатель на требуемую запись в буфере результатов:

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong row)
```

Применяя функцию mysql_data_seek() совместно с mysql_fetch_row(), можно считывать произвольную запись таблицы результатов. Перед обращением к данной функции необходимо вызвать функцию mysql_store_result (). **Использование** перечисленных функций демонстрируется в листинге 15.4.

```
#include <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;
   MYSQL_RES *result;
   MYSQL_ROW row;
   char *query = "SELECT State FROM tax ORDER BY State";

   mysql_init(&mysql);
   mysql_real_connect(&mysql, "localhost", "freetrade", "",
                      ''freetrade'', 0, NULL, 0);

   /*
    ** Извлечение информации из таблицы tax.
    */
```

```
    if(mysql_query(&mysql, query))
    {
       fprintf(stderr, "Query failed: %s\n",
               mysql_error(&mysql));
    }
    else
    {
       /*
        ** Занесение результатов в буфер.
        */
       result = mysql_store_result(&mysql);

       /*
        ** Переход на пятнадцатую запись.
        */
       raysql_data_seek(result,14);

       /*
        ** Выборка записи.
        */
       row = mysql_fetch_row(result);

       /*
        ** Отображение названия штата.
        */
       printf("%s\n", row[0]);
    }

    mysql_free_result(result);
    mysql_close(&mysql);
}
```

mysql_debug()

Эта функция включает режим отладки:

```
void mysql_debug(const char *debug)
```

Ей требуется передать строку формата наподобие `d:t:0, /tmp/client.trace` либо пустую строку. Во втором случае используется переменная среды MYSQL_DEBUG. Чтобы эта функция была доступна, необходимо скомпилировать клиентскую библиотеку с включением средств отладки.

В MySQL применяется библиотека функций отладки Фреда Фиша (Fred Fish). О ней рассказывается в главе 31, "Расширение возможностей MySQL".

mysql_drop_db()

Эта устаревшая функция полностью удаляет указанную базу данных и все ее таблицы:

```
intraysql_drop_db(MYSQL*mysql, const char *db)
```

Вместо нее рекомендуется посылать инструкцию DROP DATABASE с помощью функции `mysql_query()`.

mysql_dump_debug_info()

Эта функция заставляет сервер записывать отладочную информацию в журнальный файл:

```
int mysql_dump_debug_info(MYSQL *mysql)
```

mysql_eof()

Эта устаревшая функция проверяет, достигнут ли конец результирующего набора записей:

```
my_bool mysql_eof(MYSQL_RES *res)
```

Вместо нее предпочтительнее пользоваться функцией mysql_fetch_row().

mysql_errno()

Эта функция возвращает код ошибки последней операции:

```
uint mysql_errno(MYSQL *mysql)
```

Если ошибок не было, возвращается нуль. Список кодов ошибок и соответствующих им сообщений приведен в приложении Г, "Коды ошибок MySQL".

В листинге 15.5 будет выдано следующее сообщение:

```
Query failed: (1064) You have an error in your SQL syntax near
'FROM tax' at line 1.
```

```
#include <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;

   mysql_init(&mysql);
   mysql_real_connect(&mysql, "localhost", "freetrade", "",
                      "freetrade", 0, NULL, 0);

   /*
    ** Попытка выполнить неправильный запрос.
    */
   if(mysql_query(&mysql, "SELECT FROM tax"))
   {
      fprintf(stderr, "Query failed: (%d) %s\n",
              mysql_errno(&mysql), mysql_error(&mysql));
   }
   mysql_close(&mysql);
}
```

mysql_error()

Эта функция возвращает сообщение об ошибке, соответствующее последней операции:

```
char * mysql_error(MYSQL *mysql)
```

Если ошибок не было, возвращается пустая строка. Список кодов ошибок и соответствующих им сообщений приведен в приложении Г, "Коды ошибок MySQL". Пример использования функции `mysql_error()` приводился в листинге 15.5.

mysql_escape_string()

Эта устаревшая функция добавляет к заданной строке символы обратной косой черты, что позволяет использовать строку в SQL-инструкциях:

```
ulong mysql_escape_string(
    char *to,
    const char *from,
    ulong length)
```

Эта функция игнорирует сведения о наборе символов. Вместо нее лучше пользоваться функцией mysql_real_escape_string ().

mysql_fetch_field()

Эта функция извлекает информацию о столбцах таблицы результатов, по одному за раз:

```
MYSQL_FIELD * mysql_fetch_field(MYSQL_RES *result)
```

В MySQL хранится внутренний указатель на текущий столбец. Этот указатель изменяется после каждой следующей операции выборки столбца. Когда столбцов не остается, возвращается NULL. Указатель сбрасывается в момент поступления нового запроса. Его можно также изменять вручную с помощью функции `mysql_field_seek()`.

Функция `mysql_fetch_field()` возвращает структуру MYSQL_FIELD, описание которой было приведено выше. В режиме небуферизованного чтения размерность столбцов типа BLOB устанавливается равной 8 Кбайт, а не реальной размерности. Если же результаты запроса заносятся в буфер, учитывается размерность самого большого значения столбца.

В листинге 15.6 с помощью функции `mysql_fetch_field()` отображается информация о каждом столбце таблицы tax.

```
#include <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;
```

```
    MYSQL_RES *result;
    MYSQL_FIELD *field;

    mysql_init(&mysql);
    mysql_real_connect(&mysql, "localhost", "freetrade", "",
                       "freetrade", 0, NULL, 0);
    mysql_query(&mysql, "SELECT * FROM tax");
    result = mysql_use_result(&mysql);

    /*
     ** Получение информации о столбцах.
     */
    while((field = mysql_fetch_field(result)))
    {
       printf("Name: %s\n", field->name);
       printf("Table: %s\n", field->table);
       printf("Default Value: %s\n\n", field~>def);
    }

    mysql_free_result(result);
    mysql_close(&mysql);
}
```

mysql_fetch_field_direct()

Эта функция возвращает структуру MYSQL_FIELD с информацией о заданном столбце таблицы результатов:

```
MYSQL_FIELD * mysql_fetch_field_direct(
   MYSQL_RES *res,
   uint fieldnr)
```

Нумерация столбцов начинается с нуля. В остальном данная функция аналогична функции mysql_fetch_field().

mysql_fetch_fields()

Эта функция извлекает информацию обо всех столбцах таблицы результатов и возвращает массив структур MYSQL_FIELD:

```
MYSQL_FIELD * mysql_fetch_fields(MYSQL_RES *res)
```

Определить общее количество столбцов в таблице позволяет функция mysql_num_fields(). Программа, представленная в листинге 15.7, выводит данные о каждом элементе массива столбцов.

```
#include <stdio.h>
ttinclude <mysql/mysql.h>

int main(int argc, char *argv[])
{
```

```
    MYSQL mysql;
    MYSQL_RES *result;
    MYSQL_ROW  row;
    MYSQL_FIELD*field;
    unsigned int num_fields;
    unsigned long *lengths;
    unsigned int i;

    mysql_init(&mysql);
    mysql_real_connect(&mysql, "localhost", "freetrade", "",
                       "freetrade", 0, NULL, 0);
    mysql_query(&mysql, "SELECT * FROM tax");
    result = mysql_store_result(&mysql);
    row = mysql_fetch_row(result);

    /*
     ** Получение информации о столбцах.
     */
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    field = mysql_fetch_fields(result);
    for(i = 0; i < num_fields; i++)
    {
       printf("Name: %s\n", field[i].name);
       printf("Table: %s\n", field[i].table);
       printf("Default Value: %s\n",
              field[i].def ? field[i].def : "NULL");
       printf("Length: %lu bytes\n\n", lengths[i]);
    }

    mysql_free_result(result);
    mysql_close(&mysql);
}
```

mysql_fetch_lengths()

Эта функция возвращает массив, в котором указана размерность каждого столбца текущей строки:

```
ulong * mysql_fetch_lengths(MYSQL_RES *res)
```

Функция `mysql_fetch_lengths()` предпочтительнее, чем `strlen()`, так как последняя неправильно определяет длину двоичной строки. В случае ошибки, вызванной наличием неправильного указателя, возвращается NULL.

Пример использования данной функции приводился в листинге 15.7.

mysql_fetch_row()

Эта функция извлекает следующую запись из указанного набора:

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *res)
```

При достижении конца набора в режиме буферизованного чтения возвращается NULL. В небуферизованном режиме значение NULL является как признаком конца на-

бора, так и признаком ошибки. В этом случае необходимо проверить наличие ошибки с помощью функции mysql_errno().

Возвращаемое значение можно трактовать как массив значений столбцов. Их число определяет функция mysql_num_fields(), а функция mysql_fetch_lengths() находит длину каждого столбца.

В листинге 15.8 запрашиваются значения трех столбцов таблицы user. Обратите внимание на функцию printf() во внутреннем цикле. В ней проверяется реальная длина каждого столбца. С помощью оператора ? значения NULL преобразуются в строку "NULL". Указатель, содержащийся в массиве row, будет равен NULL, если значение в столбце отсутствует. Если же столбец содержит пустую строку, в массиве row будет находиться указатель на пустую строку.

```
#include <string.h>
#include <mysql/mysql.h>
int main(int argc, char *argv[])
{
   MYSQL mysql;
   MYSQL_RES *result;
   MYSQL_ROW row;
   MYSQL_FIELD *field;
   unsigned int num_fields;
   unsigned long *lengths;
   unsigned int i;
   char *col;

   mysql_init(&mysql);
   mysql_real_connect(&mysql, "localhost", "root", "",
                      "mysql", 0, NULL, 0);
   mysql_query(&mysql, "SELECT Host, User, Password FROM user");
   result = mysql_store_result(&mysql);

   /*
    ** Просмотр результатов.
    */
   num_fields = mysql_num_fields(result);

   while(row = mysql_fetch_row(result))
   {
      lengths = mysql_fetch_lengths(result);
      for(i = 0; i < num_fields; i++)
      {
         printf("[%.*s] ", (int) lengths[i],
                row[i] ? row[i] : "NULL");
      }
      printf("\n");
   }

   mysql_free_result(result);
   mysql_close(&mysql);
}
```

mysql_field_count()

Эта функция определяет количество столбцов, участвовавших в последней операции заданного сеанса:

```
unsigned int mysql_field_count(MYSQL *mysql)
```

Схожие действия выполняет функция `mysql_num_fields()`, которая в качестве аргумента принимает указатель на конкретную таблицу результатов.

mysql_field_seek()

Эта функция перемещает внутренний указатель на заданный столбец таблицы результатов:

```
MYSQL_FIELD_OFFSET mysql_field_seek(
    MYSQL_RES *result,
    MYSQL_FIELD_OFFSET  field_offset)
```

Первый столбец имеет нулевое смещение. Функция возвращает предыдущее значение указателя.

mysql_field_tell()

Эта функция возвращает значение внутреннего указателя полей:

```
uint mysql_field_tell(MYSQL_RES *res)
```

mysql_free_result()

Эта функция освобождает память, занимаемую таблицей результатов запроса:

```
void mysql_free_result(MYSQL_RES *result)
```

По окончании работы с набором записей необходимо всегда освобождать память.

mysql_get_client_info()

Эта функция определяет версию клиентской библиотеки, с которой ведется работа:

```
char * mysql_get_client_info(void)
```

Пример использования данной функции был приведен в листинге 15.3.

mysql_get_host_info()

Эта функция возвращает строку, описывающую узел, с которым установлено соединение:

```
char * mysql_get_host_info(MYSQL *mysql)
```

Пример использования данной функции был приведен в листинге 15.3.

mysql_get_proto_info()

Эта функция определяет версию применяемого коммуникационного протокола:

```
uint mysql_get_proto_info(MYSQL *mysql)
```

Пример использования данной функции был приведен в листинге 15.3.

mysql_get_server_info()

Эта функция определяет версию программы MySQL, установленной на сервере:

```
char * mysql_get_server_info(MYSQL *mysql)
```

Пример использования данной функции был приведен в листинге 15.3.

mysql_info()

Эта функция возвращает строку с описанием записей, измененных в ходе последнего запроса:

```
char *mysql_info(MYSQL *mysql)
```

Непустая строка возвращается лишь в том случае, когда в операции вставки (INSERT) или обновления (UPDATE) участвовала более чем одна запись. Например, если с помощью инструкции INSERT INTO в таблицу была добавлена одна запись, функция `mysql_info()` вернет пустую строку. Непустая строка возвращается также в случае выполнения инструкций ALTER TABLE и LOAD DATA INFILE.

mysql_init()

Эта функция инициализирует соединение:

```
MYSQL * mysql_init(MYSQL *mysql)
```

В качестве аргумента можно передать адрес существующей структуры MYSQL или NULL. Во втором случае функция выделит память под новую структуру. Для освобождения выделенной памяти предназначена функция `mysql_close()`.

mysql_insert_id()

Эта функция возвращает последнее значение поля-счетчика, использовавшееся в указанном сеансе:

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

То же самое значение возвращается встроенной SQL-функцией LAST_INSERT_ID(). Если в последнем запросе не участвовало поле-счетчик, функция `mysql_insert_id()` возвращает нуль.

mysql_kill()

Эта функция уничтожает поток, связанный с базой данных:

```
int mysql_kill(MYSQL *mysql, ulong pid)
```

Список идентификаторов потоков можно получить с помощью инструкции SHOW PROCESSES или функции mysql_list_processes(). В случае неудачи функция mysql_kill() возвращает нуль.

mysql_list_dbs()

Эта функция возвращает набор записей с именами всех баз данных:

```
MYSQL_RES * mysql_list_dbs(MYSQL *mysql, const char *wild)
```

Аргумент wild может содержать неполное имя с метасимволами (_ и %), что позволяет фильтровать результаты. Аналогичные действия выполняет инструкция SHOW DATABASES. Не забывайте, что полученный набор записей необходимо удалить с помощью функции mysql_free_result().

В листинге 159 выводится список баз данных, доступных пользователю root.

```
#include <string.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;
   MYSQL_RES *result;
   MYSQL_ROW row;
   MYSQL_FIELD *field;
   unsigned long *lengths;

   mysql_init(&mysql);
   mysql_real_connect(&mysql, "localhost", "root", "",
                      "mysql", 0, NULL, 0);
   result = mysql_list_dbs(&mysql, NULL);

   /*
    ** Просмотр результатов.
    */
   while(row = mysql_fetch_row(result))
   {
      lengths = mysql_fetch_lengths(result);
      printf("%.*s\n", (int) lengths[0], row[0]);
   }

   mysql_free_result(result);
   mysql_close(&mysql);
}
```

mysql_list_fields()

Эта функция возвращает набор записей с именами всех столбцов в указанной таблице:

```
MYSQL_RES * mysql_list_fields(
    MYSQL *mysql,
    const char *table,
    const char *wild)
```

Аргумент wild может содержать неполное имя с метасимволами (_ и %), что позволяет фильтровать результаты. Аналогичные действия выполняет инструкция SHOW COLUMNS. Не забывайте, что полученный набор записей необходимо удалить с помощью функции mysql_free_result().

mysql_list_processes()

Эта функция возвращает набор записей с идентификаторами всех потоков:

```
MYSQL_RES * mysql_list_processes(MYSQL *mysql)
```

Аналогичные действия выполняет инструкция SHOW PROCESSLIST. Не забывайте, что полученный набор записей необходимо удалить с помощью функции mysql_free_result().

mysql_list_tables()

Эта функция возвращает набор записей с именами всех таблиц указанной базы данных:

```
MYSQL_RES * mysql_list_tables(MYSQL *mysql, const char *wild)
```

Аргумент wild может содержать неполное имя с метасимволами (_ и %), что позволяет фильтровать результаты. Аналогичные действия выполняет инструкция SHOW TABLES. Не забывайте, что полученный набор записей необходимо удалить с помощью функции mysql_free_result().

mysql_num_fields()

Эта функция определяет количество полей в указанном наборе записей:

```
unsigned int mysql_num_fields(MYSQL_RES *res)
```

mysql_num_rows()

Эта функция определяет количество записей в указанном наборе:

```
my_ulonglong mysql_num_rows(MYSQL_RES *res)
```

mysql_options()

Эта функция задает параметры соединения до того, как оно будет установлено:

```
int mysql_options(
   MYSQL *mysql,
   enum mysql_option option,
   const char *arg)
```

Функция `mysql_options()` должна вызываться после `mysql_init()`, но до `mysql_real_connect()`. Она вызывается по одному разу для каждого параметра (табл. 15.6). В случае параметра `MYSQL_OPT_CONNECT_TIMEOUT` функции необходимо передать указатель на беззнаковое целое число, приведенный к типу `char *`.

Константа	*Тип аргумента*	*Директива*
`MYSQL_INIT_COMMAND`	`char *`	Немедленно выполнить указанную SQL-инструкцию при первом или повторном подключении
`MYSQL_OPT_COMPRESS`	Не используется	Сжимать данные, передаваемые между клиентом и сервером
`MYSQL_OPT_CONNECT_TIMEOUT`	`unsigned int *`	Считать попытку подключения неудачной по истечении указанного числа секунд
`MYSQL_OPT_NAMED_PIPE`	Не используется	Использовать именованные каналы в Windows NT
`MYSQL_READ_DEFAULT_FILE`	`char *`	Прочитать установки из указанного конфигурационного файла, а не того, который задан по умолчанию (например, `/etc/my.cnf`)
`MYSQL_READ_DEFAULT_GROUP`	`char *`	Прочитать установки из указанной группы конфигурационного файла

Благодаря функции `mysql_options()` разработчики MySQL могут добавлять к программе новые параметры соединений, не модифицируя код функции `mysql_real_connect()`. Пример задания параметров приведен в листинге 15.10.

```
#include <string.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;
   MYSQL_RES *result;
   MYSQL_ROW row;
   MYSQL_FIELD *field;
   unsigned long *lengths;
   uint timeout=60;

   mysql_init(&mysql);

   /*
    ** Задание параметров соединения.
    */
   mysql_options(&mysql,
                 MYSQL_OPT_COMPRESS, NULL);
   mysql_options(&mysql,
                 MYSQL_OPT_CONNECT_TIMEOUT, (char *)&timeout);
   mysql_options(&mysql,
                 MYSQL_INIT_COMMAND, "SET AUTOCOMMIT=0");

   mysql_real_connect(&mysql, "localhost", "root", "",
                      "mysql", 0, NULL, 0);
   result = mysql_list_processes(&mysql);

   while(row = mysql_fetch_row(result))
   {
      lengths = mysql_fetch_lengths(result);
      printf("%.*s\n", (int) lengths[0], row[0]);
   }

   mysql_free_result(result);
   mysql_close(&mysql);
}
```

mysql_ping()

Эта функция проверяет **наличие связи с сервером и в случае** необходимости повторно **устанавливает** соединение:

```
int mysql_ping(MYSQL *mysql)
```

mysql_query()

Эта функция выполняет указанный запрос в рамках заданного сеанса:

```
int mysql_query(MYSQL *mysql, const char *query)
```

Она представляет собой оболочку функции mysql_real_query(), вызывая функцию strlen() для определения длины запроса. Если в запросе содержится двоичная строка, необходимо обратиться к функции mysql_real_query() напрямую, сообщив ей реальную длину запроса.

Функция mysql_query() выполняет одиночный запрос. Символ точки с запятой в конце строки не нужен. За этой функцией должен следовать вызов функции mysql_store_result() или mysql_use_result().

В случае успешного выполнения запроса функция возвращает нуль, иначе — ненулевое значение. Применение этой функции демонстрируется во многих примерах данной главы.

mysql_read_query_result()

Эта функция извлекает результаты запроса, посланного ранее функцией mysql_send_query():

```
int mysql_read_query_result(MYSQL *mysql)
```

В случае успешного выполнения запроса функция возвращает нуль, иначе — код ошибки.

С помощью этих двух функций — mysql_read_query_result() и mysql_send_query() — можноорганизовать обработку запросов в фоновом режиме. Запрос посылается серверу, и пока этот запрос обрабатывается, выполняются другие действия (листинг 15.11).

```
ttinclude <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
    MYSQL *mysql;
    MYSQL_RES *result;
    MYSQL_ROW row;
    const char *query = "SELECT * FROM user";
    ulong *lengths;
    uint num_fields, i;

    mysql = mysql_init(NULL);

    mysql_real_connect(mysql, "localhost", "freetrade", "",
                       "freetrade", 0, NULL, 0);

    /*
     ** Отправка запроса.
     */
    mysql_send_query(mysql, query, strlen(query));

    /*
     ** Имитация других действий.
     */
```

```
    sleep(3);

    /*
     ** Получение результатов запроса.
     */
    if(!mysql_read_query_result(mysql))
    {
       result = mysql_store_result(mysql) ;
       num_fields = mysql_num_fields(result);

       while(row = mysql_fetch_row(result))
       {
          lengths = mysql_fetch_lengths(result) ;
          for(i = 0; i < num_fields; i++)
          {
             printf("[%.*s] ", (int) lengths[i],
                    row[i] ? row[i] : "NULL");
          }
          printf("\n");
       }

       mysql_free_result(result);
    }

    mysql_close(mysql);
}
```

mysql_real_connect()

Эта функция устанавливает соединение с сервером MySQL:

```
MYSQL * mysql_real_connect(
    MYSQL *mysql,
    const char *host,
    const char *user,
    const char *passwd,
    const char *db,
    uint port,
    const char *unix_socket,
    uint client_flag)
```

Она заменяет функцию mysql_connect (), использовавшуюся в предыдущих версиях MySQL. Как минимум, ей необходимо передать структуру MYSQL, **проинициализированную** функцией mysql_init (). Остальные аргументы можно задать равными NULL. В качестве аргумента host может присутствовать доменное имя или IP-адрес. Значение NULL соответствует локальному узлу. Функция пытается подключиться к локальному узлу, используя **сокеты** или именованные каналы, если они поддерживаются в операционной системе. Если указан IP-адрес, соединение устанавливается по протоколам TCP/IP.

Значение NULL в качестве имени пользователя соответствует текущему пользователю, запустившему программу. Пароль предоставляется в незашифрованном виде (функция сама выполняет шифрование). Если аргумент passwd равен NULL, пароль будет пустым.

Аргумент db определяет стандартную базу данных. Аргумент port задает номер порта TCP/IP. В случае нулевого значения будет использован стандартный порт 3306. Аргумент unix_socket должен содержать путь к файлу сокета или именованного канала.

Аргумент client_flag представляет собой битовое поле, в котором задаются различные опции (табл. 15.7). Некоторые из них перекрывают опции, заданные в функции mysql_options().

Константа	*Директива*
CLIENT_COMPRESS	Сжимать данные, передаваемые между клиентом и сервером
CLIENT_FOUND_ROWS	Сообщать число найденных записей, а не число измененных записей
CLIENT_IGNORE_SPACE	Разрешить пробелы между именем функции и открывающей скобкой
CLIENT_INTERACTIVE	Разрывать соединение по истечении интервала таймаута, заданного в переменной interactive_timeout, а не wait timeout
CLIENT_NO_SCHEMA	Не разрешать синтаксис *имя_БД.имя_таблицы.имя_столбца*
CLIENT_ODBC	Работать с клиентом по протоколу ODBC
CLIENT_SSL	Шифровать соединение с помощью протокола SSL

В табл. 15.8 перечислены коды ошибок, генерируемые функцией mysql_real_connect(). В случае успешного завершения функция возвращает указатель на структуру, содержащую описание соединения. При возникновении ошибки возвращается NULL.

Константа	*Описание*
CR_COMMANDS_OUT_OF_SYNC	Команды выполнялись в неправильном порядке
CR_SERVER_GONE_ERROR	Сервер не отвечает
CR_SERVER_LOST	Во время выполнения запроса пропала связь с сервером
CR_UNKNOWN_ERROR	Неизвестная ошибка
ER_ACCESS_DENIED_ERROR	Имя пользователя или пароль неверны
ER_BAD_DB_ERROR	Имя базы данных не распознано
ER_DBACCESS_DENIED_ERROR	Пользователю отказано в доступе
ER_UNKNOWN_COM_ERROR	Команда не реализована
ER_WRONG_DB_NAME	Имя базы данных слишком длинное

Применение функции mysql_real_connect() демонстрируется во многих примерах данной главы.

mysql_real_escape_string()

Эта функция вставляет в строку символы, отменяющие специальное назначение содержащихся в ней управляющих символов:

```
ulong mysql_real_escape_string(
    MYSQL *mysql,
    char  *to,
    const char *from,
    ulong length)
```

Функция извлекает строку из аргумента from и помещает ее преобразованную версию в аргумент to. Преобразование осуществляется с учетом используемого набора символов. Функция возвращает количество символов в полученной строке.

Аргумент to должен указывать на **массив**, в котором уместится преобразованная строка. Худший случай — это когда придется защищать от интерпретации каждый символ. Кроме того, добавляется еще и символ конца строки, поэтому к удвоенному размеру исходной строки нужно еще добавить единицу.

В листинге 15.12 функция mysql_real_escape_string() читает аргументы командной строки и заносит их в преобразованном виде в таблицу tax.

```
#include <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
    MYSQL *mysql;
    char *query;
    char state[1024];
    char rate [1024];
    char taxshipping;

    mysql = mysql_init(NULL);

    mysql_real_connect(mysql, "localhost", "freetrade", "",
                       "freetrade", 0, NULL, 0);

    if(argc != 4)
    {
       fprintf(stderr, "Wrong number of arguments!\n");
       exit();
    }

    /*
     ** Извлечение аргументов командной строки.
     */
    mysql_real_escape_string(mysql, state,
                             argv[1], strlen(argv[1]));
```

```
    mysql_real_escape_string(mysql, rate,
                             argv[2], strlen(argv[2]));
    taxshipping = argv[3][0];

    query = (char *)my_malloc (4096, 0);
    sprintf(query, "INSERT INTO tax VALUES('%s', %s, '%c')",
            state, rate, taxshipping);

    mysql_query(mysql, query);

    my_free(query, 0);

    mysql_close(mysql);
}
```

mysql_row_tell()

Эта функция возвращает смещение последней записи, извлеченной из набора с помощью функции `mysql_fetch_row()`:

```
MYSQL ROWS * mysql row tell(MYSQL RES *res)
```

mysql_select_db()

Эта функция задает стандартную базу данных:

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

В случае успешного завершения возвращается нуль.

mysql_send_query()

Эта функция посылает серверу запрос, но не ждет получения его результатов:

```
int mysql_send_query(MYSQL* mysql, const char* query, uint length)
```

Результаты можно извлечь позднее с помощью функции mysql_read_query_result(). С помощью этих двух функций можно организовать фоновую обработку длинных запросов, что позволит программе оперативнее реагировать на действия пользователя. Соответствующий пример приводился в листинге 15.11.

mysql_shutdown()

Эта функция останавливает работу сервера:

```
int mysql_shutdown(MYSQL *mysql)
```

mysql_ssl_cipher()

Эта функция возвращает SSL-шифр, используемый в указанном соединении:

```
char * mysql_ssl_cipher(MYSQL *mysql)
```

mysql_ssl_clear()

Эта функция освобождает память, выделенную функцией `mysql_ssl_set()`:

```
int mysql_ssl_clear(MYSQL *mysql)
```

mysql_ssl_set()

Эта функция подготавливает клиентскую программу к шифрованию соединения по протоколу SSL (Secure Sockets Layer — протокол защищенных сокетов):

```
int mysql_ssl_set(
   MYSQL *mysql,
   const char *key,
   const char *cert,
   const char *ca,
   const char *capath)
```

Она должна вызываться до того, как соединение будет установлено с помощью функции `mysql_real_connect()`. Об ошибках параметров функции `mysql_ssl_set()` сообщается только после того, как будет сделана попытка подключения. Память, выделяемая данной функцией, должна освобождаться функцией `mysql_ssl_clear()`.

Чтобы функция `mysql_ssl_set()` стала доступна, необходимо включить поддержку библиотеки OpenSSL на этапе компиляции клиентской библиотеки.

mysql_stat()

Эта функция возвращает строку с описанием состояния сервера:

```
char * mysql_stat(MYSQL *mysql)
```

Аналогичная информация возвращается командой `mysqladmin status`.

mysql_store_result()

Эта функция помещает результаты запроса во внутренний буфер:

```
MYSQL_RES * mysql_store_result(MYSQL *mysql)
```

Она возвращает идентификатор набора записей, который можно передавать функциям `mysql_fetch_row()` и `mysql_data_seek()`, а также ряду других функций. По окончании работы с набором записей его необходимо удалить с помощью функции `mysql_free_result()`.

Применение функции `mysql_store_result()` демонстрируется во многих примерах данной главы.

mysql_thread_id()

Эта функция возвращает идентификатор клиентского потока:

```
ulong mysql_thread_id(MYSQL *mysql)
```

Данный идентификатор появится в списке, который возвращается инструкцией SHOW PROCESSLIST. Его (идентификатор) можно передать функции `mysql_kill()`, чтобы разорвать соединение с сервером. В случае повторного подключения к серверу идентификатор потока меняется.

mysql_thread_safe()

Эта функция возвращает 1, если на этапе компиляции в клиентскую библиотеку были включены средства поддержки безопасной работы потоков:

```
uint mysql_thread_safe(void)
```

mysql_use_result()

Эта функция возвращает идентификатор набора записей, не находящегося в буфере:

MYSQL_RES * mysql_use_result(MYSQL *mysql)

Работая с набором записей данного типа, клиентская библиотека извлекает по одной записи за раз с помощью функции mysql_fetch_row(). Применять функции mysql_data_seek() и mysql_row_seek() к таким наборам нельзя.

Функции работы с массивами

Ниже перечислены функции, предназначенные для работы с динамическими массивами.

```
byte
*alloc_dynamic(DYNAMIC_ARRAY *array)

void
delete_dynamic(DYNAMIC_ARRAY *array)

void
delete_dynamic_element(DYNAMIC_ARRAY *array, uint idx)

void
freeze_size(DYNAMIC_ARRAY *array)

void
get_dynamic(DYNAMIC_ARRAY *array, gptr element, uint idx)

my_bool
init_dynamic_array(DYNAMIC_ARRAY *array,
                   uint element_size, uint init_alloc,
                   uint alloc_increment)
my_bool
insert_dynamic(DYNAMIC_ARRAY *array, gptr element)

byte
*pop_dynamic(DYNAMIC_ARRAY *array)

my_bool
set_dynamic(DYNAMIC_ARRAY *array, gptr element, uint idx)
```

Функции работы с наборами символов

Ниже перечислены функции, предназначенные для работы с наборами символов.

```
CHARSET_INFO *
add_charset(uint cs_number, const char *cs_name)

void .
charset_append(DYNAMIC STRING *s, const char *name)
```

```
my_bool
charset_in_string(const char *name, DYNAMIC_STRING *s)

my_bool
fill_array(uchar *array, int sz, struct simpleconfig_buf_st *fb)

CHARSET_INFO *
find_charset(CHARSET_INFO **table, uint cs_number, size_t tablesz)

CHARSET_INFO *
find_charset_by_name(CHARSET_INFO **table, const char *name,
                     size_t tablesz)

CHARSET_INFO *
find_compiled_charset(uint cs_number)

CHARSET_INFO *
find_compiled_charset_by_name(const char *name)

const char *
compiled_charset_name(uint8 charset_number)

uint8
compiled_charset_number(const char *name)

void
free_charsets(void)

CHARSET_INFO *
get_charset(uint cs_number, myf flags)

CHARSET_INFO *
get_charset_by_name(const char *cs_name, myf flags)

void
get_charset_conf_name(uint cs_number, char *buf)

const char *
get_charset_name(uint charset_number)

uint
get_charset_number(const char *charset_name)

char *
get_charsets_dir(char *buf)

CHARSET_INFO *
get_internal_charset(uint cs_number)

CHARSET_INFO *
get_internal_charset_by_name(const char *name)

my_bool
get_word(struct simpleconfig_buf_st *fb, char *buf)
```

```
my_bool
init_available_charsets(myf myflags)

char *
list_charsets(myf want_flags)

char *
name_from_csnum(CS_ID **cs, uint number)

uint
num_from_csname(CS_ID **cs, const char *name)

my_bool
read_charset_file(uint cs_number, CHARSET_INFO *set, myf myflags)

my_bool
read_charset_index(CS_ID ***charsets, myf myflags)

my_bool
set_default_charset(uint cs, myf flags)

my_bool
set_default_charset_by_name(const char *cs_name, myf flags)
```

Функции работы с файлами

Ниже перечислены функции, предназначенные для манипулирования файлами и каталогами.

```
char *
convert_dirname(my_string to)

uint
cleanup_dirname(register my_string to, char *from)

File
create_temp_file(char *to, char *dir, char *prefix, int mode,
                 myf MyFlags)

uint
dirname_length(const char *name)

uint
dirname_part(my_string to, char *name)

my_string
expand_tilde(my_string *path)

static char *
find_file_in_path(char *to, char *name)

my_string
fn_ext(const char *name)

my_string
```

```
fn_format(my_string to, char *name, char *dsk, char *form,
          int flag)

my_string
intern_filename(my_string to, const char *from)

void
make_ftype(my_string to, int flag)

int
my_close(File fd, myf MyFlags)

File
my_create(char *FileName, int CreateFlags, int access_flags,
          myf MyFlags)

int
my_delete(char *name, myf MyFlags)

my_string
my_filename(File fd)

int
my_fclose(FILE *fd, myf MyFlags)

FILE *
my_fdopen(File Filedes, char *name, int Flags, myf MyFlags)

FILE *
my_fopen(char *FileName, int Flags, myf MyFlags)

uint
my_fread(FILE *stream, byte *Buffer, uint Count, myf MyFlags)

my_off_t
my_fseek(FILE *stream, my_off_t pos, int whence, myf MyFlags)

my_off_t
my_ftell(FILE *stream, myf MyFlags)

uint
my_fwrite(FILE *stream, byte *Buffer, uint Count, myf MyFlags)

int
my_getwd(my_string buf, uint size, myf MyFlags)

my_string
my_load_path(my_string to, char *path, char *own_path_prefix)

File
my_open(char *FileName, int Flags, myf MyFlags)

my_string
my_path(my_string to, char *progname, char *own_pathname_part)

uint
```

```
my_read(File Filedes, byte *Buffer, uint Count, myf MyFlags)

File
my_register_filename(File fd, char *FileName,
                     enum file_type type_of_file,
                     uint error_message_number, myf MyFlags)

int
my_setwd(char *dir, myf MyFlags)

uint
my_write(int Filedes, byte *Buffer, uint Count, myf MyFlags)

void
pack_dirname(my_string to, char *from)

void
symdirget(char *dir)

int
test_if_hard_path(char *dir_name)

uint
unpack_dirname(my_string to, char *from)

my_string
unpack_filename(my_string to, char *from)

uint
system_filename(my_string to, char *from)

void
to_unix_path(my_string to)

int
wild_compare(char *str, char *wildstr)
```

Функции обработки ошибок

Ниже перечислены функции, предназначенные для обработки ошибок и выдачи сообщений об ошибках.

```
int
my_error(int nr, myf MyFlags, ...)

int
my_message_no_curses(uint error, char *str, myf MyFlags)

int
my_printf_error (uint error, char *format, myf MyFlags, ...)
```

Функции работы с хэш-таблицами

Перечисленные ниже функции реализуют **хэш-таблицу**, в которой нет переполнения страниц и пустых слотов. Кроме того, она защищена от проблем фрагментации.

```
my_bool
hash_check(HASH *hash)

my_bool
hash_delete(HASH *hash, byte *record)

byte *
hash_element(HASH *hash, uint idx)

void
hash_free(HASH *hash)

my_bool
hash_init(HASH *hash, uint size, uint key_offset, uint key_length,
          hash_get_key get_key, void (*free_element)(void*),
          uint flags)

my_bool
hash_insert(HASH *info, const byte *record)

byte *
hash_key(HASH *hash, const byte *record, uint *length,
         my_bool first)

uint
hash_mask(uint hashnr, uint buffmax, uint maxlength)

gptr
hash_next(HASH *hash, const byte *key, uint length)

uint
hash_rec_mask(HASH *hash, HASH_LINK *pos, uint buffmax,
              uint maxlength)

gptr
hash_search(HASH *hash, const byte *key, uint length)

my_bool
hash_update(HASH *hash, byte *record, byte *old_key,
            uint old_key_length)
```

Функции работы со списками

Ниже перечислены функции, предназначенные для работы с двухсвязными списками.

```
LIST *
list_add(LIST *root, LIST *element)
```

```
LIST *
list_cons(void *data, LIST *list)

LIST *
list_delete(LIST *root, LIST *element)

void
list_free(LIST *root, pbool free_data)

uint
list_length(LIST *list)

LIST *
list_reverse(LIST *root)

int
list_walk(LIST *list, list_walk_action action, gptr argument)
```

Функции управления памятью

Ниже перечислены функции, предназначенные для управления памятью.

```
gptr
alloc_root(MEM_ROOT *mem_root, unsigned int Size)

void
bchange(char *dst, uint old_length, char *src, uint new_length,
        uint tot_length)

void
bmove(char *dst, *src, uint len)

void
bmove_upp(char *dst, char *src, uint len)

void
free_root(MEM_ROOT *root, myf MyFlags)

void
init_alloc_root(MEM_ROOT *mem_root, uint block_size,
                uint pre_alloc_size)

char *
memdup_root(MEM_ROOT *root, char *str, uint len)

byte *
my_compress_alloc(byte *packet, ulong *len, ulong *complen)

gptr
my_malloc(unsigned int Size, myf MyFlags)

gptr
my_multi_malloc(myf myFlags, ...)

void
```

```
my_no_flags_free(gptr ptr)

gptr
my_once_alloc(unsigned int Size, myf MyFlags)

void
my_once_free(void)

gptr
my_realloc(gptr oldpoint, uint Size, myf MyFlags)

my_string
my_strdup(char *from, myf MyFlags)

char *
strdup_root(MEM_ROOT *root, char *str)
```

Функции работы с опциями

Перечисленные ниже функции предназначены для управления конфигурационными файлами.

```
void
free_defaults(char **argv)

void
load_defaults(const char *conf_file, const char **groups,
              int *argc, char ***argv)

void
print_defaults(const char *conf_file, const char **groups)

my_bool
search_default_file(DYNAMIC_ARRAY *args, MEM_ROOT *alloc,
                    const char *dir, const char *config_file,
                    const char *ext, TYPELIB *group);
```

Функции обработки паролей

Следующие две функции предназначены для работы с паролями.

```
char *
get_tty_password(char *opt_message)

void
make_scrambled_password(char *to, const char *password)
```

Функции обработки строк

Ниже перечислены функции, манипулирующие строками, включая функции преобразования строк в данные других типов.

```
ulong
atoi_octal(char *str)

void
casedn(my_string str, uint length)

void
casedn_str(my_string str)

void
caseup(my_string str, uint length)

void
caseup_str(my_string str)

my_bool
dynstr_append(DYNAMIC_STRING *str, char *append)

my_bool
dynstr_append_mem(DYNAMIC_STRING *str, char *append, uint length)

void
dynstr_free(DYNAMIC_STRING *str)

my_bool
dynstr_realloc(DYNAMIC_STRING *str, ulong additional_size)

my_bool
dynstr_set(DYNAMIC_STRING *str, char *init_str)

my_bool
init_dynamic_string(DYNAMIC_STRING *str, const char *init_str,
                    uint init_alloc, uint alloc_increment)

char *
int10_to_str(long int val, char *dst, int radix)

char *
int2str(long int val, char *dst, int radix)

int
is_prefix(char *s, char *t)

char *
llstr(longlong value, char *buff)

char *
longlong2str(longlong val, char *dst, int radix)

int
my_casecmp(const char *s, const char *t, uint len)

my_bool
my_compress(byte *packet, ulong *len, ulong *complen)

void
```

```
my_inet_ntoa(struct in_addr in, char *buf)

char *
my_itoa(int val, char *dst, int radix)

char *
my_ltoa(long int val, char *dst, int radix)

int
my_sortcmp(const char *s, const char *t, uint len)

int
my_sortncmp(const char *s, uint s_len, const char *t, uint t_len)

my_string
my_strcasestr(const char *str, const char *search)

int
my_strcasecmp(const char *s, const char *t)

int
my_strsortcmp(const char *s, const char *t)

my_bool
my_uncompress (byte *packet, ulong *len, ulong *complen)

char *
str2int(register const char *src, register int radix,
        long int lower, long int upper, long int  *val)

char *
strcend(register const char *s, register pchar c)

my_string
strcont(reg1 const char *str, reg2 const char *set)

char *
strend(register const char *s)

my_string
strfill(my_string s, uint len, pchar fill)

uint
strinstr(char *str, char *search)

char *
strmake(char *dst, char *src, uint length)

char *
strmov(char *dst, char *src)

uint
strnlen(char *s, uint maxlen)

char *
strnmov(char *dst, char *src, uint n)
```

```
char *
strtol(char *src, char **ptr, int base)

char *
strtoul(char *src, char **ptr, int base)

char *
strtoll(char *src, char **ptr, int base)

char *
strtoull(char *src, char **ptr, int base)

char *
strxmov(char *dst, const char *src, ...)
```

Функции работы с потоками

Ниже перечислены функции, реализующие **системно-независимые** потоки, а также исключающие семафоры **(мьютексы)**.

```
void
my_pthread_attr_setprio(pthread_attr_t *attr, int priority)

int
my_pthread_getprio(pthread_t thread_id)

void *
my_pthread_getspecific_imp(pthread_key_t key)

void
my_pthread_setprio(pthread_t thread_id, int prior)

int
safe_cond_timedwait(pthread_cond_t *cond, safe_mutex_t *mp,
                    struct timespec *abstime, char *file,
                    uint line)

int
safe_cond_wait(pthread_cond_t *cond, safe_mutex_t *mp, char *file,
               uint line)

int
safe_mutex_destroy(safe_mutex_t *mp, char *file, uint line)

int
safe_mutex_init(safe_mutex_t *mp, const pthread_mutexattr_t *attr)

int
safe_mutex_lock(safe_mutex_t *mp, char *file, uint line)

int -
safe mutex unlock(safe mutex_t *mp, char *file, uint line)

int
my sigwait(sigset t *set, int *sig)
```

Часть III

СОЗДАНИЕ КЛИЕНТОВ MYSQL

В этой части рассматриваются вопросы написания приложений, взаимодействующих с сервером MySQL. Для этих целей в MySQL есть базовая библиотека функций языка C. Модули других языков либо ставят этим функциям в соответствие свои локальные функции, либо создают для них **функции-"оболочки".** Все содержащиеся в данной части главы коротки и конкретны. Предполагается, что читатели имеют опыт программирования на соответствующем языке.

В главе 16, "Использование библиотеки языка C", рассматривается процедура написания приложений, непосредственно работающих с функциями языка C. В главе 17, **"JDBC",** рассказывается о программировании приложений на языке **Java** с использованием интерфейса JDBC. В главе 18, "VBScript и ODBC", описывается процесс создания **Web-приложений** с применением технологии ASP и интерфейса ODBC. В главе 19, **"PHP",** речь идет о создании **Web-приложений** на языке PHP. В главе 20, **"Perl",** рассматриваются язык Perl и библиотека DBI. В главе **21,** "Python", описываются язык Python и его модуль **MySQLdb.** В главе 22, "Библиотека **MySQL++",** описывается **библиотека** функций языка C++, которая называется **MySQL++.**

ИСПОЛЬЗОВАНИЕ БИБЛИОТЕКИ ЯЗЫКА С

В этой главе...

Подготовка программы
Извлечение данных
Изменение **данных**

Глава 16

В этой главе рассказывается о том, как писать программы на языке C, взаимодействующие с сервером MySQL. Собственно программирование на языке C выходит за рамки данной книги. Предполагается, что читатели имеют опыт написания и компиляции таких программ. Программы для MySQL можно компилировать в большинстве версий UNIX и Windows.

Функции библиотеки языка C рассматривались в главе 15, **"Библиотека функций языка C".**

Подготовка программы

При работе с клиентской библиотекой MySQL необходимо включить в программу файл **mysql.h,** а затем подключить файл библиотеки на этапе компоновки. Файлы заголовков обычно находятся в каталоге **/usr/local/include/mysql,** а **библиотечные файлы —** в каталоге **/usr/local/lib/mysql.** Для подключения библиотеки укажите в командной строке компилятора ключ **-lmysqlclient.**

Если компилятор **gcc** не находит файлы заголовков, воспользуйтесь опцией **-I,** а если не найден файл библиотеки — опцией -L. Ниже показана команда компиляции тестового клиента в системе, где программа MySQL инсталлирована в стандартные каталоги.

```
gcc -o testmy testmy.c -L/usr/local/lib/mysql -Imysqlclient
```

Естественно, в более сложных случаях необходимо создавать **make-файл.**

В Windows файл **libmysql.** lib представляет собой оболочку динамической **библиотеки libmysql.dll.** Статическая версия библиотеки называется **mysqlclient.** lib.

Любая клиентская программа, прежде чем подключаться к серверу, должна в первую очередь инициализировать структуру MYSQL, содержащую описание соединения. Это делает функция **mysql_init().** Она принимает указатель на существующую структуру или самостоятельно выделяет для нее память. По окончании сеанса необходимо освободить память с помощью функции **mysql_close().** Со структурой MYSQL работают многие функции библиотеки языка C.

После инициализации сеанса необходимо подключиться к серверу с помощью функции mysql_real_connect(). Все ее аргументы, кроме структуры MYSQL, являются необязательными. На место отсутствующих параметров подставляются значения по умолчанию. Например, в UNIX клиент по умолчанию подключается к узлу localhost через локальный сокет. Для регистрации на сервере используется имя текущего пользователя.

Перед подключением можно задать различные параметры сеанса с помощью функции mysql_options(), а также подготовить программу к шифрованию соединения, вызвав функцию mysql_ssl_set(). В ходе самого сеанса разрешается выполнять произвольное число запросов. В конце сеанса вызывается функция mysql_close().

В листинге 16.1 показана минимальная программа, которая не делает ничего полезного, а лишь подключается к серверу, придерживаясь стандартных установок.

```
#include <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;

   /*
   ** Инициализация соединения.
   */
   mysql_init(&mysql);

   /*
   ** Подключение к Сазе данных со стандартными установками.
   */
   mysql_real_connect(&mysql, NULL, NULL, NULL, NULL, 0, NULL, 0);

   /*
   ** Закрытие соединения.
   */
   mysql_close(&mysql);
}
```

Извлечение данных

После подключения к серверу баз данных можно посылать ему запросы с помощью функций mysql_query(), mysql_real_query() и mysql_send_query(). Первая из них определяет длину переданной ей строки запроса с помощью функции strlen(), поэтому в запрос не могут входить строки, содержащие символы NUL (ASCII-код 0). Это ограничение легко обойти. Достаточно вызвать функцию mysql_real_escape_string(), которая защитит все специальные символы от интерпретации.

При использовании функций mysql_query() и mysql_real_query() клиент посылает серверу запрос и ждет, пока сервер не вернет его результаты. Если запрос выполняется слишком долго, вызовите функцию mysql_send_query(), которая отпра-

вит запрос и немедленно завершится. Получить результаты можно будет позднее с помощью функции mysql_read_query_result(). Но не ждите слишком долго, поскольку сервер может разорвать соединение по истечении тайм-аута. Эти функции предназначены для того, чтобы можно было запустить медленный запрос в фоновом режиме и продолжить реагировать на действия пользователя.

Инструкция SELECT и ряд других инструкций, в частности SHOW PROCESSLIST, возвращают результаты запроса в виде набора записей. Клиент может поместить весь набор в буфер или извлекать по одной записи за раз.Работа с буфером ведется чуть быстрее, но необходимо иметь достаточный объем памяти, чтобы занести в буфер все записи. Функция mysql_store_result() помещает результаты запроса в буфер, а функция mysql_use_result() подготавливает программу к режиму небуферизованного чтения.

В любом случае отдельные записи извлекаются с помощью функции mysql_fetch_row(). За один вызов возвращается одна запись. Тип записи — MYSQL_ROW. Это системно-независимое представление строки таблицы. Если записи находятся в буфере, можно воспользоваться функцией mysql_data_seek() для перехода к произвольной записи. В режиме небуферизованного чтения менять внутренний указатель записей нельзя.

Функция mysql_num_rows() позволяет определить число записей в наборе, но в большинстве случаев просто вызывают функцию mysql_fetch_row() до тех пор, пока она не вернет значение NULL. Если при последующем вызове функции mysql_errno() возвращается нуль, значит, достигнут конец набора записей.

В листинге 16.2 показана простая программа, которая извлекает данные из таблицы user. Обратите внимание на то, что программа пытается подключиться к серверу от имени пользователя root с явно неправильным паролем. Если будете экспериментировать с этой программой, подставьте корректные значения имени пользователя и пароля.

```
#include <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;
   MYSQL_RES *result;
   MYSQL_ROW row;

   uint num_fields, i;
   ulong *lengths;

   /*
   ** Инициализация соединения.
   */
   if(!mysql_init(&mysql))
   {
      fprintf(stderr, "Unable to initialize MYSQL struct!\n");
      exit();
   }
```

```
    /*
    ** Подключение к базе данных.
    */
    if(!mysql_real_connect(&mysql, "localhost", "root", "password",
                           "mysql", 0, NULL, 0))
    {
       fprintf(stderr, "%d: %s\n", mysql_errno(&mysql),
               mysql_error(&mysql));
       exit();
    }

    /*
    ** Передача запроса серверу.
    */
    if(mysql_query(&mysql,
                   "SELECT User, Host FROM user ORDER BY 1,2"))
    {
       /*
       ** Запрос завершился неудачей!
       */
       fprintf(stderr, "%d: %s\n", mysql_errno(&mysql),
               mysql_error(&mysql));
    }
    else
    {
       /*
       ** Занесение результатов в буфер, подсчет числа полей.
       */
       result = mysql_store_result(&mysql);
       num_fields = mysql_num_fields(result);

       while(row = mysql_fetch_row(result))
       {
          lengths = mysql_fetch_lengths(result);
          for(i = 0; i < num_fields; i++)
          {
             printf("[%.*s]", (int) lengths[i],
                    row[i] ? row[i] : "NULL");
          }
          printf("\n");
       }

       /*
       ** Освобождение буфера.
       */
       mysql_free_result(result);
    }

    /*
    ** Закрытие соединения.
    */
    mysql_close(&mysql);
}
```

Остановимся на том фрагменте программы, где результаты запроса заносятся в буфер. В принципе, заранее известно, что таблица результатов содержит два столбца, но программа написана таким образом, что могут обрабатываться результаты произвольной инструкции SELECT. Определив с помощью функции `mysql_num_fields()` число столбцов, мы получаем возможность в цикле пройти по каждому полю каждой записи. Запись представляет собой массив полей, нумерация которых начинается с нуля. Если необходимо определить типы полей, воспользуйтесь функцией mysql_fetch_fields(), `mysql_fetch_field()` либо `mysql_fetch_field_direct()`.

В цикле while из таблицы результатов последовательно извлекаются записи. Функция `mysql_fetch_lengths()` определяет размерность каждого столбца. Это не та размерность, которая указана в определении столбца, а реальная длина строки, содержащейся в соответствующей ячейке. Функция `mysql_fetch_lengths()` чрезвычайно удобна, поскольку с помощью функции `strlen()` нельзя определять длину двоичных строк.

В цикле for функция `printf()` отображает значение каждой ячейки. Обратите внимание на спецификацию `*` в строке формата. Она заставляет функцию искать в списке аргументов целое число, задающее размерность строкового аргумента. Благодаря этому пропадает необходимость завершать каждую строку символом NUL.

Запись, возвращаемая функцией `mysql_fetch_row()`, представляет собой массив указателей на строки. В случае пустой строки элемент массива ссылается на строку, которая начинается с символа NUL, а функция `mysql_fetch_lengths()` сообщает о том, что длина такой строки равна нулю. Если же столбец содержит значение NULL, то в массиве будет находиться пустой указатель. С помощью оператора `?` такому указателю ставится в соответствие строка `"NULL"`.

Изменение данных

Запросы на вставку или обновление данных не возвращают наборы записей. Они тоже выполняются с помощью функции mysql_query(), но вызывать функцию `mysql_store_result()` нет необходимости. Если требуется узнать число добавленных или измененных записей, воспользуйтесь функцией `mysql_affected_rows()` (листинг 16.3).

```
#include <stdio.h>
#include <mysql/mysql.h>

int main(int argc, char *argv[])
{
   MYSQL mysql;
   MYSQL_RES *result;
   MYSQL_ROW row;

   uint i;
   char query[4096];

   /*
```

```
** Вставляемые данные.
*/
const char *names[] = {
     "Leon", "Vicky", "Carl", "Ricky", "Nicki", "Jeff",
     "Bob", "Tina", "Joey"
};
uint num_rows = sizeof(names)/sizeof(char *);

/*
** Запросы.
*/
const char *query_create =
     "CREATE TABLE testapi (\
     ID INT(11) NOT NULL AUTO_INCREMENT, \
     Name VARCHAR(64), \
     PRIMARY KEY(ID))";

const char *query_insert =
     "INSERT INTO testapi (Name) VALUES ('%s')";

const char *query_delete =
     "DELETE FROM testapi WHERE ID < 4";

const char *query_update =
     "UPDATE testapi SET Name = 'None' WHERE ID=5";

const char *query_drop = "DROP TABLE testapi";


/*
** Инициализация соединения.
*/
if(!mysql_init(&mysql))
{
   fprintf(stderr, "Unable to initialize MYSQL struct!\n");
   exit ();
}

/*
** Подключение к базе данных.
*/
if(!mysql_real_connect(&mysql, "localhost", NULL, NULL,
                       "test", 0, NULL, 0))
{
   fprintf(stderr, "%d: %s\n", mysql_errno(&mysql),
           mysql_error(&mysql));
   exit();
}

/*
** Создание таблицы.
*/
if(mysql_query(&mysql, query_create))
{
   /*
   ** Запрос завершился неудачей!
```

```
   */
   fprintf(stderr, "Could not create table!\n%d: %s\n",
           mysql_errno(&mysql), mysql_error(&mysql));
   mysql_close(&mysql);
   exit();
}

/*
** Вставка записей.
*/
for(i=0; i<num_rows; i++)
{
   sprintf(query, query_insert, names[i]);
   if(mysql_query(&mysql, query))
   {
      /*
      ** Запрос завершился неудачей!
      */
      fprintf(stderr, "Could not insert row!\n%s\n%d: %s\n",
              query, mysql_errno(&mysql), mysql_error(&mysql)
      mysql_close(&mysql);
      exit();
   }

   printf("%d row insert\n", mysql_affected_rows(&mysql));
}

/*
** Удаление записей.
*/
if(mysql_query(&mysql, query_delete))
{
   /*
   ** Запрос завершился неудачей!
   */
   fprintf(stderr, "Could not delete rows!\n%d: %s\n",
           mysql_errno(&mysql), mysql_error(&mysql));
   mysql_close(&mysql);
   exit();
}

printf("%d rows deleted\n", mysql_affected_rows(&mysql));

/*
** Обновление записей.
*/
if(mysql_query(&mysql, query_update))
{
   /*
   ** Запрос завершился неудачей!
   */
   fprintf(stderr, "Could not update rows!\n%d: %s\n",
           mysql_errno(&mysql), mysql_error(&mysql));
   mysql_close(&mysql);
   exit();
}
```

```
    printf("%d rows updated\n", mysql_affected_rows(&mysql));

    /*
    ** Удаление таблицы.
    */
    if(mysql_query(&mysql, query_drop))
    {
       /*
       ** Запрос завершился неудачей!
       */
       fprintf(stderr, "Could not drop table!\n%d: %s\n",
               mysql_errno(&mysql), mysql_error(&mysql));
       mysql_close(&mysql);
       exit();
    }

    /*
    ** Закрытие соединения.
    */
    mysql_close(&mysql);
}
```

Программа, представленная в листинге 16.3, создает таблицу и добавляет в нее записи. Далее происходит удаление части записей, обновление записи и, наконец, удаление всей таблицы. Обратите внимание на способ вставки записей. В цикле `for` благодаря функции `sprintf()` происходит повторное использование одной и той же инструкции INSERT, в которую подставляются разные параметры. Эта простая **методика** не так эффективна, как подготовка одной инструкции INSERT, вставляющей группу записей. Но представьте себе программу, принимающую произвольные **пользовательские** запросы на добавление, обновление и удаление записей. Тогда такая методика окажется весьма удобной.

JDBC

Глава 17

JDBC (Java Database Connectivity) — это стандартный интерфейс Java, предназначенный для взаимодействия с базами данных. Он поддерживает работу как с реляционными базами данных, так и с простыми табличными данными, например с текстовыми файлами, поля которых разделены символами табуляции. Интерфейсом JDBC пользуются как прикладные программисты, так и разработчики драйверов баз данных. Существует несколько **JDBC-драйверов** для MySQL. Новейшие их версии доступны по адресу *http://mmmysql.sourceforge.net.*

В этой главе предполагается, что читатели знакомы с языком Java и умеют компилировать **Java-программы.** Каждый работает в той среде, которая ему нравится. Для простоты я воспользовался утилитами компании Sun, установив их в Windows.

Подготовка программы

Интерфейс JDBC является частью пакета Java 2 SDK. Чтобы иметь возможность подключаться к серверу MySQL, необходимо добавить каталог драйвера MySQL в список путевых имен классов. Я инсталлировал драйвер MM.MySQL в каталог `jdk1.3`. В результате был создан подкаталог `mm.mysql.jdbc-1.` 2c. Затем я расширил список путевых имен классов, чтобы он выглядел следующим образом:

```
.; C:\JDK1.3\LIB; C:\JDK1.3\MMMYSQ~1.2C
```

Обратите внимание на формат имен 8.3, свойственный Windows.

Библиотека функций JDBC инкапсулирована в пакете `java.sql`, поэтому программа должна импортировать все его классы (`java.sql.*`). Сам драйвер неимпортируется. Он загружается на этапе выполнения программы с помощью метода `Class.forName()`. Имя драйвера MM.MySQLвыглядит так: `org.gjt.mm.mysql.`Driver. Этот класс, как и все драйверы JDBC, реализует интерфейс `java.sql.` Driver.

После загрузки драйвера воспользуйтесь услугами менеджера драйверов для установления соединения с сервером. Для этого нужно сформировать URL-адрес соединения. Формат адреса таков:

```
jdbc:mysql: // [узел] [:порт]/база_данных[?параметр=значение]
                                        [параметр=значение]...
```

В табл. 17.1 приведен список параметров, которые можно указывать в строке подключения.

Параметр	*Описание*
autoReconnect	Если равен True, драйвер будет повторно подключаться к серверу в случае потери связи с ним; по умолчанию равен False
characterEncoding	Если параметр useUnicode равен True, то данный параметр задает используемый набор символов
initialTimeout	Задает интервал времени в секундах между попытками подключения к серверу (по умолчанию — 2 секунды)
maxReconnects	Задает максимальное число повторных попыток подключения к серверу (по умолчанию — 3)
maxRows	Задает максимальное число записей, которое может быть получено в ответ на запрос; по умолчанию равен нулю, т.е. ограничение отсутствует
password	Задает пароль для подключения
user	Задает имя пользователя
useUnicode	Если равен True, драйвер будет кодировать строки по стандарту Unicode; по умолчанию равен False

В рамках сеанса можно посылать произвольное число инструкций. Чтобы закрыть соединение, вызовите метод close().

В листинге 17.1 показана минимальная Java-программа, которая подключается к серверу и тут же отключается.

```
import java.sql.*;

public class CoreConnect
{
   public static void main(String[] Args)
   {
```

```
        // Загрузка драйвера.
        try
        {
           Class.forName("org.gjt.mm.mysql.Driver").newInstance();
        }
        catch (Exception e)
        {
           e.printStackTrace();
        }

        try
        {
           // Подключение к базе данных.
           Connection c = DriverManager.getConnection(
                "jdbc:mysql://localhost/test?user=root");

           // Закрытие соединения.
           c.close();
        }
        catch (SQLException e)
        {
           e.printStackTrace();
        }
    }
}
```

Извлечение данных

Прежде чем посылать запросы базе данных, необходимо создать объект класса Statement. Его можно многократно использовать в запросах. В классе Statement есть три метода выполнения запросов: `execute()`, `executeQuery()` и `executeUpdate()`. Каждый из них в качестве аргумента принимает текст запроса. Метод `executeQuery()` возвращает объект класса `ResultSet`, содержащий результирующий набор записей. Метод `executeUpdate()` возвращает число записей, участвовавших в запросе (это особенно удобно для инструкций INSERT и UPDATE). Метод `execute()` возвращает -1 в случае инструкции SELECT и число записей для остальных запросов. Его удобно использовать тогда, когда тип выполняемого запроса неизвестен заранее. В случае необходимости результирующий набор записей можно будет получить с помощью метода `getResultSet()`.

Объект класса ResultSet хранит указатель на текущую запись результирующего набора. Для получения первой записи необходимо вызвать метод `next()`. Этот метод удобно использовать в качестве условия цикла, так как по достижении конца набора он возвращает NULL. После извлечения записи можно определить значения ее полей с помощью одного из методов семейства `getX()`. Полное имя метода соответствует типу возвращаемого значения. Например, метод `getInt()` возвращает числовое представление ячейки, а метод `getDate()` возвращает объект класса Date. Можно вызывать любой метод — драйвер самостоятельно выполнит необходимое преобразование.

В методах семейства `getX()` столбцы задаются по имени или по номеру. В JDBC нумерация столбцов начинается с единицы. Если столбцы адресуются по имени, не-

обходимо, чтобы результирующий набор записей содержал различающиеся имена столбцов. Для устранения неоднозначностей пользуйтесь псевдонимами столбцов.

В листинге 17.2 показана простая программа, которая извлекает данные из таблицы user. Обратите внимание на то, что программа пытается подключиться к серверу от имени пользователя root с явно неправильным паролем. Если будете экспериментировать с этой программой, подставьте корректные значения имени пользователя и пароля.

```
import java.sql.*;

public class CoreSelect
{
   public static void main(String[] Args)
   {
      // Загрузка драйвера.
      try
      {
         Class.forName("org.gjt.mm.mysql.Driver").newInstance();
      }
      catch (Exception e)
      {
         e.printStackTrace();
      }

      try
      {
         // Подключение к базе данных.
         Connection c = DriverManager.getConnection(
              "jdbc:mysql://localhost/test?user=root");

         // Создание инструкции.
         Statement stmt = c.createStatement();

         // Выполнение запроса.
         ResultSet rs = stmt.executeQuery(
              "SELECT User, Host FROM user ORDER BY 1,2");

         // Получение метаданных.
         ResultSetMetaData rsmd = rs.getMetaData();

         // Вывод результатов запроса.
         while (rs.next())
         {
            // Перебор всех столбцов.
            for(int i=1; i<=rsmd.getColumnCount(); i++)
            {
               System.out.print("[" + rs.getString(i) + "]");
            }

            System.out.println("");
         }
```

```
            // Удаление инструкции.
            stmt.close();

            // Закрытие соединения.
            c.close();
         }
         catch (SQLException e)
         {
            e.printStackTrace();
         }
      }
}
```

Если число столбцов в таблице результатов запроса неизвестно, необходимо создать объект класса `ResultSetMetaData`. В этот класс входят методы, позволяющие определять количество столбцов, их имена, размерности и т.д.

Изменение данных

Запросы на вставку или обновление данных не возвращают наборы записей. Используйте метод `executeUpdate()`, который возвращает число записей, участвовавших в запросе. Если ни одна запись не была изменена, возвращается нуль.

Программа, показанная в листинге 17.3, создает таблицу и добавляет в нее записи. Далее происходит удаление части записей, обновление записей и, наконец, удаление всей таблицы. Обратите внимание на способ вставки записей. Класс PreparedStatement позволяет описать запрос с параметрами, которые будут подставляться драйвером. В других СУБД поддерживаются предварительно скомпилированные запросы, которые анализируются один раз, а потом многократно выполняются. В MySQL вместо этого применяются запросы с параметрами.

В подготовленном запросе метасимвол `?` обозначает параметр, подставляемый драйвером перед выполнением запроса. Есть группа методов семейства `setX()`, позволяющих присваивать значения параметрам по номерам. Первый параметр имеет номер 1. Благодаря этому нет необходимости использовать различные управляющие символы, защищаемые одинарными кавычками. Если вызвать метод `setObject()`, то перед отправкой запроса серверу драйвер сохранит на диске образ объекта.

```
import java.sql.*;

public class CoreUpdate
{
   public static void main(String[] Args)
   {
      Connection c;

      // Загрузка драйвера.
      try
      {
```

```
      Class.forName("org.gjt.mm.mysql.Driver").newInstance();
   }
   catch (Exception e)
   {
      e.printStackTrace();
   }

   try
   {
      // Подключение к базе данных.
      c = DriverManager.getConnection(
          "jdbc:mysql://192.168.123.194/test?user=leon");

      // Создание инструкции.
      Statement stmt = c.createStatement();

      ResultSet rs;
      int rowsAffected;

      // Создание таблицы.
      stmt.executeUpdate(
          "CREATE TABLE testapi (" +
          "ID INT(11) NOT NULL AUTO_INCREMENT, " +
          "Name VARCHAR(64), " +
          "PRIMARY KEY(ID))");

      // Подготовка шаблона инструкции INSERT.
      String queryInsert = new String(
           "INSERT INTO testapi (Name) VALUES (?)");
      PreparedStatement ps = c.prepareStatement(queryInsert);

      // Выполнение инструкций INSERT.
      String[] names = {"Leon", "Vicky", "Carl", "Ricky",
                  "Nicki", "Jeff", "Bob", "Tina", "Joey"};

      for(int i=0; i < names.length; i++)
      {
         ps.setString(1, names[i]);
         rowsAffected = ps.executeUpdate();
         System.out.println(rowsAffected + " row inserted.");
      }

      // Удаление записей.
      rowsAffected = stmt.executeUpdate(
          "DELETE FROM testapi WHERE ID < 4");
      System.out.println(rowsAffected + " rows deleted.");

      // Обновление записей.
      rowsAffected = stmt.executeUpdate(
          "UPDATE testapi SET Name = 'None'");
      System.out.println(rowsAffected + " rows updated.");

      // Удаление таблицы.
      stmt.executeUpdate("DROP TABLE testapi");

      // Удаление инструкции.
```

```
            stmt.close();

            // Закрытие соединения.
            c.close();
        }
        catch (SQLException e)
        {
            e.printStackTrace();
        }
    }
}
```

VBSCRIPT И ODBC

В этой главе...

Подготовка программы
Извлечение данных
Изменение данных

Глава 18

Драйвер **MyODBC** позволяет подключаться к серверу MySQL по протоколу ODBC (Open Database **Connectivity).** Это особенно удобно пользователям Windows, поскольку для многих приложений Windows интерфейс ODBC — единственное средство взаимодействия с базами данных. В UNIX этот интерфейс используется только для переноса приложений Windows. Помимо драйвера MyODBC в UNIX необходим менеджер ODBC, лицензия на который стоит немало.

Технология ASP (Active Server **Pages),** как и ODBC, тоже была разработана компанией Microsoft. Она применяется сервером IIS (Internet Information Server) компании Microsoft для обслуживания Web-приложений. Сценарии ASP представляют собой HTML-файлы со специальными тэгами, которые интерпретируются сервером как исходный код. Несмотря на то что технология ASP поддерживает несколько языков сценариев, большинство разработчиков придерживаются языка VBScript.

В этой главе рассказывается о написании программы на языке VBScript, которая подключается к серверу MySQL по протоколу ODBC. Предполагается, что читатели имеют опыт написания сценариев ASP и работали с ODBC, по крайней мере на прикладном уровне.

Подготовка программы

Прежде чем начинать работать с базой данных, необходимо инсталлировать драйвер MyODBC. Это отдельный файл, доступный на Web-узле MySQL. Загрузите инсталляционную программу для своей платформы и установите драйвер. Можно создать системный источник данных (Data Source Name, DSN) или включить всю информацию о соединении непосредственно в сценарий.

Для запуска сценариев ASP необходим сервер IIS либо личный Web-сервер (Personal Web Server, PWS). Для пользователей Windows они доступны бесплатно. Сервер IIS работает в Windows NT/2000, а PWS - в Windows 98/98/ME. Оба сервера являются частью вспомогательных пакетов. Я работал с сервером PWS в Windows 98. Этот сервер есть на инсталляционных **компакт-дисках** Windows 98, но в случае **необ**ходимости его можно загрузить с **Web-узла** компании Microsoft. Для установки сервера

достаточно запустить инсталляционную программу и указать корневой каталог создаваемого Web-узла.

В листинге 18.1 показан минимальный сценарий, который подключается к серверу MySQL на локальномкомпьютере с использованием объектов ADO (ActiveX Data Objects).

```
<%@ LANGUAGE="VBSCRIPT" %>
<% Option Explicit %>
<html>
<head>
<title>minimal.asp</title>
</head>
<body>
<%
   dim connection

   ' Подключение к серверу.
   set connection=server.createobject("adodb.connection")
   connection.open "DRIVER=MySQL;SERVER=localhost;" & _
                   "USER=leon;DATABASE=test"

   ' Очистка.
   connection.close
   set connection=nothing
%>
</body>
</html>
```

Метод open () принимает в качестве аргумента строку подключения к ODBC. В данном случае в строке подключения указан не DSN, а непосредственно драйвер. Вообще-то драйвер MySQL — это библиотечный файл myodbc.dll, но имя драйвера — "MySQL", как указано в панели управления ODBC.

В строке подключения могут содержаться различные параметры, разделяемые точкой с запятой. Параметры, передаваемые драйверу MySQL, перечислены в табл. 18.1.

Параметр	*Описание*
DATABASE	Имя стандартной базы данных
OPTION	Различные опции (табл. 18.2)
PASSWORD, PWD	Пароль для подключения
PORT	Порт для подключения (по умолчанию — 3306)
SERVER	Доменное имя или IP-адрес сервера (по умолчанию — localhost)
SOCKET	Файл сокета или именованного канала
STMT	Запрос, выполняемый после установления соединения
USER, UID	Имя пользователя (по умолчанию — ODBC)

Параметр OPTION представляет собой битовое поле, в котором могут быть заданы 18 опций (табл. 18.2). При создании источника данных ODBC на панели управления эти опции отображаются в виде флажков. Чтобы активизировать несколько опций, просто сложите их значения.

Значение опции	*Описание*
1	Всегда сообщать формальную, а не фактическую размерность столбца
2	Сообщать о количестве просмотренных, а не модифицированных записей
4	Вести журнал отладки
8	Не ограничивать количество записей в таблице результатов запроса
16	Отменить выдачу каких бы то ни было приглашений
32	Имитировать ODBC 1.0
64	Игнорировать имена баз данных в табличных ссылках вида `mysql.user`
128	Использовать наборы записей менеджера ODBC
256	Не задавать идентификатор регионального стандарта
512	Дополнять поля типа CHAR до полной размерности столбца
1024	Возвращать полные имена столбцов в функции `SQLDescribeCol()`
2048	Сжимать данные, передаваемые между клиентом и сервером
4096	Разрешить пробелы между именем функции и открывающей скобкой
8192	Использовать именованные каналы
16384	Приводить столбцы типа BIGINT к типу INT
32768	Определять принадлежность таблиц по имени пользователя, а не базы данных
65536	Учитывать опции, заданные в конфигурационных файлах
131072	Проверять наличие ошибок

```
dim connection      Объект подключения к базе данных
dim query           Запрос
dim rs              Результаты запроса
dim num_fields      Число столбцов в таблице результатов
dim i               Счетчик цикла
```

```
    ' Выборка всех записей.
   do while not rs.eof
       response.write("<tr>" & vbcr)

        ' Отображение значений столбцов.
       for i = 0 to num_fields
          response.write("<td valign=""top"">")
          response.write(rs(i) & " ")
          response.write("</td>" & vbcr)
       next
       response.write("</tr>" & vbcr)

        ' Переход к следующей записи.
       rs.movenext
   loop

    ' Удаление результатов запроса.
   rs.close
   set rs=nothing

    ' Закрытие соединения.
   connection.close
   set  connection=nothing

   response.write("</table>" & vbcr)
%>
</body>
</html>
```

Обратите внимание на то, что в сценарии не делается явных предположений о количестве столбцов в таблице результатов запроса. Сценарий опрашивает свойство count и создает HTML-таблицу имен столбцов соответствующего размера. Имя столбца извлекается из свойства name.

Изменение данных

Запросы на вставку или обновление данных не возвращают наборы записей, но иногда требуется узнать, сколько записей было добавлено или изменено. Программа, представленная в листинге 18.3, использует объект command для лучшего контроля соединения. Программа создает таблицу и добавляет в нее записи. Далее происходит удаление части записей, обновление записей и, наконец, удаление всей таблицы.

```
<%@ LANGUAGE="VBSCRIPT" %>
<% Option Explicit %>
<html>
<head>
<title>update.asp</title>
</head>
<body>
<%
   dim connection    ' Объект подключения к базе данных
```

```
dim cmd            ' Объект взаимодействия с базой данных
dim query      ' Запрос
dim names          ' Массив имен для инструкции INSERT
dim n              ' Счетчик цикла
dim rows           ' Число измененных записей

 ' Подключение к серверу.
set connection=server.createobject("adodb.connection")
connection.open "DSN=myserver"

set cmd=server.createobject("adodb.command")
set cmd.activeconnection=connection

 ' Создание таблицы.
query = "CREATE TABLE testapi  ("
query = query & "ID INT(11) NOT NULL AUTO_INCREMENT,  "
query = query & "Name VARCHAR(64),  "
query = query & "PRIMARY KEY(ID)"
query = query & ")"
cmd.commandtext=query
cmd.execute

 ' Инструкция для вставки записей.
query = "INSERT INTO testapi  (Name) VALUES  (?)"
cmd.commandtext=query

 ' Создание входного параметра типаVARCHAR(64).
cmd.parameters.append cmd.createparameter("Name",200,1,64, "")

cmd.prepared=true

 ' Вставка записей.
names = array("Leon", "Vicky", "Carl", "Ricky", "Nicki", _
              "Jeff", "Bob", "Tina", "Joey")

for each n in names
   cmd("Name")=n
   cmd.execute rows
   response.write(rows & "  row inserted<br>" & vbcr)
next

 ' Очистка.
cmd.parameters.delete 0
cmd.prepared=false

 ' Удаление записей.
cmd.commandtext="DELETE FROM testapi  WHERE ID < 4"
cmd.execute rows
response.write(rows & "  rows deleted<br>" & vbcr)

 ' Обновление записей.
cmd.commandtext="UPDATE testapi SET Name = 'None'"
cmd.execute rows
response.write(rows & "  rows updated<br>" & vbcr)

 ' Удаление таблицы.
cmd.commandtext="DROP TABLE  testapi"
cmd.execute
```

```
    ' Закрытие соединения.
    set cmd=nothing
    connection.close
    set connection=nothing

    response.write("</table>" & vbcr)
%>
</body>
</html>
```

Сравните использование объекта command в листинге 18.3 с примером, показанным в листинге 18.2. Сразу после создания объект связывается с активным соединением. Текст запроса не передается непосредственно методу execute (), а заносится в свойство commandtext, после чего метод execute () вызывается без аргументов.

Обратите внимание на способ вставки записей. Массив names содержит имена, вставляемые в таблицу testapi. Чтобы не нужно было создавать отдельный запрос для каждой инструкции INSERT, объекту command передается параметризованный запрос. Вместо значений, подставляемых позднее, указан метасимвол ?. В других СУБД поддерживаются предварительно скомпилированные запросы, которые анализируются один раз, а потом многократно выполняются. В MySQL вместо этого применяются запросы с параметрами.

Перед выполнением запроса необходимо определить значение каждого параметра. В листинге 18.3 единственный параметр инструкции INSERT имеет тип VARCHAR(64). Метод createparameter() создает объект, содержащий описание параметра, а метод append() добавляет этот объект в коллекцию параметров (parameters) объекта command. Метод createparameter() принимает пять аргументов. Первый из них идентифицирует параметр запроса. Второй аргумент задает его SQL-тип. Это должна быть целочисленная константа. В частности, число 200 соответствует типу VARCHAR. Описание соответствующих констант можно найти в библиотеке MSDN компании Microsoft. Третий аргумент определяет, является ли параметр входным, выходным или двойственным. Подобно предыдущему аргументу, это целочисленная константа, описанная в документации по VBScript. Последние два аргумента задают длину параметра и его значение. В каждой операции вставки значение параметра будет разным, поэтому первоначально оно определено как пустая строка.

Прежде чем переходить к циклу вставки значений, нужно предупредить объект command о том, что он будет работать с подготовленной инструкцией. Тем самым сервер будет уведомлен о наличии у инструкции параметров. По окончании цикла вставки данный режим отключается.

В цикле for параметру инструкции INSERT по очереди присваиваются значения, хранящиеся в массиве names. Метод execute() помещает в переменную rows число записей, вставленных в таблицу. Последовательная вставка записей — не самое эффективное решение, учитывая, что в MySQL есть разновидность инструкции INSERT, позволяющая обрабатывать группу записей (см. главу 13, "Инструкции SQL"). Данный метод удобен тогда, когда требуется выполнять произвольные пользовательские запросы. Кроме того, он позволяет избежать проблем с управляющими символами.

PHP

В этой главе.

Подготовка программы
Извлечение данных
Изменение данных

Глава 19

MySQL – это наиболее популярная СУБД среди программистов, работающих на языке PHP. Обе эти системы извлекают преимущества из популярности друг друга. В частности, для MySQL выгодно растущее число сторонников писать Web-приложения на PHP, а для PHP выгодна высокая оперативность команды разработчиков MySQL. Монти **Видениус**, ответственный за сопровождение MySQL, входит в группу разработчиков ядра PHP и является членом зала славы PHP (*http://www.zend. com/zend/hof/*).

PHP — это серверный язык сценариев, предназначенный для встраивания кода в HTML-файлы с помощью специальных тэгов. Он поддерживает работу с MySQL, определяя собственные **функции-"оболочки"**, которые содержат вызовы функций библиотеки языка C (рассматривалась в главе 15, "Библиотека функций языка C"). Описание функций PHP можно найти на Web-узле *www.php.net*.

Подготовка программы

В PHP имеются собственные средства поддержки MySQL. В двоичные дистрибутивы входит модуль расширения MySQL, а в исходный дистрибутив включен каталог с исходными кодами, предназначенными для реализации интерфейса к этой СУБД. Если пакет PHP компилируется из исходных текстов, то внешняя библиотека функций MySQL не нужна: она входит в состав пакета.

PHP хорошо взаимодействует не только с сервером Apache, но и с сервером IIS в Windows. В любом случае Web-сервер должен быть настроен таким образом, чтобы файлы, имеющие расширение `.php`, были ассоциированы с модулем PHP.

Подключиться к серверу MySQL из **PHP-сценария** несложно. Достаточно вызвать функцию `mysql_connect()` или `mysql_pconnect()` с соответствующими аргументами. Обе функции возвращают идентификатор соединения, требуемый для боль-

шинства остальных функций. Функция `mysql_pconnect()` создает постоянное соединение. Оно будет сохранено после завершения сценария на случай повторного использования. Такие соединения закрывать не нужно.

Первый аргумент функции mysql_pconnect `()` идентифицирует сервер по имени или адресу. Через двоеточие может быть задан номер порта, если он отличается от стандартного (3306). Аналогичным образом задается **сокет.** Другими двумя аргументами являются имя пользователя и пароль.

В листинге 19.1 показан сценарий, который устанавливает соединение с сервером MySQL на узле `localhost` и задает базу данных, используемую по умолчанию. В сценарии не выполняется проверка ошибок, но модуль **PHP** выдаст предупреждение в случае неудачного завершения одной из функций.

```
<html>
<head>
<title>minimal.php</title>
</head>
<body>
<?
    // Открытие постоянного соединения.
    $dblink = mysql_pconnect("localhost", "leon", "");

    // Выбор базы данных 'test'.
   mysql_select_db("test", $dbLink);
?>
</body>
</html>
```

Извлечение данных

После подключения к серверу баз данных можно посылать ему запросы с помощью функции `mysql_query()`. Эта функция принимает в качестве аргумента текст SQL-инструкции плюс идентификатор соединения, полученный от функции `mysql_` connect `()` или mysql_pconnect `()`, и возвращает идентификатор результирующего набора записей. В случае ошибки возвращается нуль.

Функция mysql_result `()` извлекает значение одной ячейки, но гораздо удобнее пользоваться функцией `mysql_fetch_row()`, `mysql_fetch_array()` или `mysql_fetch_object()`. Все они возвращают следующую запись из указанного набора, что позволяет эффективно применять их в цикле while.

Рассмотрим пример, показанный в листинге 19.2. Этотсценарий извлекает записи из таблицы user, оформляя результаты запроса в виде HTML-таблицы. Обратите внимание на то, что сценарий пытается подключиться к серверу от имени пользователя root с явно неправильным паролем. Если будете экспериментировать с этим сценарием, подставьте корректные значения имени пользователя и пароля.

```
<html>
<head>
<title>select.php</title>
</head>
<body>
<?
   // Открытие постоянного соединения.
   if(!($dbLink = mysql_pconnect("localhost", "root", "pass")))
   {
      print("Could not connect to server!<br>\n");
      print(mysql_errno() . ": " . mysql_error() . "<br>\n");
      exit();
   }

   // Выбор базы данных 'mysql'.
   if(!mysql_select_db("mysql", $dbLink))
   {
      print("Could not select database!<br>\n");
      print(mysql_errno() . ": " . mysql_error() . "<br>\n");
      exit();
   }

   // Выполнение запроса.
   $Query - "SELECT User, Host FROM user ORDER BY 1,2";
   if(!($dbResult = mysql_query($Query, $dbLink)))
   {
      print("Could not execute query!<br>\n");
      print(mysql_errno() . ": " . mysql_error() . "<br>\n");
      exit();
   }

   print("<table border=\"1\" cellspacing=\"0\">\n");

   // Получение информации о столбцах.
   print("<tr>\n");
   while($field = mysql_fetch_field($dbResult))
   {
      print("<th>$field->name</th>\n");
   }
   print("</tr>\n");

   // Извлечение записей.
   while($row = mysql_fetch_row($dbResult))
   {
      print("<tr>\n");
      foreach($row as $r)
      {
         print("<td>$r </td>\n");
      }
      print("</tr>\n");
   }
   print("</table>\n");
?>
</body>
</html>
```

В этом примере ведется тщательный контроль ошибок. Многие PHP-функции в случае ошибки возвращают FALSE. Внутри инструкций if отображается сообщение об ошибке, за которым следуют номер ошибки и ее описание, полученные от сервера MySQL.

Прежде чем извлекать значения столбцов, сценарий создает заголовок таблицы с именами столбцов. Функция `mysql_fetch_field()` возвращает объект, содержащий описание полей указанного набора записей. В данном примере используется только свойство name этого объекта, хотя в нем хранится вся информация о столбце, включая его тип и размерность. При очередном обращении функция `mysql_fetch_field()` возвращает описание следующего столбца, поэтому она вызывается в цикле.

Функция `mysql_fetch_row()` возвращает массив значений столбцов, индексация которых начинается с нуля. Поскольку в данном примере записи отображаются целиком (в цикле `foreach`), нет необходимости адресовать отдельные столбцы.

Изменение данных

Запросы на вставку или обновление данных не возвращают наборы записей. Они тоже выполняются с помощью функции mysql_query(), но вызывать функцию `mysql_fetch_row()` нет необходимости. Если требуется узнать число добавленных или измененных записей, воспользуйтесь функцией `mysql_affected_rows()` (листинг 19.3).

Представленный ниже сценарий создает таблицу и добавляет в нее записи. Далее происходит удаление части записей, обновление записей и, наконец, удаление всей таблицы. Обратите внимание на способ вставки записей. В массиве name содержатся девять имен, вставляемых в базу данных одной инструкцией INSERT. Сценарий составляет запрос, поочередно добавляя к нему элементы массива, отформатированные в соответствии с синтаксисом многострочной инструкции INSERT. Функция `addslashes()` вставляет обратную косую черту перед каждым символом, имеющим специальное назначение в SQL, например перед одинарной кавычкой. С помощью функции `substr()` из текста запроса удаляется хвостовая запятая.

```
<html>
<head>
<title>select.php</title>
</head>
<body>
<?
   // Открытие постоянного соединения.
   if(!($dbLink = mysql_pconnect("localhost", "leon", "")))
   {
      print("Could not connect to server!<br>\n");
      print(mysql_errno() . ": " . mysql_error() . "<br>\n");
      exit();
   }

   // Выбор базы данных 'test'.
   if(!mysql_select_db("test", $dbLink))
   {
```

```
   print("Could not select database!<br>\n");
   print(mysql_errno() . ": " . mysql_error() . "<br>\n");
   exit();
}

// Создание таблицы.
$Query = "CREATE TABLE testapi (" .
    "ID INT(11) NOT NULL AUTO_INCREMENT, " .
    "Name VARCHAR(64), " .
    "PRIMARY KEY(ID))";
if(!($dbResult = mysql_query($Query, $dbLink)))
{
   print("Could not create table!<br>\n");
   print(mysql_errno() . ": " . mysql_error() . "<br>\n");
   exit();
}

// Вставка записей.
$Query = "INSERT INTO testapi (Name) VALUES ";
$name = array("Leon", "Vicky", "Carl", "Ricky", "Nicki",
              "Jeff", "Bob", "Tina", "Joey");
foreach($name as $n)
{
   $Query .= "('" . addslashes($n) . "'),";
}
$Query = substr($Query, 0, -1);

if(!($dbResult = mysql_query($Query, $dbLink)))
{
   print("Could not insert rows!<br>\n");
   print(mysql_errno() . ": " . mysql_error() . "<br>\n");
}

print(mysql_affected_rows($dbLink) . " rows inserted<br>\n");

// Удаление записей.
$Query = "DELETE FROM testapi WHERE ID < 4";
if(!($dbResult = mysql_query($Query, $dbLink)))
{
   print("Could not delete rows!<br>\n");
   print(mysql_errno() . ": " . mysql_error() . "<br>\n");
}

print(mysql_affected_rows($dbLink) . " rows deleted<br>\n");

// Обновление записей.
$Query = "UPDATE testapi SET Name = 'None'";
if(!($dbResult = mysql_query($Query, $dbLink)))
{
   print("Could not update rows!<br>\n");
   print(mysql_errno() . ": " . mysql_error() . "<br>\n");
}

print(mysql_affected_rows($dbLink) . " rows updated<br>\n");

// Удаление таблицы.
```

```
    $Query = "DROP TABLE testapi";
    if(!($dbResult = mysql_query($Query, $dbLink)))
    {
       print("Could not drop table!<br>\n");
       print(mysql_errno() . ": " . mysql_error() . "<br>\n");
    }
?>
</body>
</html>
```

PERL

В этой главе…

Подготовка программы
Извлечение данных
Изменение данных

Глава 20

Трудно представить UNIX-систему без языка Perl. Это один из основных языков сценариев, завоевавший Internet. Он применяется при системном администрировании, в научных расчетах и при создании **Web-приложений.** Некоторые сценарии MySQL написаны именно на Perl. Сценарии Perl прекрасно работают в Windows и в других операционных системах.

В этой главе предполагается, что читатели знакомы с данным языком и умеют писать собственные сценарии. Будет рассмотрен интерфейс DBI, но лишь в общих чертах. В документации к MySQL есть описание драйвера `DBD::mysql` и методов DBI. Для получения дополнительной информации можно также воспользоваться командной `perldoc`.

Подготовка программы

Поддержка MySQL в Perl реализована в интерфейсе DBI (Database Interface). Этот интерфейс определяет унифицированные методы доступа ко многим источникам данных, включая реляционные СУБД. Помимо модуля DBI необходим драйвер DBD (Database Driver) для конкретной СУБД. В случае MySQL и mSQL драйвер называется `DBD::mysql`. Соответствующие файлы можно загрузить на Web-узле MySQL или из архивов CPAN (Comprehensive Perl Archive Network) по адресу *http://www.perl.com/CPAN/*. Посетите также Web-узел, посвященный интерфейсу DBI (*http://dbi.symbolstone.org*).

Процесс инсталляции модуля Perl включает в себя запуск Perl-сценария перед компиляцией исходного кода. В остальном все делается так же, как и в случае любой другой программы. Последовательность команд будет такой:

```
perl Makefile.PL
make
make test
make install
```

После инсталляции модуля DBI и драйвера MySQL подключение к серверу **осуществляется** с помощью **одного-единственного** метода: `connect()` (листинг 20.1). Он возвращает дескриптор базы данных. Первый аргумент метода определяет источник данных. Компоненты этой строки разделяются двоеточиями. Первый компонент **всегда** равен **"DBI"**. Второй компонент идентифицирует драйвер, в данном случае **"mysql"**. Третий компонент — это имя базы данных. Дополнительно можно указать **адрес** узла и порт. Остальные два аргумента метода `connect()` задают имя пользователя и порт. По окончании сеанса необходимо вызвать метод `disconnect()`.

```
#!/usr/bin/perl

# Подключение модуля DBI.
use DBI;

I Подключение к базе данных.
$dbh = DBI->connect('DBI:mysql:test', 'leon', 'pass');

# Отключение.
$dbh->disconnect;
```

Извлечение данных

После подключения к серверу баз данных можно посылать ему запросы с помощью методов `prepare()` и `execute()`. Первый из них подготавливает инструкцию к выполнению и возвращает ее дескриптор. Второй метод передает инструкцию серверу и просит его подготовить результаты запроса.

Рассмотрим пример, показанный в листинге 20.2. Этот сценарий извлекает записи из таблицы user и отображает их на экране. Обратите внимание на то, что сценарий пытается подключиться к серверу от имени пользователя root с явно неправильным паролем. Если будете экспериментировать со сценарием, подставьте корректные значения имени пользователя и пароля.

```
#!/usr/bin/perl

# Подключение модуля DBI.
use DBI;

§ Подключение к базе данных.
my $dbh = DBI->connect("DBI:mysql:test:localhost", 'root',
                       'pass');

tt Подготовка запроса.
my $query = "SELECT User, Host FROM user ORDER BY 1,2";
my $sth = $dbh->prepare($query)
    or die "Can't prepare $query: " . $dbh->errstr . "\n";
```

```
# Выполнение запроса.
$sth->execute
    or die "Can't execute $query: " . $dbh->errstr . "\n";

i Извлечение записей.
my $row;
while(@row = $sth->fetchrow_array)
{
    my $i;
    for $i (0..($sth->{NUM_OF_FIELDS}-1))
    {
       print "[" . $row[$i] . "]";
    }

    print("\n");
}

i Удаление инструкции.
$sth->finish;

ttОтключение.
$dbh->disconnect;
```

В этом примере ведется тщательный контроль ошибок. Многие Perl-функции в случае ошибки возвращают нуль. Ветвь or die интерпретируется только тогда, когда первая часть конструкции равна нулю. Обратите внимание на сообщение об ошибке, в котором указывается, какой запрос потерпел неудачу, и приводится сообщение, полученное от драйвера.

Метод `fetchrow_array()` возвращает массив значений столбцов, индексация которых начинается с нуля. Данный сценарий достаточно универсален и позволяет обрабатывать результаты произвольной инструкции SELECT. Не делается никаких предположений о том, сколько столбцов имеется в таблице результатов запроса. Это значение определяется с помощью свойства NUM_OF_FIELDS объекта инструкции. Метод `fetchrow_array()` возвращает следующую запись из набора, пока не будет достигнут конец таблицы.

В конце сценария вызывается метод `finish()`, который освобождает память, занимаемую инструкцией.

Изменение данных

Если требуется выполнить инструкцию, которая не возвращает набор записей (например, CREATE STATEMENT), воспользуйтесь методом `do()` объекта базы данных. Этот метод возвращает число измененных записей. Для запросов, которые выполнились успешно, но не меняли никаких записей, возвращается специальное значение `0E0`.

Представленный ниже сценарий создает таблицу и добавляет в нее записи. Далее происходит удаление части записей, обновление записей и, наконец, удаление всей таблицы. Обратите внимание на способ вставки записей. В массиве name содержатся

девять имен, вставляемых в базу данных. В цикле for вызывается метод execute(), **в** котором **параметру инструкции** INSERT (обозначается символом ?) по очереди **присваиваются значения, хранящиеся в массиве. Учитывая,** что **в MySQL** поддерживается многострочная **инструкция** INSERT, **это не самый эффективный** метод **вставки имен,** но он удобен, когда **требуется** выполнять произвольные пользовательские запросы.

```
#!/usr/bin/perl

# Подключение модуля DBI.
use DBI;

my $query;
my $rows;

# Подключение к базе данных.
my $dbh = DBI->connect("DBI:mysql:test:localhost", 'leon', '');

# Создание таблицы.
$query = "CREATE TABLE testapi (" .
    "ID INT(11) NOT NULL AUTO_INCREMENT, " .
    "Name VARCHAR(64), " .
    "PRIMARY KEY(ID)" .
    ")";
$dbh->do($query)
    or die "Can't execute $query: " . $dbh->errstr . "\n";

# Вставка записей.
$query = "INSERT INTO testapi (Name) VALUES (?)";

my @name = ('Leon', 'Vicky', 'Carl', 'Ricky', 'Nicki',
            'Jeff', 'Bob', 'Tina', 'Joey');

my $sth = $dbh->prepare($query)
    or die "Can't prepare $query: " . $dbh->errstr . "\n";

my $n;
foreach $n (@name)
{
    $sth->execute($n)
        or die "Can't execute $query: " . $dbh->errstr . "\n";
    print($sth->rows . " row inserted\n");
}

# Удаление записей.
$query="DELETE FROM testapi WHERE ID < 4";

$rows = $dbh->do($query)
    or die "Can't execute $query: " . $dbh->errstr . "\n";

print($rows . " rows deleted\n");

tt Обновление записей.
```

```
$query="UPDATE testapi SET Name = 'None'";

$rows = $dbh->do($query)
    or die "Can't execute $query: " . $dbh->errstr . "\n";

print($rows . " rows updated\n");

I Удаление таблицы.
$query="DROP TABLE testapi";
$dbh->do($query)
    or die "Can't execute $query: " . $dbh->errstr . "\n";

# Отключение.
$dbh->disconnect;
```

PYTHON

В этой главе.

- Подготовка программы
- Извлечение данных
- Изменение данных

Глава 21

Гвидо ван Россум (Guido van Rossum) назвал свой объектно-ориентированный язык сценариев Python в честь известного британского комедийного телесериала "Monty Python's Flying Circus". Сценарии Python выполняются в UNIX, Windows и многих других операционных системах. Язык Python ценится многими за простоту изучения.

В этой главе рассматривается MySQLdb – библиотека языка Python, предназначенная для взаимодействия с серверами MySQL. Предполагается, что читатели умеют писать программы на этом языке. Тем, кому он в диковинку, советуем посетить Web-узел *www.python.org*. Там есть много ресурсов, посвященных изучению языка.

Подготовка программы

Авторы проекта Python разработали спецификацию DBI-API 2.0 для д райверов баз данных. Благодаря этому взаимодействие с базами данных осуществляется унифицированным образом. Драйвер MySQL реализован Энди Дастманом (Andy Dustman) и доступен по адресу *http://dustman.net/andy/python/MySQLdb* либо *http://sourceforge.net/projects/mysql-python*.Там можно загрузить исходные коды и модули RPM для Linux. Если сценарии Python предполагается запускать в Windows, то Герхард Херинг (Gerhard Häring) предлагает скомпилированную версию библиотеки MySQLdb на своем узле по адресу *http://www.cs.fhm.edu/~ifw00065/*.

Подробное описание процедуры инсталляции выходит за рамки данной книги. В архивах MySQLdb содержатся сведения по компиляции и инсталляции модуля. Сценарий `setup.py` автоматически управляет всем процессом.

Подключиться к серверу MySQL из сценария Python несложно. Достаточно импортировать модуль MySQLdb и вызвать метод Connect(). Параметры метода перечислены в табл. 21.1. Он возвращает объект Connection, который можно использовать для получения объекта Cursor.

Параметр	*Описание*
Db	Стандартная база данных
Host	Имя или IP-адрес узла
Passwd	Пароль
Port	Порт TCP/IP
unix_socket	Путевое имя сокета
user	Имя пользователя

В листинге 21.1 показан минимальный сценарий подключения к серверу MySQL, расположенному на узле localhost. Имя базы данных – test. В сценарии создается указатель набора записей. Вообще-то в MySQL не разрешен непосредственный доступ к указателям, но в библиотеке MySQLdb эти функции имитируются.

```
#!/usr/bin/python

import  MySQLdb

# Подключение к серверу баз данных.
myDB = MySQLdb.Connect(host='localhost', user='leon', passwd='',
                       db='test')

# Создание указателя набора записей.
cursor = myDB.cursor()

# Отключение.
myDB.close()
```

Извлечение данных

После подключения к серверу баз данных можно посылать ему запросы с помощью методов execute() и executemany(). Оба они вкачестве первого аргумента принимают текст запроса.

Инструкция SELECT и ряд других инструкций, в частности SHOW PROCESSLIST, возвращают результаты запроса в виде набора записей. Метод fetchone() возвращает следующую запись из набора, а метод fetchall() возвращает весь набор в виде массива.

В листинге 21.2 показан сценарий, который извлекает данные из таблицы user и отображает их в виде таблицы. Обратите внимание на то, что сценарий пытается подключиться к серверу от имени пользователя root с явно неправильным паролем. Если будете экспериментировать с этим сценарием, подставьте корректные значения имени пользователя и пароля.

```
#!/usr/bin/python

import MySQLdb

# Подключение к серверу баз данных.
myDB = MySQLdb.Connect(host='localhost', user='root',
                       passwd='pass', db='mysql')

# Создание указателя набора записей.
cursor = myDB.cursor()

# Выполнение запроса.
cursor.execute("SELECT User, Host FROM user ORDER BY 1,2")

# Отображение заголовка таблицы.
for field in cursor.description:
    width = field[2]
    print '%-*s' % (width, field[0][0:width]),
print ""
fieldLen = ()
for field in cursor.description:
    line = ""
    for i in range (0,field[2]): line += "-"
    print line,
    fieldLen += (field[2],)
print ""

# Отображение результатов запроса.
resultSet = cursor.fetchall()
for cursorRecord in resultSet:
    f=0
    for cursorField in cursorRecord:
        width = fieldLen[f]
        print '%-*s' % (width, cursorField[0:width]),
        f += 1
    print ""

# Закрытие соединения.
myDB.close()
```

Перед выборкой значений столбцов сценарий создает строку заголовка, включая в нее имена столбцов. Массив описаний столбцов хранится в свойстве `description`. Каждое описание включает семь атрибутов: имя, тип, отображаемая **размерность**, внутренняя размерность, точность, степень масштабирования для десятичных **столбцов** и допустимость значений NULL. В листинге 21.2 результаты форматируются на основании отображаемой размерности.

Обратите внимание на спецификацию * в команде `print`. Она позволяет задавать размерность на этапе выполнения сценария.

Изменение данных

Запросы на вставку или обновление данных не возвращают наборы записей. Они тоже выполняются с помощью метода execute(), но извлекать записи нет необходимости. Если требуется узнать число добавленных или измененных записей, воспользуйтесь свойством rowcount объекта cursor.

Сценарий, показанный в листинге 21.3, создает таблицу и добавляет в нее записи. Далее происходит удаление части записей, обновление записей и, наконец, удаление всей таблицы. Обратите внимание на способ вставки записей. В массиве names содержатся девять имен, добавляемых в таблицу путем **одного-единственного** вызова метода `executemany()`. Вместо значения имени в тексте запроса стоит спецификация `%s`, которая заменяется реальным значением при вызове метода executemany(). Вторым аргументом этого метода является массив списков. Каждый список **соответствует** одной добавляемой записи, а элементы **списка** — параметрам запроса. В листинге 21.3 у инструкции INSERT один параметр, поэтому кажется, будто в определении массива names стоит слишком много скобок и запятых. На самом деле хвостовые запятые нужны для правильного определения типа массива.

```
#!/usr/bin/python

import MySQLdb

# Подключение к серверу баз данных.
myDB = MySQLdb.Connect(host='localhost', user='leon',
                       passwd='pass', db='test')

# Создание указателя набора записей.
cursor = myDB.cursor()

# Создание таблицы.
query = \
    "CREATE TABLE testapi (" + \
    "ID INT(11) NOT NULL AUTO_INCREMENT, " + \
    "Name VARCHAR(64), " + \
    "PRIMARY KEY(ID)" + \
    ")"
cursor.execute(query)

# Вставка записей.
query = "INSERT INTO testapi (Name) VALUES (%s)"
names = (('Leon',), ('Vicky',), ('Carl',), ('Ricky',),
         ('Nicki',), ('Jeff',), ('Bob',), ('Tina',), ('Joey',),)
cursor.executemany(query, names)
print str(cursor.rowcount) + " rows inserted"

# Удаление записей.
cursor.execute("DELETE FROM testapi WHERE ID < 4")
print str(cursor.rowcount) + " rows deleted"

# Обновление записей.
```

```
cursor.execute("UPDATE testapi SET Name = 'None'")
print str(cursor.rowcount) + " rows updated"

# Удаление таблицы.
cursor.execute("DROP TABLE testapi")

# Отключение.
myDB.close()
```

БИБЛИОТЕКА MYSQL++

В этой главе.

- Подготовка программы
- Извлечение данных
- Изменение данных

Глава 22

Группа разработчиков MySQL выпустила **MySQL++** – официальную библиотеку классов языка C++, предназначенных для взаимодействия с MySQL. Изначально написанная Кевином **Аткинсоном** (не родственник), она в настоящее время сопровождается уже упоминавшимся **Синишей Миливоевичем**. Библиотека **MySQL++** работает с большинством компиляторов **C++**, включая GNU C++ и Visual C++компании Microsoft.

В этой главе предполагается, что читатели имеют опыт написания и компиляции программ C++. БиблиотекаMySQL++ описывается лишь в общих чертах. Узнать о ней подробнее и загрузить исходный код можно по адресу *http://mysql.com/downloads/api-mysql++.html*.

Подготовка программы

При использовании библиотеки MySQL++ необходимо включить в программу файл `sqlplus.hh`. Стандартный make-файл инсталлирует файлы заголовков этой библиотеки в каталог `/usr/local/include`, в отличие от файлов клиентской библиотеки MySQL, размещаемых в каталоге `/usr/local/include/mysql`. Если эту установку нужно изменить, отредактируйте `make`-файл или сконфигурируйте компилятор соответствующим образом.

Библиотека MySQL++ использует функции библиотеки языка C, поэтому на этапе компиляции нужно подключить два файла: `sqlplus` и `mysqlclient`. С файлом `sqlplus` такая же ситуация, как и с файлами заголовков. По умолчанию он находится в каталоге `/usr/local/bin`, а не `/usr/local/lib/mysql`. Ниже показан пример компиляции тестового клиента.

```
c++ minimal.cc -o minimal -I/usr/local/include/mysql \
-L/usr/local/lib/mysql -L/usr/local/lib \
-lmysqlclient -lsqlplus
```

Естественно, в более сложных случаях необходимо создавать make-файл.

Подключиться к серверу MySQL средствами библиотеки MySQL++ несложно: достаточно создать объект класса Connection. У этого класса четыре конструктора:

```
Connection ();
Connection (bool te) ;
Connection (const char *db, const char *host = "",
            const char *user = "", const char *passwd = "",
            bool te = true);
Connection (const char *db, const char *host, const char *user,
            const char *passwd, uint port, my_bool compress = 0,
            unsigned int connect_timeout = 60, bool te = true,
            cchar *socket_name = "", unsigned int client_flag=0);
```

В основном аргументы конструктора соответствуют аргументам функции mysql_real_connect() библиотеки языка C. Аргумент te определяет, будут ли генерироваться исключения.

В листинге 22.1 показан текст минимального клиента. Эта программа подключается к базе данных test на узле localhost. Код очистки не нужен, так как деструктор класса Connectionсамостоятельно закрывает соединение.

```
ttinclude <iostream>
#include <iomanip>
ttinclude <sqlplus.hh>

int main(int argc, char *argv[])
{
   // Подключение к серверу.
   Connection con("test", "localhost", "leon", "");

   return 0;
}
```

Извлечение данных

Подключившись к серверу баз данных, программа должна создать объект класса Query, с помощью которого будут посылаться запросы. Этот класс является потомком класса iostream, но емуразрешается посылатьлишь входные данные, а оператор >> не поддерживается. Чтобы направить запрос серверу, необходимо записать его в поток с помощью оператора << и вызвать метод Query: : store() либо Query:: execute(). Первый из них возвращает таблицу результатов запроса.

В листинге 22.2 показана простая программа, которая извлекаетданные из таблицы user. Обратите внимание на то, что программа пытается подключиться к серверу от имени пользователя root с явно неправильным паролем. Если будете экспериментировать с этой программой, подставьте корректные значения имени пользователя и пароля.

Обратите также внимание на использование **итератора класса** Result **в цикле** `for`. **На** каждом **шаге цикла создается** объект **класса** Row, который **ведет себя как массив** строк. **Метод** `Row::size()` возвращает число полей **записи.**

```
#include <iostream>
#include <iomanip>
#include <sqlplus.hh>

int main(int argc, char *argv[])
{
   try
   {
      // Подключение к серверу.
      Connection con(use_exceptions);
      con.connect("mysql", "localhost", "root", "passw");

      // Создание объекта запроса.
      Query query = con.query();

      // Отправка запроса.
      query << "SELECT User, Host FROM user ORDER BY 1,2";

      // Получение результатов запроса.
      Result res = query.store();

      // Отображение записей.
      Row row;
      cout.setf(ios::left);
      Result::iterator i;

      for (i = res.begin(); i != res.end(); i++)
      {
         row = *i;
         for (unsigned int j=0; j < row.size(); j++)
         {
            cout << "[" << row[j] << "]";
         }
         cout << endl;
      }

      return 0;
   }
   catch (BadQuery er)
   {
      cerr << "Error: " << er.error << endl;
      return -1;
   }
}
```

Изменение данных

Метод `Query::execute()` предназначен для выполнения **запросов**, которые не возвращают результаты в виде наборов записей. С помощью метода Connection::affected_rows() можно узнать число добавленных, удаленных или обновленных записей (листинг 22.3).

```
#include <iostream>
#include <iomanip>
#include <sqlplus.hh>

int main(int argc, char *argv[])
{
   try
   {
      //
      // Подключение к серверу.
      //
      Connection con(use_exceptions);
      con.connect("test", "localhost", "leon", "");

      // Создание объекта запроса.
      Query query = con.query();

      //
      // Создание таблицы.
      //
      query << "CREATE TABLE IF NOT EXISTS testapi ("
            << "ID.INT(11) NOT NULLAUTO_INCREMENT, "
            << "Name VARCHAR(64), "
            << "PRIMARY KEY(ID)"
            << ")";
      query.execute(RESET_QUERY);

      //
      // Вставка записей.
      //
      const char *names[] = {
          "Leon", "Vicky", "Carl", "Ricky", "Nicki",
          "Jeff", "Bob", "Tina", "Joey"
          };
      uint num_rows = sizeof(names)/sizeof(char *);

      // Подготовка запроса.
      query << "INSERT INTO %0 (Name) VALUES (%1q)";
      query.parse();

      for(int i=0; i<num_rows; i++)
      {
         // Выполнение запроса.
         query.execute("testapi", names[i]);
         cout << con.affected rows()
```

```
            << " row inserted" << endl;
      }

      query.reset();

      //
      // Удаление записей.
      //
      query << "DELETE FROM testapi WHERE ID < 4";
      query.execute(RESET_QUERY);
      cout << con.affected_rows()
           << " rows deleted" << endl;

      //
      // Обновление записей.
      //
      query << "UPDATE testapi SET Name = 'None'";
      query.execute(RESET_QUERY);
      cout << con.affected_rows()
           << " rows updated" << endl;

      //
      // Удаление таблицы.
      //
      query << "DROP TABLE testapi";
      query.execute(RESET_QUERY);

      return 0;
   }
   catch (BadQuery er)
   {
      cerr << "Query Error: " << er.error << endl;
      return -1;
   }
}
```

Программа, представленная в листинге 22.3, создает таблицу и добавляет в нее записи. Далее происходит удаление части записей, обновление записей и, наконец, удаление всей таблицы. Обратите внимание на способ вставки записей. Инструкция INSERT, переданная объекту query, содержит коды **параметров**, начинающиеся с символа `%`. Стоящий затем номер определяет порядок аргументов метода `execute()`. В данном примере методу требуется узнать имя таблицы и значение столбца. Символ `q` говорит о том, что значение берется в кавычки.

Для параметризованных запросов должен вызываться метод Query: `:parse()`, позволяющий правильно пометить параметры. При выполнении запроса нужно указывать значение каждого параметра. Можно также задать стандартные значения параметров, чтобы метод `execute()` их не требовал. Для этого предназначен **строковый** массив `Query::def`.

Часть IV

СЛОЖНЫЕ ТЕМЫ

В этой части рассматриваются сложные темы, которые могут быть неинтересны рядовым пользователям MySQL. Представленный здесь материал предназначен, скорее, администраторам баз данных, стремящимся повысить производительность своей СУБД или расширить ее функциональные возможности.

В главе 23, "Администрирование баз данных", читатели узнают о том, какая ответственность лежит на администраторе баз данных. В главе 24, "Физическое хранение данных", рассказывается о принципах использования физических ресурсов в MySQL. В главе 25, "Устранение последствий катастроф", описываются стратегии предотвращения катастроф и устранения их последствий. Сюда входит восстановление поврежденных таблиц и создание резервных копий. Глава 26, "Оптимизация", посвящена вопросам оптимизации баз данных и запросов. В главе 27, "Безопасность", рассматриваются вопросы обеспечения безопасности баз данных.

В главе 28, "Перенос данных в разные СУБД", рассказывается о том, зачем может понадобиться перейти из другой СУБД в MySQL или из MySQL — в другую СУБД и какие нюансы необходимо при этом учесть.

В главе 29, "Распределенные базы данных", рассматриваются концепции распределенных баз данных. В частности, описываются принципы синхронизации и репликации таких баз данных.

В главе 30, "Работа с объектами", описывается применение объектно-ориентированных методик при работе с базами данных. Приводится пример работы с объектами баз данных на PHP.

Глава 31, "Расширение возможностей MySQL", посвящена расширению функциональных возможностей MySQL. Сюда входит добавление новых наборов символов, функций и процедур. Рассматривается также библиотека функций отладки MySQL.

АДМИНИСТРИРОВАНИЕ БАЗ ДАННЫХ

В этой главе...

Ответственность
Обеспечение доступности данных
Поддержание целостности данных
Подготовка к катастрофе
Поддержка пользователей
Разработка и внедрение стандартов

Глава 23

В этой главе рассматриваются задачи, выполняемые администратором баз данных. Отдельный человек или целая команда принимает на себя ответственность за непрерывное функционирование сервера MySQL и обязуется непрерывно выполнять анализ производительности, устранять проблемы и настраивать конфигурацию.

Ответственность

В худшем случае администратор баз данных выступает в качестве пожарного. Он едва успевает погасить один пожар, как начинается следующий. Это не самый приятный опыт как для самого администратора, так и для пользователей. В идеале администратор должен предвидеть возможные проблемы и заранее спланировать работу системы так, чтобы она оставалась максимально "здоровой".

Перечислим пять основных обязанностей администратора:

- обеспечение доступности данных;
- поддержание целостности данных;
- подготовка к катастрофе;
- поддержка пользователей;
- разработка и внедрение стандартов.

Обеспечение доступности данных

Базы данных предназначены для накопления и обработки информации. Администратор обязан прилагать усилия, чтобы гарантировать доступность этой информации. Пользователи предпочитают немедленно получать доступ к интересующим их сведениям. Сервер базданных должен функционировать в то время, когда у пользова-

телей может возникнуть необходимость обратиться к нему. Под него должно быть выделено оборудование соответствующего уровня.

Сервер MySQL работает в виде демона, обычно круглосуточно. Сценарий `safe_mysqld`, входящий в состав дистрибутива, отслеживает те редкие случаи, когда демон зависает, и перезапускает его. Время от времени все же необходимо приостанавливать сервер и отменять все соединения, чтобы можно было выполнить плановые проверки и восстановить целостность базы данных. Это лучше всего делать в периоды отсутствия активности со стороны пользователей, как правило, ночью. В случае базы данных Web-узла выбор сделать не так-то легко. Здесь рекомендуется проанализировать журнальные файлы Web-сервера и определить часы наименьшей активности, когда количество подключений минимально.

Несмотря на все усилия по оптимизации и настройке исполняемых файлов MySQL и конфигурированию сервера, основной вклад в производительность все же вносит оборудование. Чем оперативнее сервер выполняет запросы, тем более быстродействующим он кажется пользователям. Естественно, большую роль играет устройство хранения данных. Быстрые жесткие диски существенно влияют на производительность сервера. Приведем лишь несколько советов, касающихся запуска MySQL на персональном компьютере.

Не полагайтесь лишь на скорость передачи данных, которой обладает жесткий диск. Производительность SCSI-дисков будет выше, чем IDE-дисков, поскольку в технологии SCSI поддерживаются одновременные операции чтения. Еще больше повышает производительность технология RAID (Redundant Array of Independent Disks — матрица независимых дисковых накопителей с избыточностью), в которой группа дисков представляется единым устройством. Программа MySQL поддерживает эту технологию, позволяя распределять данные между несколькими файлами, которые могут находиться на разных дисках.

Администрирование баз данных — это работа для талантливого специалиста, обладающего специальными знаниями. Зато язык SQL настолько прост, что вполне может быть освоен большинством пользователей. Администраторы только выигрывают от самообучаемости пользователей. Потратьте время на то, чтобы научить пользователей работать с утилитами MySQL илидругими приложениями, допускающими ввод произвольных запросов. Когда пользователи смогут самостоятельно создавать отчеты, администратор освободится для решения более важных задач.

Поддержание целостности данных

Зачастую данные— это важнейший актив организации. Попробуйте оценить стоимость потерянной информации! Мебель, оборудование и даже служащих можно заменить, а данные — далеко не всегда. Если потерять информацию о заказах, клиентам нельзя будет выписать счета. Если потерять результаты научных исследований, не останется ничего другого, как пересчитать все заново.

Целостность базы данных должна защищаться административными мерами. Постарайтесь минимизировать число учетных записей на сервере. Не выдавайте учетные записи пользователям базы данных без особой на то необходимости. Пусть подключаются в режиме удаленного доступа с помощью клиентских программ из своих собственных систем. Не забудьте задать список узлов, от которых можно принимать запросы. Одних

лишь имени пользователя и пароля недостаточно. Должен быть указан еще и перечень разрешенных IP-адресов. Сам компьютер, на котором запущен сервер MySQL, должен находиться в изолированном помещении с ограниченным доступом.

О схеме выдачи привилегий в MySQL пойдет речь в главе 27, "Безопасность". В некоторых организациях есть одна большая база данных, доступ к которой разрешен множеству пользователей, но только в режиме чтения. В других организациях каждому пользователю предоставляется собственная закрытая база данных.

Следует регулярно проверять таблицы привилегий и журнальные файлы для выявления нарушений безопасности. Можно воспользоваться командой `diff`, чтобы быстро просмотреть изменения, внесенные в таблицы привилегий. Эта системная команда возвращает отличающиеся фрагменты двух текстовых файлов. Если дампы базы данных `mysql` создаются регулярно, можно сравнить отличия предыдущего дампа от текущего. Рассмотрим пример, показанный в листинге 23.1. Сравнение дампов позволяет сделать вывод о том, что в базу данных была добавлена учетная запись нового пользователя, которому предоставлены полные привилегии на доступ ко всем базам данных. В Windows имеется адаптированная версия команды `diff`, входящая в состав пакета Cygnus. Кроме того, многие текстовые редакторы Windows обладают средствами сравнения текстовых файлов.

```
# diff then.sql now.sql
62a63
> INSERT INTO db VALUES
 ('%','mysql','hacker','Y','Y','Y','Y','Y','Y','N','Y','Y','Y');
170a172
> INSERT INTO user VALUES
('%','hacker','','N','N','N','N','N','N','N','N','N','N',
'N','N','N');
```

Просмотр журнальных файлов MySQL позволяет выявить необычные операции или обнаружить признаки неэффективной организации таблиц. Подробнее о журнальных файлах, создаваемых программой MySQL, рассказывается в главе 24, "Физическое хранение данных". Эти файлы очень быстро разрастаются, поэтому необходимо регулярно осуществлять их ротацию. С помощью инструкции FLUSH LOGS можно заставить сервер начать новые журнальные файлы, но предварительно следует переименовать существующие файлы и поместить самый старый файл в архив. В Linux-дистрибутив MySQL входит сценарий `mysql-log-rotate`, автоматизирующий этот процесс.

Регулярно проверяйте таблицы на предмет наличия в них ошибок. Подобную проверку также можно делать в автоматическом режиме. В UNIX для этого используется программа `crontab`, а в Windows — программа-планировщик. Поиск ошибок в таблицах MyISAM можно вести с помощью утилиты `myisamchk` или инструкции CHECK TABLES. Не забудьте сохранить полученные результаты в файле для последующего анализа.

Если в таблице обнаружено повреждение, его можно немедленно исправить. У демона `mysqld` есть опция `--myisam-recover`, при наличии которой в строке запуска демон будет пытаться автоматически восстановить поврежденные таблицы. Это хорошее решение на случай непредвиденных аварий, например таких, которые происходят вследствие внезапного выключения питания.

Подготовка к катастрофе

Как уже говорилось выше, компьютер, на котором работает сервер MySQL, должен находиться в изолированном помещении. Систему нужно защищать не только от незаконного проникновения, но и от физических повреждений, в том числе от природных катаклизмов. Подключите систему к источнику бесперебойного питания, чтобы в случае крайней необходимости ее можно было корректно выключить. Внезапные потери питания способны привести к повреждению таблиц.

Несмотря на все меры предосторожности, стопроцентной защиты от катастроф не существует. Иногда жесткие диски просто перестают работать. Пожары и землетрясения, слава богу, случаются редко, однако их последствия могут быть ужасающими. Но самые серьезные разрушения обычно происходят по вине пользователей. Небольшое недоразумение превращается в крупную проблему, если у администратора нет плана. Например, можно легко забыть заблокировать таблицы перед их восстановлением, что приведет лишь к еще большему разрушению данных. Чтобы защитить себя от подобных неприятностей, регулярно создавайте резервные копии и храните их вне сервера. Продумайте систему маркировки носителей резервных копий, чтобы их можно было легко идентифицировать.

Не забудьте протестировать средства восстановления данных. Инсталлируйте MySQL в другой системе и восстановите резервную копию на "пустом месте". Регулярно проверяйте архивы, чтобы защитить себя от неприятных сюрпризов.

О создании резервных копий и восстановлении таблиц будет рассказываться в главе 25, "Устранение последствий катастроф".

Поддержка пользователей

Были проведены серьезные научные исследования в области моделей данных, но ни в одной из них нельзя учесть природу самих пользователей и особенности их работы с базами данных. По мере того как пользователи знакомятся с возможностями сервера и их базы данных разрастаются, пользователи изобретают новые способы взаимодействия с сервером. Сама программа MySQL не стоит на месте, а постоянно совершенствуется. Администратор баз данных обязан следить за тем, чтобы потребности пользователей соответствовали возможностям сервера.

Администратор должен понимать, зачем пользователям нужна та или иная база данных. Потратьте время на изучение операций, выполняемых над хранимыми в ней данными. Например, база данных с информацией о кинофильмах может служить разным целям. Если известно, что она используется служащими фирмы по прокату видеокассет, то предъявляемые к ней требования будут одними, а если к ней обращаются миллионы пользователей Internet как к интерактивной энциклопедии, то требования станут совершенно другими.

Нужно также учитывать особенности взаимодействия пользователей с системой. Если большинство из них находятся в системе в рабочее время, то можно спланировать обслуживающие процедуры на период наименьшей активности, т.е. на ночь. Пользователи не знают структуру работы системы так же хорошо, как администратор, поэтому он может подсказать им наиболее эффективные способы решения основных задач.

Зная текущие потребности и особенности работы пользователей, несложно предсказать их будущие запросы. Продумайте заранее возможное расширение системы. Если база данных каждый день увеличивается на 100 Мбайт, очень скоро понадобится покупать новый жесткий диск или же сбрасывать старые данные в архив. Довольно часто происходят изменения самой программы MySQL. Организуйте регулярное обновление версий программы. Разработчики MySQL всегда сообщают о том, насколько "стабилен" тот или иной выпуск, чтобы пользователи могли решить, стоит ли на него переходить. Следите за новыми возможностями программы, которые позволят повысить эффективность работы с системой.

Новые пользователи должны быть посвящены в детали функционирования системы. Нужно не только создать для них учетные записи, но и познакомить их с существующим порядком работы. Сделайте краткий обзор утилит, имеющихся в распоряжении пользователей, и укажите, где можно найти дополнительную информацию. Продолжительность обучения зависит от сложности системы и уровня подготовки пользователей.

Пользователи рассчитывают на помощь администратора в решении проблем, связанных с базами данных. Администратор может потребовать от пользователей составлять отчеты о замеченных аномалиях. Иногда приходится заниматься разрешением конфликтов. Нужно заранее предусмотреть ситуацию, когда кто-то из пользователей попытается монополизировать системные ресурсы. Старайтесь переносить выполнение особо ресурсоемких заданий на периоды пониженной активности системы.

Разработка и внедрение стандартов

В главе 7, "Проектирование баз данных", говорилось о важности планирования жизненного цикла системы. Независимо от того, принимал ли администратор участие в разработке спецификации проекта, он должен на основании этой спецификации составить формальный план обслуживания базы данных.

Как минимум, нужно стандартизировать процедуру доступа пользователей к базе данных. Желательно также описать правила именования таблиц и столбцов. Это можно оформить в виде руководства по стилю, пример которого приведен в приложении Д, "Руководство по оформлению SQL-сценариев".

Задокументируйте каждую операцию, выполняемую над базой данных. Сюда входят уже упоминавшиеся ранее процедуры резервного копирования и восстановления, а также операции, уникальные для данной системы. Составляйте описание так, чтобы его мог понять другой администратор, ведь если по какой-то причине вы больше не сможете обслуживать систему, на ваше место придется взять другого человека.

Помимо самой базы данных администратор должен участвовать в проектировании и реализации приложений, предназначенных для работы с ней. Программисты могут больше разбираться в особенностях того или иного языка и не знать оптимальных методик взаимодействия с базой данных.

Кроме того, в обязанности администратора входит анализ и верификация существующих приложений баз данных. Опыт администратора позволяет ему лучше ориентироваться в выборе таких приложений, как, например, генераторы отчетов. Прежде чем рекомендовать то или иное приложение, обязательно протестируйте его на имеющейся базе данных.

ФИЗИЧЕСКОЕ ХРАНЕНИЕ ДАННЫХ

В этой главе...

Способ хранения таблиц и баз данных
Выделенные разделы
Типы таблиц
Столбцы
Блокировки таблиц
Индексы
Дескрипторы файлов
Системная память
Журнальные файлы

Глава 24

В этой главе рассказывается о том, как в MySQL организуется физическое хранение данных. Описывается способ представления таблиц на жестком диске и формат журнальных файлов. Понимание этих вопросов важно для создания эффективных баз данных.

Способ хранения таблиц и баз данных

В MySQL таблице соответствует несколько файлов. Их имена совпадают с именем таблицы, а расширение определяет назначение файла. К примеру, файл с расширением `.frm` содержит описание структуры таблицы. Что касается баз данных, то они являются подкаталогами основного каталога **данных** (по умолчанию это `/usr/local/var`). Имя подкаталога соответствует имени базы данных. Это означает, что имена баз данных и таблиц отвечают тем же требованиям, которые предъявляются к именам файлов в данной системе. Скажем, файловая система ext2 в Linux чувствительна к регистру символов, а FAT32 в Windows — нет.

Операционные системы налагают свои ограничения на максимальный размер файла. Обычно он составляет от 2 до 4 Гбайт. Для таблиц типа **MyISAM**, описываемого ниже, все данные сохраняются в одном файле, следовательно, максимальный размер файла одновременно является максимальным размером таблицы.

MySQL позволяет разбивать табличные данные на несколько файлов при наличии опции `RAID_TYPE` в инструкции `CREATE TABLE`. Тогда максимальный размер таблицы возрастет во столько раз, сколько файлов для нее создается. Можно даже разместить эти файлы на разных физических дисках, чтобы повысить производительность базы данных.

Выделенные разделы

В некоторых СУБД поддерживаются выделенные файловые системы. Такая файловая система инсталлируется в собственный дисковый **раздел**, и лишь СУБД знает, как ее использовать. Сервер берет на себя выполнение всех функций файловой системы, тогда как обычно они предоставляются самой операционной системой. Теоретически это способствует повышению производительности.

Тем не менее MySQL не позволяет записывать базы данных в выделенные разделы, так как изоляция от операционной системы не стоит незначительного выигрыша производительности. Современные операционные системы обладают гораздо большей производительностью, чем раньше, в основном из-за того, что они способны **кэшировать** дисковые блоки в оперативной памяти. Если бы программе MySQL пришлось выполнять эти функции в обход операционной системы, то тогда память, используемая для кэширования, тоже стала бы выделенной, а объем общедоступной оперативной памяти уменьшился бы.

Управление файлами в рамках операционной системы имеет свои преимущества. Во-первых, разработчики системы проделывают огромную работу по отладке соответствующих функций, а во-вторых,появляется возможность пользоваться стандартными утилитами обработки файлов. Создать резервную копию базы данных MySQL можно с помощью обычной утилиты tar, а не специализированной программы. Выделенные файловые системы требуют целого набора управляющих утилит, в частности для проверки и восстановления файлов. Всего этого нет в MySQL

Типы таблиц

В MySQL версии 3.23.37 поддерживаютсясемь типов таблиц. Три из них — **Berkeley DB**, Gemini и **InnoDB** — ориентированы на транзакции, а четыре — Heap, ISAM, Merge и **MyISAM** — нет. Транзакции являются относительно новым понятием в MySQL, но соответствующие функции для таблиц Berkeley DB и InnoDB существуют уже достаточно давно, что позволило включить их в стандартные бинарные дистрибутивы. Подробнее об этом рассказывалось в главе 9, "Транзакции и параллельные вычисления".

Стандартным типом таблиц в MySQL является тип MylSAM. Он возник на основе более старого типа ISAM, который все еще существует, хотя использовать его не рекомендуется. Переопределить установку по умолчанию позволяет опция TYPE инструкций CREATE TABLE и ALTER TABLE (см. главу 13, "Инструкции SQL").

Berkeley DB

Проект Berkeley DB (**BDB**) начался в Калифорнийском университете в Беркли. Впоследствии его авторы сформировали компанию **Sleepycat** Software *(www.sleepycat.com)* и занялись распространением коммерческой версии СУБД. Многие утилиты до сих пор работают со старыми версиями **BDB** (1.85 и 1.86), в то время как компания Sleepycat Software предлагает уже семейство версий *3.x* и 4.x Эта СУБД свободно **распространяется** с исходными кодами, и ею можно пользоваться бесплатно, за исключением случаев, когда на ее основе планируется создавать приложения, не распространяемые на условиях открытой лицензии.

BDB — это простая файловая СУБД, поддерживающая транзакции, но не располагающая каким-либо языком запросов. В MySQL эта СУБД нужна для того, чтобы можно было работать с таблицами в режиме транзакций.

С Web-узла MySQL можно загрузить скомпилированную версию программы, в которую встроена поддержка BDB. Если впоследствии потребуется отключить эту поддержку, достаточно будет запустить демон `mysqld` с опцией `--skip-bdb`. На этапе компиляции MySQL поддержка BDB включается с помощью опции `--with-berkeley-db`. Для MySQL нужна исправленная версия BDB, которая входит в исходный дистрибутив MySQL.

Разработчики MySQL тесно сотрудничают с программистами компании **Sleepycat**, чтобы гарантировать максимальную эффективность использования библиотеки BDB. Вообще-то, поддержка BDB в MySQL появилась не так давно (в версии 3.23.24), но, учитывая высокую стабильность обоих продуктов, их интеграция не привела к возникновению каких-либо трудностей для пользователей. Разработчики MySQL планируют и дальше улучшать поддержку BDB.

С функциональной точки зрения таблицы BDB ведут себя аналогично таблицам **MyISAM**. Нет никаких ограничений на число столбцов или индексов, как в случае резидентных таблиц. Единственное условие: для таблиц BDB обязательно наличие первичного ключа. Если он не задан, MySQL самостоятельно создаст внутренний первичный ключ, охватывающий первые пять байтов каждой записи. К таблицам BDB можно применять инструкцию LOCK TABLES, но при частой работе с такими таблицами лучше пользоваться преимуществами транзакций.

Сами транзакции реализуются посредством журнальных файлов, куда записываются сведения об изменении табличных данных. Когда происходит отмена транзакции, функции библиотеки BDB читают журнальные файлы и делают "обратные" исправления. Журнальные файлы носят имена вида `log.0000000001` и располагаются в каталоге данных, хотя эту установку можно изменить с помощью опции `--bdb-logdir`, указываемой при запуске сервера.

MySQL пытается очищать журнальные файлы BDB в момент создания нового журнального файла. При этом удаляются ненужные файлы. Если не хотите ждать, воспользуйтесь инструкцией FLUSH LOGS.

В BDB таблицы блокируются на уровне страниц. Страница — это совокупность последовательно расположенных записей таблицы. Страничные блокировки необходимы, когда MySQL сканирует таблицу, а также при удалении, вставке и обновлении записей. В отличие от таблиц MylSAM, блокировки таблиц BDB могут приводить к возникновению тупиков, т.е. взаимоблокировок. В подобной ситуации одна или несколько транзакций отменяется. Это следует учитывать при написании приложений. Инструкция, которая приводит к автоматической отмене транзакции, возвращает сообщение об ошибке. Транзакции отменяются также в случае нехватки места в файловой системе.

СУБД BDB не ведет подсчет записей в таблицах, но MySQL хранит собственный счетчик. Он используется модулем оптимизации объединений при выборе индексов. Значение счетчика может быть неточным, если произошел сбой базы данных, но счетчик можно сбросить с помощью инструкции ANALYZE TABLE или OPTIMIZE TABLE. Ведение счетчика записей немного замедляет работу с таблицами BDB.

Табличные данные хранятся в виде двоичного дерева. Это более медленный метод, чем тот, который применяется для таблиц MylSAM. СУБД BDB оставляет в дере-

ве пустые позиции, чтобы операции вставки выполнялись быстрее. В результате размер файла становится больше, чем нужно.

В отличие от таблиц MyISAM, индексы в таблицах BDB не сжимаются, т.е. они занимают больше места. Если в запросе участвуют столбцы одного индекса, обращение к табличным данным не производится, так как в этом нет необходимости. С этой целью BDB позволяет объединять первичный ключ со столбцами другого индекса (для таблиц MylSAM такая возможность не поддерживается).

Gemini

Функции работы с таблицами Gemini реализовали программисты компании NuSphere *(www.nusphere.com)*. Эта компания обеспечивает поддержку и обучение пользователей MySQL. Бета-тестирование таблиц Gemini началось в апреле 2001 г. и на момент написания книги еще продолжалось.

В этих таблицах отсутствуют столбцы типа BLOB и TEXT. Количество пользователей, которые могут одновременно работать с таблицами, по умолчанию равно 100. Данную установку можно изменить с помощью серверной переменной `gemini_connection_limit`.

Таблицы Gemini блокируются на уровне записей, так как это выгоднее с точки зрения многопользовательской работы. Если заблокировать всю таблицу, другие потоки вынуждены будут встать в очередь на доступ к таблице. Блокировки записей предотвращают доступ к единичным записям, позволяя нескольким потокам работать с одной таблицей, но в разных ее "участках".

Как и в случае таблиц MylSAM, доступ кфайлу данных таблицы Gemini не требуется, если в запросе участвуют столбцы одного индекса. Для этих таблиц также ведется счетчик записей, используемый модулем оптимизации объединений.

Heap

MySQL хранит таблицы типа Heap в памяти, а не в файловой системе. Следовательно, доступ к ним осуществляется чрезвычайно быстро. Для поиска записей применяется хэш-таблица, но проблем со вставкой или с удалением записей не возникает, в отличие от других реализаций резидентных таблиц.

Резидентные таблицы не располагают многими возможностями обычных таблиц. Они не могут иметь столбцы типа BLOB или TEXT. Нельзя использовать флаг AUTO_INCREMENT. Можно создавать индексы, но нельзя индексировать столбцы, допускающие значения NULL. Индексы используются только в операциях = и <=>.

Записи резидентных таблиц имеют фиксированную длину. Для столбцов типа VARCHAR сразу выделяется максимальное число байтов. Поскольку память – ограниченный ресурс, можно задать предельное количество записей в резидентной таблице. Для этого предназначена опция `max_rows` инструкции CREATE TABLE. Серверная переменная `max_heap_table_size` задает максимальный объем памяти, занимаемой всеми резидентными таблицами.

Доступ к резидентным таблицам имеют все пользователи. Эти таблицы уничтожаются при выключении сервера.

InnoDB

СУБД InnoDB была разработана **Хейкки Туури (Heikki Tuuri)** из компании **Innobase Oy (*www.innodb.com*)** — финского производителя программного обеспечения, специализирующегося на технологии реляционных баз данных. InnoDB представляет собой **результат** исследований, проводимых Хейкки в университете Хельсинки. Поддержка InnoDB появилась в MySQL версии 3.23.34a. Сама СУБД доступна на условиях открытой лицензии.

На Web-узле InnoDB можно найти массу информации о деталях работы ядра этой СУБД. Ядро не существует само по себе, а является дополнением к MySQL. С Web-узла MySQL можно загрузить скомпилированную версию программы, в которую встроена поддержка InnoDB. Если впоследствии потребуется отключить эту поддержку, достаточно будет запустить демон `mysqld` с опцией `--skip-innodb`. На этапе компиляции MySQL поддержка InnoDB включается с помощью опции `--with-innodb`. Исходные коды InnoDB входят в исходный дистрибутив MySQL.

В отличие от таблиц **MyISAM**, где для каждой таблицы создается один файл данных, данные InnoDB хранятся в больших совместно используемых файлах. Можно создать произвольное число файлов данных, но их нельзя будет удалить. Размер файлов определяется в конфигурационном файле. Если нужно уменьшить объем **дискового** пространства, занимаемого таблицами InnoDB, создайте резервные копии таблиц, после чего удалите все файлы InnoDB и позвольте программе MySQL восстановить их в соответствии с новыми установками **конфигурационного** файла. Журнальные файлы InnoDB можно безопасно удалить после остановки сервера. При повторном запуске сервера программа MySQL создаст журнальные файлы заново.

В листинге 24.1 приведен пример опций, которые необходимо добавить в конфигурационный файл в группу `[mysqld]`, чтобы активизировать таблицы InnoDB. Размер файлов здесь задан относительно небольшим, что вполне подходит для целей эксперимента. На практике используются файлы гораздо большего размера. Список опций и переменных демона mysqld был приведен в главе 14, "Утилиты командной строки".

```
innodb_data_home_dir = /usr/local/var/innodb/
innodb_data_file_path = ibdata1/ibdata1:100M;ibdata2/ibdata2:100M
set-variable = innodb_mirrored_log_groups=1
innodb_log_group_home_dir = /disk2/innodb/log
set-variable = innodb_log_files_in_group=3
set-variable = innodb_log_file_size=16M
set-variable = innodb_log_buffer_size=8M
innodb_flush_log_at_trx_commit=1
innodb_log_arch_dir = /disk2/innodb/log
innodb_log_archive=0
set-variable = innodb_buffer_pool_size=25M
set-variable = innodb_additional_mem_pool_size=5M
set-variable = innodb_file_io_threads=4
set-variable = innodb_lock_wait_timeout=50
```

После добавления опций в конфигурационный файл необходимо перезапустить сервер MySQL. Все необходимые файлы будут созданы автоматически. На это может уйти некоторое время, в зависимости от размера файлов и скорости жесткого диска.

Таблицы InnoDB блокируются на уровне записей. Это происходит без участия пользователей по мере выполнения инструкций в рамках транзакций. Инструкция LOCK TABLE может конфликтовать с блокировками InnoDB. В отличие от таблиц MyISAM, блокировки таблиц InnoDB способны приводить к возникновению тупиков, т.е. взаимоблокировок. В подобной ситуации одна или несколько транзакций отменяется. Это следует учитывать при написании приложений. Инструкция, которая приводит к автоматической отмене транзакции, возвращает сообщение об ошибке. Транзакции отменяются также в случае нехватки места в файловой системе.

На случай отмены транзакций ведется журнал транзакций. Он подвержен внутренней ротации, т.е. когда заполняются все записи, самые старые из них начинают удаляться.

На момент написания книги существовало несколько ограничений таблиц InnoDB. Самое существенное из них заключалось в способе отслеживания таблиц. В InnoDB ведется каталог таблиц, который не поддерживается инструкцией DROP TABLE, поэтому каждую таблицу приходится удалять отдельно. Не разрешается индексировать префикс столбца, а также индексировать столбцы типа BLOB и TEXT. Максимальное количество столбцов в таблице— 1000. Флаг DELAYED в инструкции INSERT не поддерживается.

ISAM

До версии 3.23 стандартным типом таблиц в MySQL был тип ISAM. Он не обладает такими возможностями, как более новый тип MyISAM, поэтому в современных версиях MySQL использовать его не рекомендуется.

Merge

В таблице типа Merge группируется несколько таблиц MylSAM одинаковой структуры. Программа MySQL создает файл с расширением .MRG, в котором содержится список таблиц. При доступе к объединенной таблице программа обращается к каждой таблице из списка. Если в списке всего одна таблица, то создается только ее псевдоним. Если же таблиц две или более, их записи трактуются так, будто они находятся в одной таблице. С функциональной точки зрения объединенная таблица обладает всеми свойствами обычной таблицы.

Объединенная таблица создается очень быстро, так как требуется всего лишь сформировать список имен исходных таблиц. В случае уничтожения такой таблицы удаляется лишь MRG-файл, но не исходные таблицы.

У объединенных таблиц есть ряд недостатков. В них нельзя вставлять записи, поскольку программа MySQL не имеет возможности определить, в какую из исходных таблиц они должны быть помещены. Кроме того, при работе с такими таблицами используется большее число файловых дескрипторов, так как программе приходится открывать каждую исходную таблицу в отдельности. Следовательно, извлечение данных из объединенных таблиц осуществляется медленнее, чем из таблиц других типов.

Даже если дескрипторы индексных файлов совместно используются несколькими потоками, все равно приходится читать индексный файл каждой исходной таблицы.

Несмотря на упомянутые ограничения, у объединенных таблиц есть и несомненные преимущества. Они позволяют интерпретировать группу таблиц как единое целое и в то же время продолжать работать с отдельными ее компонентами. Это идеальное решение для крупных таблиц, которые легко разбиваются на фрагменты. Например, журнальные файлы Web-сервера можно группировать в месячные таблицы. Если одна из таблиц повреждается, необходимо восстанавливать только ее, а не всю группу. Кроме того, исходные таблицы можно хранить на разных физических дисках, что способствует повышению производительности.

MyISAM

MyISAM — это стандартный тип таблиц в MySQL, если только в конфигурационном файле не задано иное. Для таблиц этого типа создан ряд специализированных утилит, позволяющих манипулировать табличными файлами. Сюда входят утилита `myisamchk` для проверки и восстановления таблиц и утилита `myisampack` для создания сжатых таблиц (см. главу 14, "Утилиты командной строки").

Таблицы MyISAM являются платформно-независимыми. Табличные файлы можно перемещать между компьютерами разных архитектур и разными операционными системами без всякого преобразования. Для этого MySQL хранит все числа с плавающей запятой в формате IEEE, а все целые числа — в формате с прямым порядком следования байтов. С точки зрения производительности это совершенно непринципиально.

MySQL хранит счетчик подключений к таблице MyISAM. Когда таблица закрывается, счетчик сбрасывается в нуль. Если сервер неожиданно завершает работу, счетчик остается положительным числом. В таком случае в процессе перезапуска сервер обнаружит проблему. Это не означает, что таблица повреждена, но подобная возможность существует. Следует немедленно выполнить инструкцию CHECK TABLE или вызвать утилиту myisamchk. Можно также запустить демон `mysqld` с опцией `--myisam-recover`, чтобы заставить его восстанавливать все таблицы MyISAM с ненулевым значением счетчика.

Для таблиц MyISAM разрешены одновременные операции вставки и выборки, если только в таблице нет пустых участков. Такие участки создаются инструкциями DELETE и могут быть заполнены последующими инструкциями INSERT. MySQL блокирует таблицу MyISAM, пока инструкция INSERT заполняет пустой участок. Для удаления пустых мет необходимо оптимизировать таблицу.

Для автоинкрементных столбцов таблиц MyISAM программа MySQL ведет внутренний счетчик, а не просто добавляет единицу к наибольшему значению столбца. Это дает небольшой выигрыш производительности при операциях вставки, но также означает, что значения столбца никогда не используются повторно. В таблицах других типов при удалении строки с наибольшим значением счетчика и последующей вставке новой строки ей будет присвоен тот же самый идентификатор.

Индексные файлы имеют расширение `.MYI`. Файлы с расширением `.MYD` содержат данные, а с расширением `.frm` — схему таблицы. Если индексный файл по какой-то причине теряется, программа перестраивает индексы, используя информацию из `frm`-файла.

По умолчанию в каждой таблице может быть не более тридцати двух индексов, но это значение можно повысить до шестидесяти четырех. Индексы создаются в виде двоичных деревьев. Разрешается индексировать столбцы типа BLOB и TEXT, а также столбцы, допускающие значения NULL.

В таблицах MyISAM могут быть фиксированные, динамические либо сжатые записи. Выбор между фиксированным и динамическим форматом диктуется определениями столбцов. Для создания сжатых таблиц предназначена утилита `myisampack` (см. главу 14, "Утилиты командной строки").

Таблица будет иметь записи фиксированной длины, если в ней нетстолбцов типа VARCHAR, BLOB или TEXT. Одинаковая длина записей имеет свои преимущества. Утилите `myisamchk` будет проще восстанавливать поврежденные записи, если она знает их точную длину. Такие записи никогда не приходится разбивать на части при наличии в таблице пустых промежутков, что ускоряет операции чтения. Правда, записи фиксированной длины обычно занимают больше места на диске.

Все записи таблицы будут динамическими, если в ней есть столбцы типа VARCHAR, BLOB или TEXT. Возможно также приведение столбцов типа CHAR к типу VARCHAR, если их длина больше четырех символов. Длина каждой записи отслеживается по специальному заголовку. В нем указана длина текущего сегмента записи. Поскольку при повреждении таблицы связи между фрагментами могут теряться, корректное восстановление записи не всегда возможно.

Динамические записи часто требуют дефрагментации. При их удалении возникают пустые участки, которые не всегда в точности заполняются вставляемыми записями. Более того, из-за операций обновления длина записи может увеличиваться или уменьшаться. Если запись не помещается в отведенном для нее месте, она разбивается на два или более сегмента. По мере распределения записей по файлу время поиска данных возрастает. Устранить фрагментацию можно с помощью инструкции `OPTIMIZE TABLES`.

Сжатые таблицы занимают гораздо меньше места, чем таблицы с фиксированными или динамическими записями. Их удобно создавать в медленных файловых системах, например в тех, которые используются в компакт-дисках. Каждая запись сжимается отдельно с применением отдельной хэш-таблицы для каждого столбца. Сжатая таблица создается утилитой myisampack. С помощью утилиты myisamchk можно преобразовать сжатую таблицу обратно в фиксированный или динамический формат.

Если таблица MylSAM находится на переполненном диске и в нее добавляется запись, программа MySQLперейдет в бесконечный цикл, ожидая освобождения места на диске.

Столбцы

В табл. 24.1 указаны размерности стандартных типов данных MySQL. Значения некоторых типов всегда занимают фиксированный объем памяти. Например, размерность столбцов типа INTEGER всегда составляет 4 байта. Столбцы типа CHAR могут иметь размерность от 0 до 255, но в момент создания таблицы под них отводится фиксированный объем памяти. Существуют также столбцы переменной размерности. Например, столбцы типа VARCHAR и BLOB интерпретируются в соответствии с их содержимым.

Тип	*Размерность*
BIGINT	**8 байтов**
BLOB, TEXT	**Длина** содержимого **+ 2 байта**
CHAR (*длина*)	Указанное число **байтов**
DATE	**3 байта**
DATETIME	8 байтов
DECIMAL *(длина, точность)*	**Длина + 1 байт, если** точность **равна 0; в** противном **случае — длина + 2 байта**
DOUBLE	**8** байтов
DOUBLE PRECISION	**8** байтов
ENUM	**1 байт, если в перечислении менее 255 элементов; в** противном **случае — 2 байта**
FLOAT	**4 байта**
FLOAT (*длина*)	4 **байта, если длина <= 24; в** противном **случае — 8** байтов
INT	**4 байта**
INTEGER	**4 байта**
LONGBLOB, LONGTEXT	**Длина + 2 байта**
MEDIUMBLOB, MEDIUMTEXT	**Длина + 2 байта**
MEDIUMINT	**3 байта**
NUMERIC *(длина, точность)*	**Длина + 1 байт, если** точность **равна 0; в** противном **случае – длина + 2 байта**
REAL	**8** байтов
SET	**1, 2, 3; 4 или 8** байтов, **в** зависимости от количества элементов множества
SMALLINT	**2 байта**
TIME	**3 байта**
TIMESTAMP	**4 байта**
TINYBLOB, TINYTEXT	**Длина + 2 байта**
TINYINT	**1 байт**
VARCHAR (*длина*)	**Длина** содержимого **+ 1 байт**
YEAR	**1 байт**

Как указывалось выше, таблицы MyISAM содержат записи фиксированной либо переменной длины. Переход во второй режим осуществляется при наличии столбцов переменной размерности.

Значения столбцов типа DECIMAL и NUMERIC хранятся в строковом виде, что позволяет обеспечить точность представления десятичных чисел. Каждой цифре соответствует один символ, еще по одному символу отводится на знаковый разряд и десятичную точку.

Блокировки таблиц

В MySQL разрешается явно блокировать таблицы с помощью инструкции LOCK TABLES. Тем неменее не рекомендуется делать это для таблиц тех типов, которые поддерживают транзакции. Блокировки и транзакции — это два разных способа решения проблемы одновременного доступа к таблице, поэтому нужно сделать выбор в пользу одного из них.

В зависимости от инструкции могут также применяться неявные блокировки. Например, инструкция UPDATE способна немедленно заблокировать таблицу, запретив доступ к ней другим потокам. Блокировки обоих типов защищены от возникновения тупиковых ситуаций, так что можно не волноваться по поводу отмены той или иной инструкции.

Можно заблокировать таблицу таким образом, чтобы разрешить другим потокам обращаться к ней для чтения. Это называется блокировкой чтения. Блокировка записи гарантирует текущему потоку монопольный доступ к таблице. Запросы на чтение откладываются до тех пор, пока не будут сняты все блокировки записи. Эту установку можно изменить с помощью флагов инструкций либо путем задания специальных серверных переменных. Для SQL-инструкций создаются две очереди. Чтобы программа MySQL начала извлекать инструкции из очереди на чтение, очередь на запись должна быть пуста. При наличии флага LOW_PRIORITY инструкции DELETE, INSERT и UPDATE помещаются в очередь на чтение, т.е. они получают такой же приоритет, что и инструкции SELECT. Флаг HIGH_PRIORITY переводит инструкцию SELECT в очередь на запись.

Индексы

В MySQL индексы хранятся в виде двоичных деревьев. Деревья перестраиваются по мере вставки записей. Это означает, что каждый индекс вызывает небольшое снижение производительности. Как правило, индексы повышают скорость операций выборки за счет снижения скорости операций записи. Тем не менее наличие индекса еще не гарантирует никакого ускорения. Нужно соотносить их с теми запросами, которые планируется выполнять. Чтобы понять, насколько эффективным окажется тот или иной индекс, пользуйтесь инструкцией EXPLAIN (рассматривается в главе 26, "Оптимизация").

Определения индексов хранятся в frm-файле, асами индексируемые значения — в файле с расширением .MYI. Если индексный файл отсутствует на момент запуска сервера, он будет автоматически воссоздан. Таким образом, при создании резервных копий можно не заботиться об индексах в целях экономии места. Позднее, в процессе восстановления базы данных, программа MySQL создаст индексы заново на основании схемы таблицы.

Индексы способны повысить производительность инструкций, связанных с поиском записей. Они ускоряют процесс сравнения столбцов при выполнении операций объединения. Кроме того, они помогают находить минимальное и максимальное значения столбца и ускоряют выполнение инструкций SELECT с предложением ORDER BY.

Чтобы индекс был задействован, он должен быть указан во всех частях предложения WHERE. Если используется лишь часть индекса, то должен соблюдаться порядок обращения к индексируемым столбцам: слева направо. Для примера рассмотрим таблицу, определение которой приведено в листинге 24.2.

```
CREATE TABLE car (
    Make CHAR(32) NOT NULL,
    Model CHAR(32) NOT NULL,
    Introduced YEAR,

    PRIMARY KEY(Make, Model)
);
```

У таблицы car имеется составной первичный ключ. В запросе, который показан в листинге 24.3, индекс будет использован, так как столбец Make является самым левым компонентом индекса.

```
SELECT *
  FROM car
 WHERE Make='Ford'
```

А вот в следующем запросе (листинг 24.4) этого не произойдет, поскольку правило очередности столбцов не соблюдается.

```
SELECT *
  FROM car
 WHERE Model='Pinto'
```

В листинге 24.5 индекс также не используется, из-за того что самый левый компонент индекса нельзя применить к каждой записи. Если бы в предложении WHERE стоял оператор AND, а не OR, всебыло бы наоборот.

```
SELECT *
  FROM car
 WHERE Make='Ford'
    OR Model='Impala'
```

Следующий запрос (листинг 24.6) является правильным с точки зрения использования индекса. В данном случае просмотр значений столбца осуществляется слева направо.

```
SELECT *
  FROM car
 WHERE Make LIKE 'F%'
```

В листинге 24.7 индекс не используется, потому что просмотр значений столбца осуществляется справа налево (метасимвол % стоит вначале).

```
SELECT *
  FROM car
 WHERE Make LIKE '%d'
```

Дескрипторы файлов

Сервер MySQL представляет собой один процесс со множеством потоков. Для каждого сеанса подключения к серверу создается свой поток. Каждому потоку требуется один или несколько дескрипторов файлов, чтобы он мог осуществлять чтение и запись таблиц. Операционная система ограничивает количество файловых дескрипторов, доступных процессу. Это число может быть самым разным. Например, в AIX оно равно 2000 по умолчанию, а в Solaris — всего лишь 64. В Linux лимит по умолчанию составляет 1024 дескриптора. В Windows NT и 2000 видимый предел отсутствует.

Чтобы не исчерпать лимит ресурсов, MySQL хранит кэш файловых дескрипторов всех соединений. По умолчанию размер кэша составляет 64 позиции. В случае переполнения кэша MySQL закрывает самый старый дескриптор, освобождая место для нового. В периоды высокой активности пользователей работа программы может замедляться из-за необходимости часто закрывать и открывать файлы. Пока сервер не прекратит работу или буфер не будет принудительно очищен, файловые дескрипторы остаются открытыми. Если активность настолько высока, что все дескрипторы, находящиеся в кэше, открыты, программа временно увеличивает размер кэша.

Размер кэша дескрипторов можно задать другим, но не забывайте об ограничении, которое накладывается операционной системой. Правда, ее собственный лимит тоже можно изменить. Для этого существует, например, команда `ulimit`. Еще один способ — перекомпиляция ядра.

Чтобы получить доступ к таблице, поток должен иметь в своем распоряжении дескриптор ее файла данных. Все потоки совместно владеют дескриптором индексного файла. Таким образом, когда три потока одновременно извлекают данные из таблицы, используются четыре дескриптора. Если таблица дважды указана в предложении FROM, например в случае операции самообъединения, программа MySQL вынуждена открывать отдельный дескриптор для каждой ссылки на таблицу.

Открытие файла подразумевает его поиск в каталоге, поэтому чем больше файлов в каталоге, тем больше времени уходит на поиск файла. В MySQL таблицы хранятся в виде файлов, следовательно, чем больше таблиц в базе данных, тем дольше открывается новый дескриптор. Эта зависимость ослабевает благодаря кэшу дескрипторов, поскольку дескрипторы долгое время остаются открытыми.

Системная память

В MySQL специальные буферы и кэши используются для самых разных целей. Их размеры можно задавать в конфигурационном файле или в командной строке запуска сервера. Соответствующие опции описывались в главе 14, "Утилиты командной строки".

У каждого потока есть свой стек, буфер приема входных данных от клиента и буфер результатов запроса. Размер стека задается серверной переменной `thread_stack`, а размеры обоих буферов — переменной `net_buffer_length`. Последняя определяет начальные размеры буферов, так как они могут увеличиваться в случае необходимости, например при обработке столбцов типа BLOB или TEXT.

Все потоки совместно используют индексный буфер. Его размер определяется переменной `key_buffer_size`. В операциях объединения, проходящих без участия индексных столбцов, используется отдельный буфер (переменная `join_buffer_size`), как и в операциях сканирования таблиц (переменная `record_buffer`).

Если для выполнения операции объединения требуется временная таблица, она создается как резидентная (тип Heap). Максимальный размер таких таблиц определяется переменной tmp_table_size. После превышения этого предела таблица преобразуется в формат MyISAM. В любом случае временные таблицы удаляются по окончании операции.

Журнальные файлы

Помимо журналов транзакций, создаваемых для некоторых типов таблиц, в MySQL имеются еще семь различных журналов, все — необязательные. Если такие журналы начали вестись, то нужно следить за их размерами, поскольку они непрерывно разрастаются, причем некоторые — довольно быстро. Журнальные файлы можно удалять вручную после остановки сервера или поручить эту задачу сценарию. В дистрибутив MySQL для RedHat Linux входит сценарий ротации журнальных файлов, который называется `mysql-log-rotate`. Его можно периодически запускать с помощью демона `cron`.

Наличие любого журнального файла вызывает определенное снижение производительности сервера. Например, при каждой операции обновления программа MySQL вынуждена вносить запись в двоичный журнал.

Двоичный журнал

В двоичном журнале фиксируются все действия, связанные с изменением табличных данных. Сюда не входят инструкции DELETE и UPDATE, условию отбора которых соответствует нуль записей или которые присваивают ячейкам их текущие значения. Записи файла хранятся в эффективном двоичном формате. Изменения фиксируются

сразу же после выполнения инструкции, но до того, как будут сняты блокировки. Записывается также информация о времени, которое ушло на выполнение инструкции.

По умолчанию данный файл создается в каталоге данных, а его имя соответствует имени компьютера с добавлением префикса -bin и порядкового номера. Типичное имя выглядит так: `/usr/local/var/red-bin.001`. Если выполнить инструкцию FLUSH LOGS, будет создан новый журнальный файл со следующим порядковым номером. Имена журнальных файлов отслеживаются в файле с расширением `.index`.

Утилита `mysqlbinlog` читает двоичный журнал и записывает извлеченные из него инструкции в поток `stdout`. Их можно направить утилите `mysql`, чтобы воспроизвести все изменения. Это хороший способ восстановления базы данных после краха (он описывается в главе 25, "Устранение последствий катастроф"). В листинге 24.8 показаны две записи двоичного журнала, о которых сообщила утилита `mysqlbinlog`.

```
# at 171
#010605 11:52:14 server id 1 Query thread_id=2 exec_time=0
error_code=0
use freetime;
SET TIMESTAMP=991767134;
UPDATE session SET LastAction = now() WHERE ID='fNbbnOLBYY1qesga';
# at 269
#010605 11:52:14 server id 1 Query thread_id=2 exec_time=1
error_code=0
use freetime;
SET TIMESTAMP=991767134;
DELETE FROM project_view WHERE Project=2 AND User=2;
```

В схеме репликации, применяемой в MySQL, двоичный журнал используется для синхронизации главной и подчиненной баз данных. Помните, что после очистки журналов главного сервера старые журналы нужно хранить до тех пор, пока не произойдет синхронизация подчиненного сервера. Подробнее процесс репликации рассмотрен в главе 29, "Распределенные базы данных".

Изменения, сделанные в ходе транзакции, сохраняются в кэше, пока не будет выполнена инструкция COMMIT. Максимальный размер резидентного кэша определяется переменной binlog_cache_size. Если размер резидентной таблицы превышает этот предел, изменения записываются во временный файл.

Двоичный журнал пришел на замену журналу обновлений, который использовался в старых версиях MySQL.

Журнал отладки

Если скомпилировать клиент или сервер MySQL с включением средств отладки, то программа начнет вести журнал отладки. По умолчанию отладочная информация записывается в файл `/tmp/mysql.trace`, но эту установку можно изменить с помощью опции `--debug`.

В MySQL используется библиотека функций отладки, которую написал Фред Фиш (Fred Fish). Подробнее об отладке MySQL рассказывается в главе 31, "Расширение возможностей MySQL".

Журнал ошибок

В журнале ошибок хранится информация о запуске и остановке сервера. Здесь же регистрируются сообщения об ошибках и предупреждения. Фрагмент журнала ошибок показан в листинге 24.9. Несложно узнать время, когда был запущен и остановлен сервер. В отчете сообщается о том, что два соединения с базой данных были прерваны в процессе остановки сервера. Это были постоянные соединения, созданные **PHP-сценарием**, который выполняется в локальной системе. Разрыв соединений не повлиял на работу сценария.

```
010605 12:36:32 mysqld started
InnoDB: Started
/usr/local/libexec/mysqld: ready for connections
010605 12:37:01  /usr/local/libexec/mysqld: Normal shutdown

010605 12:37:01  Aborted connection 1 to db: 'freetime' user: 'httpd'
host: 'localhost' (Got timeout reading communication packets)
010605 12:37:01  Aborted connection 2 to db: 'freetime' user: 'httpd'
host: 'localhost' (Got timeout reading communication packets)
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
010605 12:37:01  /usr/local/libexec/mysqld: Shutdown Complete

010605 12:37:01   mysqld ended
```

Журнал ошибок находится в каталоге данных. Его имя соответствует имени компьютера с добавлением расширения `.err`.

Журнал MyISAM

Чтобы включить журнал MyISAM, необходимо запустить сервер с опцией `--log-isam`. Разработчики MySQL используют этот журнал для отладки обработчика таблиц MyISAM. Журнальный файл создается в каталоге данных под именем `myisam.log`. Существует утилита `myisamlog`, предназначенная для сбора статистической информации из этого файла (см. главу 14, "Утилиты командной строки"). В листинге 24.10 показаны результаты работы этой утилиты.

```
Commands     Used count       Errors     Recover errors
open                2  5          0                  0
write               9             0                  0
```

update	2	0	0
delete	1	0	0
close	25	0	0
extra	220	0	0
Total	282	0	0

Информация, хранящаяся в данном журнальном файле, представляет интерес только для разработчиков MySQL.

Журнал запросов

В журнале запросов регистрируются все запросы, посылаемые базе данных. Этот файл создается в каталоге данных, а его имя соответствует имени компьютера с добавлением расширения `.log`. В листинге 24.11 показаны первые несколько записей такого журнала. Последняя колонка файла содержит длинные строки, поэтому я отформатировал их, чтобы они уместились на странице.

Листинг 24.11. Журнал

```
/usr/local/libexec/mysqld, Version: 3.23.39-log, started with:
Tcp port: 3306  Unix socket: /tmp/mysql.sock
Time                 Id Command    Argument
010605 11:51:45       1 Connect    httpd@localhost on freetime
                      1 Statistics
                      1 Init DB    freetime
                      1 Query      delete from session where
                                   LastAction < '2001-06-05
                                   03:51:45'
                      1 Query      SELECT User FROM session WHERE
                                   ID='fNbbnOLBYY1qesga'
                      1 Query      UPDATE session SET LastAction
                                   = now()
                                   WHERE ID='fNbbnOLBYY1qesga'
                      1 Query      DELETE FROM session WHERE ID <>
                                   'fNbbnOLBYY1qesga' AND User=2
                      1 Query      SELECT * FROM user WHERE ID = 2
```

Журнал запросов не должен быть постоянно активен, так как он разрастается наиболее быстро. Инструкции регистрируются в том порядке, в каком они поступают на сервер. Этот порядок может отличаться от порядка выполнения инструкций.

Журнал медленных запросов

В журнале медленных запросов регистрируются инструкции, которые выполнялись слишком долго. Соответствующий предел задается в серверной переменной long_query_time. Если при запуске сервера была указана опция `--log-long-format`, то в данном журнале будут также фиксироваться запросы, в которых не используются индексы. Журнальный файл создается в каталоге данных, а его имя соответствует имени компьютера с добавлением расширения `-slow.` log.

С помощью данного журнала можно определить, какие запросы требуют оптимизации. Существует сценарий `mysqldumpslow`, позволяющий отобрать из файла самые медленные запросы. Применение этого сценария демонстрируется в листинге 24.12. В данном случае находятся три самых медленных запроса.

```
# mysqldumpslow -s at -t 3 red-slow.log

Reading mysql slow query log from red-slow.log
Count: 1  Time=1.00s (1s)  Lock=0.00s (0s)  Rows=0.0 (0),
httpd[httpd]@localhost
  DELETE FROM project_view WHERE Project=N AND User=N

Count: 1  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=1.0 (1),
httpd[httpd]@localhost
  SELECT count(*) FROM comment c, user u WHERE c.Project = N
AND c.User = u.ID AND c.Permission <> N

Count: 2  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=0.0 (0),
httpd[httpd]@localhost
  delete from session where LastAction < 'S'
```

Журнал обновлений

Журнал обновлений применялся в старых версиях MySQL и в настоящее время заменен двоичным журналом.

УСТРАНЕНИЕ ПОСЛЕДСТВИЙ КАТАСТРОФ

В этой главе...

- Проверка и восстановление таблиц
- Резервное копирование и восстановление

Глава 25

В этой главе рассказывается о том, как предотвратить катастрофу и как устранить ее последствия, если все же случилось непоправимое. Рассматриваются вопросы поиска повреждений в таблицах и их **восстановления**, а также методы создания резервных копий и последующей работы с ними.

Если база данных хранит важную информацию, опробуйте описанные в этой главе методики до того, как в них возникнет необходимость. Лучше подготовиться заранее, чем быть захваченным врасплох.

Проверка и восстановление таблиц

Повреждения в таблицах MyISAM происходят вследствие событий, которые невозможно избежать. Различные аппаратные сбои могут оказать самое непредсказуемое влияние на базу данных. Например, если жесткий диск выйдет из строя, данные окажутся полностью потерянными. Неожиданное выключение системы из-за сбоя питания может привести к тому, что изменения в таблицу будут внесены не полностью. Даже если уничтожить серверный процесс по команде `kill`, у него не будет возможности корректно завершить свою работу. Если найдена поврежденная таблица, потратьте время на выяснение причин, вызвавших повреждение. Вообще говоря, в MySQL таблицы редко оказываются поврежденными.

Существуют два способа проверки и восстановления таблиц. Первый из **них** — с помощью специальных инструкций, второй — с помощью утилиты `myisamchk`. Соответствующие инструкции называются CHECK TABLE, REPAIR TABLE и OPTIMIZE TABLE (см. главу 13, "Инструкции SQL"). Они достаточно удобны, поскольку выполняются в рамках серверного процесса. В этом смысле они ничем не отличаются, к примеру, от инструкции SELECT. Утилита myisamchk обладает рядом дополнительных возможностей, которые в ряде ситуаций оказываются весьма удобными. Она была описана в главе 14, "Утилиты командной строки".

Необходимость проверки таблицы может быть вызвана тем, что утилиты, обращающиеся к таблице, начинают себя странно вести. Например, вводимые запросы не завершаются или выдаются неожиданные сообщения об ошибках. Если при обращении к таблице возвращается номер ошибки, воспользуйтесь утилитой `perror`, которая отображает поясняющее сообщение, соответствующее данному номеру.

Частота проверок базы данных зависит от степени доверия к серверу. Разработчики MySQL рекомендуют делать это хотя бы раз в неделю, ноесли есть возможность выполнять процедуру проверки каждую ночь, то шансы на заблаговременное обнаружение ошибки возрастают. С помощью демона `cron` или программы-планировщика можно составить график проверок таким образом, чтобы они запускались в часы наименьшей активности системы. Сохраняйте результаты проверок в журнальном файле или направляйте их самому себе по электронной почте.

Возможно, имеет смысл изменить сценарий `safe_mysqld` таким образом, чтобы при запуске сервера выполнялись инструкции проверки таблиц. Файл, содержащий такие инструкции, задается с помощью опции `--init-file`. Если повреждения произошли из-за того, то сервер внезапно прекратил работу, они будут немедленно исправлены.

Обработчики таблиц, для которых поддерживаются транзакции, содержат код, позволяющий им восстанавливать таблицы на основании журнальных файлов при запуске сервера. На момент написания книги в MySQL не было средств ручной проверки и восстановления таблиц этих типов. Разработчики MySQL добавят соответствующие функции позднее. А пока что, если возникает сомнение в целостности таблицы, необходимо остановить и повторно запустить сервер, хоть это и очень неуклюжий прием.

Таблицы MyISAM снабжены флагом, указывающим на то, изменилось ли содержимое таблицы с момента последней проверки. Инструкция CHECK TABLE пропустит неизменные таблицы при наличии ключевого слова CHANGED. В утилите myisamchk соответствующий режим включается с помощью опции `--check-only-changed`. Особым образом помечаются также неправильно закрытые таблицы. Чтобы проверить только их, укажите флаг FAST (инструкция CHECK TABLE) или опцию `--fast` (утилита myisamchk).

По умолчанию утилита myisamchk ищет повреждения только в индексных файлах. В инструкции CHECK TABLE этот режим включается с помощью флага QUICK. Сама инструкция CHECK TABLE по умолчанию проверяет не только индексы, но и неправильные ссылки на удаленные записи. В утилите myisamchk этот режим включается с помощью опции `--medium-check`. Расширенный режим проверки задается флагом EXTEND и опцией `--extended-check`. В этом случае будут проверяться все индексируемыезначения.

Табличные проверки занимают много времени в случае крупных таблиц, особенно если у них много ключей. Стандартные режимы проверки в обоих методах обеспечивают вполне приемлемую производительность. Более быстрые проверки удобны, когда их нужно запускать регулярно, например по ночам. Расширенные проверки приходят на помощь, если повреждение таблицы очевидно, но обычные проверки его все равно не находят.

Б листинге 25.1 иллюстрируется процедура проверки и восстановления таблицы. В данном случае я создал небольшую таблицу и сымитировал в ней повреждение, отредактировав табличные файлы в редакторе `vi`. Как видите, таблицу удалось восстановить.

```
mysql> CHECK TABLE book;
+-----------+-------+----------+-----------------------------------------+
| Table     | Op    | Msg_type | Msg_text                                |
+-----------+-------+----------+-----------------------------------------+
| book      | check | error    | Size of indexfile is: 1924 Should be: 2048 |
| test.book | check | error    | Corrupt                                 |
+-----------+-------+----------+-----------------------------------------+
2 rows in set (0.00 sec)

mysql> REPAIR TABLE book;
+-----------+--------+----------+----------+
| Table     | Op     | Msg_type | Msg_text |
+-----------+--------+----------+----------+
| test.book | repair | status   | OK       |
+-----------+--------+----------+----------+
1 row in set (0.08 sec)
```

Таблицы можно проверять, когда сервер запущен. Программа MySQL не будет пытаться их восстановить. Но если обнаруживается поврежденная таблица, программа запрещает потокам обращаться к ней до тех пор, пока таблица не будет восстановлена. Для восстановления требуется получить монопольный доступ к таблице. В этой ситуации служебными инструкциями пользоваться удобнее, чем утилитой `myisamchk`, так как MySQL сможет заблокировать другие потоки на время восстановления таблицы. Утилита `myisamchk` может работать таким образом, только если операционная система поддерживает блокировку файлов. В Linux соответствующих функций нет, поэтому перед восстановлением таблиц нужно останавливать сервер.

Инструкция REPAIR TABLE устраняет повреждения в таблице. То же самое делает утилита myisamchk при наличии опции `--recover`. Программа MySQL поддерживает три типа процедур восстановления: быстрая, обычная и безопасная. В первом случае устраняются лишь проблемы с индексами. Во втором случае исправляется также большинство ошибок в табличном файле. В безопасном режиме таблица проверяется строка за строкой, а индексный файл создается заново. Это наиболее длительная процедура.

При удалении записей из таблицы программа MySQLсохраняет в ней пустые участки, которые повторно задействуются при последующем выполнении инструкций INSERT. Если таблица содержит пустые участки, то перед вставкой записей ее нужно заблокировать. Правда, когда записи вставляются в конец файла данных, программа разрешает другим потокам параллельно осуществлять чтение таблицы.

Таблицы с записями переменной длины неизбежно оказываются фрагментированными. Это происходит, когда обновляемая запись не помещается в отведенном для нее пространстве. В результате снижается производительность операций выборки, поскольку программа вынуждена искать запись в двух и более точках файла. Инструкция `OPTIMIZE TABLE` удаляет из таблицы пустые участки и осуществляет пересортировку записей. Аналогичные действия выполняет утилита myisamchk при наличии опции `--analyze`. Инструкция `OPTIMIZE TABLE` также сортирует индексы (соответствующая опция утилиты myisamchk называется `--sort-index`). Подробнее о процессе оптимизации рассказывается в главе 26, "Оптимизация".

Резервное копирование и восстановление

Резервная копия — это образ базы данных в конкретный момент времени. К этому образу можно вернуться в случае непредвиденной потери данных. Резервные копии можно создавать сколь угодно часто. Нужно лишь помнить о том, что это достаточно трудоемкий процесс, продолжительность которого зависит от размера базы данных и скоростных характеристик оборудования. В схеме репликации, которая описана в главе 29, "Распределенные базы данных", резервные копии создаются практически мгновенно.

Создание резервных копий требует от сервера значительных затрат ресурсов, вплоть до того, что работать с другими базами данных станет невозможно. Нужно спланировать этот процесс таким образом, чтобы он приходился на периоды минимальной загруженности сервера. Если используется репликация, то резервные копии лучше создавать на подчиненном сервере.

Если резервная копия была создана в полночь, а сбой базы данных произошел в полдень, половина дневных изменений окажется утерянной. Тем ценнее значимость **двоичного журнала**, о котором рассказывалось в главе 24, "Физическое хранение данных". В нем фиксируются все изменения базы данных. С помощью утилиты `mysqlbinlog` можно преобразовать содержимое этого файла в запросы к **восстановленной** базе данных, которые позволят воссоздать ее состояние на момент сбоя. Таким образом, планируя схему резервного копирования, не забудьте учесть ротацию и архивирование **двоичных журналов**, чтобы они были синхронизированы с копиями базы данных.

Помните общие правила обращения с резервными копиями. Если они хранятся в той же файловой системе, что и сама база данных, то данные не защищены от сбоев файловой системы. Отсюда правило: копии должны находиться на отдельном носителе. Храните их на перезаписываемом **компакт-диске**, магнитной ленте или другом жестком диске. Резервные копии могут храниться дома у начальника или администратора компании. Их можно также пересылать по сети в другую систему. Сегодня, в эпоху Internet, это делать не сложно.

В процессе планирования необходимо предусмотреть тестирование копий и проверку возможности их восстановления на практике. Не ждите, пока случится катастрофа и вам придется учиться восстанавливать архивы. Создайте тестовую среду и потренируйтесь на ней. Можно попробовать восстановить архив во временную пустую базу данных или же воспользоваться более сложной методикой, например запустить еще один сервер MySQL на другом порту либо на другом компьютере.

В MySQL существуют три основных способа архивирования данных. Первый — это копирование табличных файлов, второй — создание SQL-образов таблиц, третий — создание форматированных текстовых файлов. Первый способ является самым экономным и быстродействующим. Но для таблиц тех типов, которые поддерживают транзакции, последние два способа являются более гибкими. Например, все таблицы **InnoDB** хранятся в группе больших файлов, поэтому архивы нельзя будет сгруппировать по базам данных или таблицам.

Какой бы метод ни был выбран, не забудьте защитить таблицы от изменений на время резервного копирования. Если копируются табличные файлы, следует **остановить** сервер. В остальных случаях достаточно поставить блокировки чтения с помощью инструкций LOCK TABLES и выполнить инструкцию FLUSH TABLES. Последняя необходима для того, чтобы все изменения индексов были записаны в таблицы. На-

личие блокировок чтения позволит другим потокам параллельно обращаться к таблицам с запросами на выборку.

Инструкции BACKUP TABLE и RESTORE TABLE копируют табличные файлы в указанный каталог. Естественно, серверный процесс должен иметь право записи в этот каталог. Программа MySQL копирует туда файлы с расширениями .frm и .MYD. Индексный файл (.MYI) можно воссоздать на основании первых двух, что позволит сэкономить место в архиве. В листинге 25.2 показан пример архивирования таблицы.

```
mysql> BACKUP TABLE dictionary TO '/tmp/backup';
+-----------------+--------+----------+----------+
| Table           | Op     | Msg_type | Msg_text |
+-----------------+--------+----------+----------+
| test.dictionary | backup | status   | OK       |
+-----------------+--------+----------+----------+
1 row in set (0.27 sec)
```

Функции копирования файлов предоставляются операционной системой, поэтому данный способ создания резервных копий является самым быстрым. Таблица dictionary, скопированная в листинге 25.2, содержит более 100000 записей, а файл данных занимает почти 3 Мбайт. Как видите, процедура архивирования такой таблицы заняла менее секунды.

Инструкция BACKUP TABLE самостоятельно заботится о блокировании таблиц и очистке табличных буферов. Это означает, что, в отличие от других методов резервного копирования, дополнительные инструкции не нужны.

Инструкция RESTORE TABLE копирует архивные файлы в каталог базы данных и перестраивает индексы. Таблица не должна существовать на момент восстановления. В случае необходимости можно удалить ее с помощью инструкции DROP TABLE или же вручную удалить табличные файлы.

В листинге 25.3 показаны результаты восстановления таблицы dictionary, резервная копия которой была создана в листинге 25.2. Обратите внимание на то, что процесс восстановления длился гораздо дольше, чем архивирование. Это объясняется тем, что на перестройку индексов уходит много времени.

```
mysql> RESTORE TABLE dictionary FROM '/tmp/backup';
+-----------------+---------+----------+----------+
| Table           | Op      | Msg_type | Msg_text |
+-----------------+---------+----------+----------+
| test.dictionary | restore | status   | OK       |
+-----------------+---------+----------+----------+
1 row in set (1 min 22.24 sec)
```

Если резервные копии создаются вручную, то в архив можно также включить индексный файл. В этом случае в процессе восстановления таблицы индексный файл будет просто скопирован в каталог базы данных. Тем не менее его всегда можно воссоз-

дать с помощью инструкции REPAIR TABLE. Предположим, таблица dictionary была полностью утеряна. Процесс ее восстановления начнем с копирования frm-файла обратно в каталог базы данных. Создать пустые файлы данных и индексов можно с помощью инструкции TRUNCATE TABLE. Затем необходимо скопировать старый файл данных поверх нового. После этого вводится инструкция REPAIR TABLE. В листинге 25.4 показано, как программа MySQL обнаруживает расхождение в количестве записей и перестраивает индексы.

```
mysql> REPAIR TABLE dictionary;
+------------+--------+----------+----------------------------------------+
| Table      | Op     | Msg_type | Msg_text                               |
+------------+--------+----------+----------------------------------------+
| state      | repair | warning  | Number of rows changed from 0 to 104237 |
| test.state | repair | status   | OK                                     |
+------------+--------+----------+----------------------------------------+
2 rows in set (1min 23.41 sec)
```

Для безопасного создания резервных копий лучше пользоваться специальной программой, чем делать все вручную. С этой целью в дистрибутив MySQL входит Perl-сценарий mysqlhotcopy. В листинге 25.5 показано, как с его помощью создаются копии таблиц привилегий. Команда ls позволяет убедиться, что все файлы, в том числе индексные, на месте.

```
# mysqlhotcopy mysql /trap/he
Locked 6 tables in 0 seconds.
Flushed tables (mysql.columns_priv, mysql.db, mysql.func, mysql.host,
mysql.tables_priv, mysql.user) in 0 seconds.
Copying 18 files...
Copying indices for 0 files...
Unlocked tables.
mysqlhotcopy copied 6 tables (18 files) in 1 second (1 seconds
overall).
# ls /tmp/hc/mysql
columns_priv.MYD db.MYD func.MYD host.MYD tables_priv.MYD user.MYD
columns_priv.MYI db.MYI func.MYI host.MYI tables_priv.MYI user.MYI
columns_priv.frm db.frm func.frm host.frm tables_priv.frm user.frm
```

Параметры сценария mysqlhotcopy описывались в главе 14, "Утилиты командной строки". Он блокирует одновременно все таблицы базы данных, после чего очищает табличные буферы и копирует файлы. Сценарий можно запускать во время работы сервера, даже если в этот момент пользователи делают запросы к базе данных. Естественно, пока происходит копирование таблиц, пользователям будет запрещено вносить в них изменения.

Формат табличных файлов понятен только программе MySQL. Если же создать SQL-образы таблиц, то их можно будет перенести в другие СУБД. Кроме того, в некоторых ситуациях полезно просматривать такие SQL-инструкции. Предположим, к

примеру, что потеря данных оставалась незамеченной на протяжении нескольких месяцев. Возможно, пользователи удалили какие-то записи и лишь позднее обнаружили, что это было сделано неправильно. Нужно восстановить только удаленные записи, но не известно, когда точно они были удалены. Если резервные копии хранятся в формате SQL, можно просмотреть архивы и поискать, когда последний раз встречались требуемые записи. Недостатком такого способа резервного копирования является то, что процедура восстановления занимает много времени, поскольку программа MySQL вынуждена выполнять каждую инструкцию из архива.

Для создания **SQL-образа** таблицы предназначена утилита `mysqldump`. Она записывает текст инструкций в поток `stdout`, поэтому нужно перенаправить результаты ее работы в файл. Параметры этой утилиты были описаны в главе 14, "Утилиты командной строки". В листинге 25.6 показан созданный этой утилитой образ таблицы `db` из базы данных `mysql`. Утилита была запущена с опцией `--opt`, которая включает режим оптимальных установок.

```
# MySQL dump 8.13
#
# Host: localhost    Database: mysql
#
# Server version        3.23.39-log

#
# Table structure for table 'db'
#

DROP TABLE IF EXISTS db;
CREATE TABLE db  (
  Host char(60) binary NOT NULL default  '',
  Db char(64) binary NOT NULL default  '',
  User char(16) binary NOT NULL default  '',
  Select_priv enum('N','Y') NOT NULL default 'N',
  Insert_priv enum('N','Y') NOT NULL default 'N',
  Update_priv enum('N','Y') NOT NULL default 'N',
  Delete_priv enum('N','Y') NOT NULL default 'N',
  Create_priv enum('N','Y') NOT NULL default 'N',
  Drop_priv enum('N','Y') NOT NULL default 'N',
  Grant_priv enum('N', 'Y') NOT NULL default 'N',
  References_priv enum('N','Y') NOT NULL default 'N',
  Index_priv enum('N','Y') NOT NULL default  'N',
  Alter_priv enum('N','Y') NOT NULL default 'N',
  PRIMARY KEY  (Host,Db,User),
  KEY User  (User)
) TYPE=MyISAM COMMENT='Database privileges';

tt
# Dumping data for table 'db'
#

LOCK TABLES db WRITE;
INSERT INTO db VALUES
('%','test','','Y','Y','Y','Y','Y','Y','N','Y','Y','Y'),('%',
```

```
'test\\_%','','Y','Y','Y','Y','Y','Y','N','Y','Y','Y'),(
'localhost','freetime','httpd','Y','Y','Y','Y','N','N','N',
'N','N','N');
UNLOCK TABLES;
```

Не забудьте заблокировать все таблицы для чтения, прежде чем запускать утилиту mysqldump. В противном случае целостность результатов не гарантируется. Предположим, имеется приложение, которое хранит информацию о клиентах и их электронных адресах. Создание учетной записи нового клиента включает добавление записи в таблицу client и последующую вставку одной или нескольких записей в таблицу email_address. Если параллельно с этим создавать резервную копию базы данных, то может оказаться, что в промежутке между созданием образов таблиц client и email_address приложение попытается обновить обе эти таблицы. Доступ к первой таблице будет запрещен, а ко второй — нет. В результате в архиве появятся адреса, не соответствующие ни одной записи таблицы клиентов.

Чтобы восстановить данные из такого архива, достаточно выполнить SQL-сценарий в интерпретаторе mysql. Можно просто перенаправить сценарий на вход этой утилиты или же воспользоваться ее командой source. Интерпретатор выполнит все инструкции сценария так, как если бы они были введены в командной строке.

Утилита mysqldump имеет режим создания текстового образа таблицы. В этом режиме для каждой архивируемой таблицы создаются два файла. Один из них имеет расширение .sql и содержит соответствующую инструкцию CREATE TABLE. Второй файл имеет расширение .txt и содержит записи таблицы, причем для разделения полей применяются символы табуляции. В листинге 25.7 показана команда, создающая текстовый образ таблицы dictionary в каталоге /tmp.

```
[/tmp]# mysqldump -verbose -tab=/tmp test dictionary
# Connecting to localhost...
# Retrieving table structure for table dictionary...
# Sending SELECT query...
# Disconnecting from localhost...
```

Для восстановления данных из такого архива необходимо сначала создать *таблицу,* а затем выполнить инструкцию LOAD DATA INFILE, которая вставит записи в таблицу. Стандартный формат файла, создаваемого утилитой mysqldump, соответствует тому формату, который по умолчанию распознается инструкцией LOAD DATA INFILE. В листинге 25.8 демонстрируется загрузка данных в таблицу dictionary в среде mysql.

```
mysql> source /tmp/dictionary.sql
Query OK, 0 rows affected (0.00 sec)

Query OK, 0 rows affected (0.10 sec)
```

```
mysql> LOAD DATA INFILE '/tmp/dictionary.txt'
    ->      INTO TABLE dictionary;
Query OK, 104237 rows affected (1 min 27.70 sec)
Records: 104237  Deleted: 0  Skipped: 0  Warnings: 0

mysql>
```

Создать файл, понимаемый инструкцией LOAD DATA INFILE, позволяет также инструкция SELECT с предложением INTO (листинг 25.9). Схему таблицы необходимо получить другим путем, например с помощью инструкции SHOW CREATE TABLE.

```
mysql> SELECT *
    ->   FROM dictionary
    ->   INTO OUTFILE '/tmp/dictionary.txt';
Query OK, 104237 rows affected (6.42 sec)
```

Один из способов восстановления таблиц заключается в использовании двоичного журнала. Достаточно преобразовать его содержимое в SQL-инструкции и выполнить их. Предварительно необходимо заблокировать все таблицы для записи или отключить всех клиентов от сервера. Преобразование двоичного журнала осуществляется с помощью утилиты mysqlbinlog (листинг 25.10). Результаты ее работы нужно направить в файл или интерпретатору mysql. Обратите внимание: инструкция SET меняет метку текущего времени сеанса, чтобы дата создания таблицы осталась неизменной.

```
# mysqlbinlog -offset=1 -short-form red-bin.001
use freetime;
SET TIMESTAMP=991767105;
UPDATE session SET LastAction = now() WHERE ID='fNbbnOLBYY1qesga';
use freetime;
SET TIMESTAMP=991767134;
UPDATE session SET LastAction = now() WHERE ID='fNbbnOLBYY1qesga';
use freetime;
SET TIMESTAMP=991767134;
DELETE FROM project_view WHERE Project=2 AND User=2;
use freetime;
SET TIMESTAMP=991767135;
INSERT INTO project_view VALUES (2, 2, now());
```

ОПТИМИЗАЦИЯ

В этой главе...

Предварительные действия
Тесты производительности
Оптимизация проекта
Оптимизация приложений
Оптимизация запросов
Оптимизация инструкций
Обслуживание таблиц
Настройка конфигурации сервера
Перекомпиляция MySQL

Глава 26

Оптимизация — это процесс тонкой настройки системы, направленный на повышение скорости ее работы или сокращение объема используемой памяти. В первой части главы объясняется, когда и как нужно оптимизировать базы данных. Бинарные дистрибутивы, доступные на Web-узле MySQL, оптимизированы для общего применения. Чтобы адаптировать программу к каким-то специфическим требованиям, ее необходимо перекомпилировать. Об этом и пойдет речь во второй части главы.

Предварительные действия

Перед началом проектирования базы данных поставьте себе задачу добиться максимальной ясности спецификации, даже если на это уйдет больше времени. Помните о том, что услуги программистов стоят дорого, особенно если им проходится разбираться с малопонятным проектом. Простое решение обычно является наилучшим.

Перенося базу данных в производственную среду, позаботьтесь о том, чтобы производительность базы данных была адекватной. Если к проекту прилагается формальная спецификация требований, просмотрите, указываются ли в ней какие-либо ограничения производительности. Для приложений, работающих с базами данных, нередко задается максимальное время выполнения запросов. Продолжительность времени между вводом инструкции и получением результатов запроса зависит от многих факторов. Необходимо заранее учесть те факторы, которые впоследствии нельзя будет контроли ровать.

Если обнаруживается, что система требует оптимизации, в первую очередь подумайте об обновлении аппаратной части. Это может оказаться самым дешевым вариантом. В 1965 г. Гордон Мур (Gordon Moore) установил, что вычислительные мощности удваиваются каждые 18 месяцев. Данное правило называют законом Мура. Но, несмотря на столь стремительный рост производительности, удельная стоимость вычислительных средств неуклонно снижается. Например, центральные процессоры

за полтора года удвоят тактовую частоту, хотя стоить будут так же, как и полтора года назад. Таким образом, обновление компьютера может обойтись дешевле, чем оптимизация проекта.

Во вторую очередь стоит подумать об обновлении программного обеспечения. Основной программный компонент— это операционная система. Известно, что Linux и BSD UNIX позволяют повысить производительность старых компьютеров, превосходя в этом отношении коммерческие операционные системы, такие как Windows, особенно если бессбойная работа сервера очень важна.

Обновляется и сама программа MySQL. Когда появится версия 4, ее производительность будет повышена в сравнении с третьей версией. Но и в третью версию регулярно вносят мелкие исправления, так что желательно идти в ногу со временем.

Основная причина оптимизации — желание сэкономить деньги (оставим в стороне личное удовлетворение и другие причины). Не забывайте об этом в своих попытках повысить производительность программы. Нет смысла затрачивать на оптимизацию больше денег, чем она способна принести. Стоит потрудиться над такой программой, с которой работает множество людей, особенно если это коммерческое приложение. Что касается программ с открытыми кодами, то важность оптимизации здесь трудно определить. Лично я рассматриваю работу над такими проектами как хобби.

Чтобы процесс оптимизации был максимально эффективным, сосредоточьте усилия на самой медленной части программы, улучшение которой обеспечит наибольшую отдачу. Обычно пытаются найти более быстрые альтернативы применяемым алгоритмам. В вычислительной технике относительная эффективность алгоритма записывается в нотации "большого O". Например, запись $O(n)$ означает, что время выполнения алгоритма пропорционально числу обрабатываемых элементов n. Алгоритм типа $O(n)$ является очень медленным. Проанализируйте используемые в программе алгоритмы и подумайте, что можно сделать для их улучшения.

Тесты производительности

Прежде чем приступать к оптимизации, нужно вооружиться средствами измерения производительности. Предусмотрительные разработчики MySQL написали группу Perl-сценариев, предназначенных для тестирования производительности MySQL и других СУБД. Эти сценарии расположены в каталоге `sql-bench` исходного дистрибутива. В подкаталоге Results находятся результаты множества тестов существующих систем, которые можно сравнить с собственными оценками.

В сценариях используется демонстрационная базаданных, в которой выполняется восемь различных тестов. Эта база данных называется `test` и инсталлируется вместе с MySQL. Сценарий `run-all-tests` запускает все тесты последовательно. При наличии опции `--log` результаты работы сценария будут сохранены в каталоге `output` для последующего просмотра. Ниже приведена команда, запускающая тесты из каталога sql-bench.

```
# ./run-all-tests --user=leon --password=secret --log
```

Для работы этого сценария необходимо наличие в системе интерпретатора Perl и модуля DBI.

Для экспериментов я использую старый компьютер Pentium с частотой 100 МГц, работающий под управлением RedHat Linux. Несмотря на слабую вычислительную мощность, программа MySQL демонстрирует нанем вполне приемлемую производительность. Кроме того, ограниченные возможности системы позволяют быстро выявлять неэффективные программные решения. Результаты тестов, полученные на этом компьютере, показаны в листинге 26.1. Несложно убедиться, что моя система работает примерно в 10 раз медленнее, чем самая медленная из систем, результаты тестов которых имеются в каталоге Results. Если бы такую производительность продемонстрировал рабочий сервер, нужно было бы немедленно обновить его аппаратную часть.

```
The result logs which were found and the options:
 1 mysql-Linux_2.2.16_22_i586              : MySQL 3.23.39

================================================
Operation                          |          1|
                                   |    mysql-L|
------------------------------------------------
Results per test in seconds:                    |
------------------------------------------------
ATIS                               |     549.00|
alter-table                        |    4836.00|
big-tables                         |     270.00|
connect                            |     607.00|
create                             |    2027.00|
insert                             |  +88846.00|
select                             |  +18339.00|
wisconsin                          |     135.00|
------------------------------------------------
The results per operation:                     I
------------------------------------------------
alter_table_add (992)              |    2670.00|
alter_table_drop (496)             |    2066.00|
connect (10000)                    |      95.00|
connect+select_1_row (10000)       |     115.00|
connect+select_simple (10000)      |     106.00|
count (100)                        |     364.00|
count_distinct (1000)              |    +779.00|
count_distinct_2 (1000)            |    +605.00|
count_distinct_big (120)           |    1185.00|
count_distinct_group (1000)        |    +726.00|
count_distinct_group_on_key (1000) |    +716.00|
count_distinct_group_on_key_parts (1|    +708.00|
count_distinct_key_prefix (1000)   |    +671.00|
count_group_on_key_parts (1000)    |     706.00|
count_on_key (50100)               |   +7005.00|
create+drop (10000)                |     109.00|
create_MANY_tables (10000)         |    1196.00|
create_index (8)                   |      52.00|
create_key+drop (10000)            |     115.00|
create table (31)                  |       0.00|
```

```
delete_all (12)                          |     219.00|
delete_all_many_keys (1)                 |    8960.00|
delete_big (1)                           |       5.00|
delete_big_many_keys (128)               |    8956.00|
delete_key (10000)                       |     145.00|
drop_index (8)                           |      47.00|
drop_table (28)                          |       0.00|
drop_table_when_MANY_tables (10000)      |     105.00|
insert (350768)                          |    1044.00|
insert_duplicates (100000)               |     211.00|
insert_key (100000)                      |   17849.00|
insert_many_fields (2000)                |      97.00|
insert_select_1_key (1)                  |     102.00|
insert_select_2_keys (1)                 |     125.00|
min_max (60)                             |     301.00|
min_max_on_key (85000)                   |   +2627.00|
multiple_value_insert (100000)           |     102.00|
order_by_big (10)                        |     560.00|
order_by_big_key (10)                    |     503.00|
order_by_big_key2 (10)                   |     479.00|
order_by_big_key_desc (10)               |     531.00|
order_by_big_key_diff (10)               |     586.00|
order_by_big_key_prefix (10)             |     487.00|
order_by_key2_diff (500)                 |      57.00|
order_by_key_prefix (500)                |      31.00|
order_by_range (500)                     |      48.00|
outer_join (10)                          |     989.00|
outer_join_found (10)                    |     918.00|
outer_join_not_found (500)               |  +36000.00|
outer_join_on_key (10)                   |     810.00|
select_1_row (10000)                     |      17.00|
select_2_rows (10000)                    |      23.00|
select_big (10080)                       |     700.00|
select_column+column (10000)             |      24.00|
select_diff_key (500)                    |   +1470.00|
select_distinct (800)                    |     145.00|
select_group (2925)                      |    +896.00|
select_group_when_MANY_tables (10000     |     502.00|
select_join (100)                        |      25.00|
select_key (200000)                      |   +1123.00|
select_key2 (200000)                     |   +1213.00|
select_key2_return_key (200000)          |   +1174.00|
select_key2_return_prim (200000)         |   +1197.00|
select_key_prefix (200000)               |   +1245.00|
select_key_prefix_join (100)             |     197.00|
select_key_return_key (200000)           |   +1106.00|
select_many_fields (2000)                |     172.00|
select_query_cache (10000)               |    1075.00|
select_query_cache2 (10000)              |    1075.00|
select_range (410)                       |   +1812.00|
select_range_key2 (25010)                |     185.00|
select_range_prefix (25010)              |     197.00|
select_simple (10000)                    |      11.00|
select_simple_join (500)                 |      20.00|
update_big (10)                          |     440.00|
update_of_key (50000)                    |     627.00|
```

```
update_of_key_big (501)              |      351.00|
update_of_primary_key_many_keys (256|     3290.00|
update_with_key (300000)             |     1011.00|
update_with_key_prefix (100000)      |      290.00|
wisc_benchmark (114)                 |       44.00|
-----------------------------------------------
TOTALS                               | +124540.00|
```

Сценарий compare-results суммирует и сравнивает результаты тестов. В листинге 26.1 приведен лишь один набор результатов. В действительности я немного сократил выходные данные, удалив ряд малозначащих пояснений. В первом блоке чисел указано время выполнения каждого из восьми тестов в секундах. Во втором блоке отображается статистика отдельных операций по всем тестам. Числа со знаком "плюс" – это приблизительные оценки, полученные для тестов, время выполнения которых превысило максимум.

Результаты тестов, предоставляемые разработчиками MySQL, можно использовать для выбора аппаратной платформы и операционной системы. На Web-узле MySQL *(unvw.mysql.com/information/benchmarks.html)* постоянно публикуются обновляемые результаты и графики сравнения показателей MySQL с показателями других СУБД, работающих на идентичном оборудовании. Приводятся также данные, касающиеся работы MySQL на разных платформах.

Конечно, все эти тесты отражают лишь относительную производительность сервера. С их помощью можно узнать, насколько возрастет скорость его работы при изменении тех или иных настроек, но они не могут помочь в оптимизации базы данных. Для оценки производительности запросов необходимо воспользоваться инструкцией EXPLAIN (см. главу 13, "Инструкции SQL"). Эта инструкция, помимо всего прочего, сообщает о том, сколько записей будет прочитано при выполнении заданной инструкции SELECT. Каждая строка результатов соответствует одной исходной таблице, а порядок строк совпадает с порядком обращения к таблицам. Сообщаемое число записей может быть приблизительным, но погрешность очень мала. Произведение счетчиков записей является грубым критерием производительности запроса. Чем меньше это произведение, тем быстрее выполняется запрос.

Представим себе, к примеру, объединение таблицы, содержащей 15000 слов, с таблицей, содержащей 100000 слов. В худшем случае программе MySQL придется просмотреть все записи обеих таблиц. Сначала выбирается первая запись первой таблицы, а затем начинается просмотр записей второй таблицы до тех пор, пока не будет найдено совпадение. Умножив 15000 на 100000, получим 1,5 миллиарда операций чтения. На практике это числооказывается немного меньшим, но и его достаточно, чтобы получить представление о скорости запроса. Далее в главе будет рассказываться о том, как с помощью индексов уменьшить количество записей, читаемых в процессе объединения таблиц.

С помощью журнала медленных запросов, описанного в главе 24, "Физическое хранение данных", можно легко найти наименее эффективные запросы. В дистрибутив MySQL входит сценарий `mysqldumpslow`, предназначенный для упорядочения записей этого журнала по указанному в них времени выполнения запроса.

Оптимизация проекта

Давайте вспомним то, о чем говорилось в главе 8, "Нормализация". *Нормализация* — это такой метод оптимизации базы данных, при котором избыточность хранящейся в ней информации оказывается минимальной. Следовательно, уменьшается время, затрачиваемое приложением на поддержание целостности базы данных. Нормализация достигается за счет повышения объема работы, выполняемой сервером, так как увеличивается число таблиц и серверу приходится чаще создавать их объединения. В процессе *денормализации* в базу данных вносят некоторую избыточность, для того чтобы сократить объем работы по извлечению информации.

Наиболее эффективный тип денормализации включает создание итоговых данных. Под этим может подразумеваться добавление к таблице столбца, хранящего результаты вычислений по другим столбцам. Например, если в таблице накапливаются данные о прохождении грузов, то в ней будут столбцы с указанием времени прибытия и отбытия груза. Чтобы не вычислять каждый раз время стоянки, можно посчитать его один раз и занести результат в отдельный столбец. Управлять подобной избыточностью несложно.

Иногда создают не просто итоговые столбцы, а целые таблицы. Например, можно сохранять результаты ключевых запросов в таблице, которая обновляется раз в день. Это избавит сервер от необходимости все время выполнять одни и те же трудоемкие запросы, хотя и повысит риск получения пользователями неактуальных данных.

Если таблицы содержат часто изменяемую информацию, лучше делать их резидентными. Такие таблицы хранятся в памяти и уничтожаются при перезагрузке сервера. Приложение должно быть готово к возможному отсутствию таблицы и должно уметь воссоздавать ее в случае необходимости. Хороший пример — Web-приложение, хранящее параметры сеанса в базе данных.

Реляционные базы данных хорошо работают с типизированными значениями фиксированного размера. В MySQL поддерживаются типы переменной длины, например BLOB и TEXT, но управлять ими сложнее. Такого рода информацию лучше хранить в файлах, а в базе данных достаточно запоминать путевые имена этих файлов в столбцах типа CHAR. Если база данных используется в Web-приложениях, помните о том, что у Web-сервера есть **кэш-буферы** загружаемых файлов изображений и аудиоклипов, поэтому он будет работать с такими файлами быстрее, чем MySQL.

Еще одна причина избегать столбцов подобного типа заключается в появлении записей переменной длины со всеми вытекающими отсюда последствиями. При внесении изменений такая таблица становится **фрагментированной**, что приводит к замедлению доступа к ней. Для извлечения динамической строки может потребоваться несколько операций чтения, что также не способствует повышению производительности. О форматах хранения табличных данных рассказывалось в главе 24, "Физическое хранение данных".

Монти **Видениус** рассказал мне правило определения того, когда следует использовать столбцы типа VARCHAR, а когда — CHAR. Если в таблице есть столбцы типа BLOB или TEXT, то предпочтение отдается типу VARCHAR, потому что все записи таблицы будут динамическими. То же самое справедливо для случая, когда средняя размерность значений столбца не превышает половины его размерности. Например, столбец типа `VARCHAR(80)`, средняя размерность которого равна 10 символам, определен правильно. Если же средняя размерность превышает 40 символов, нужно поменять

тип столбца на CHAR(80). Данное правило направлено на оптимизацию скорости работы с таблицами. Когда более важным фактором является экономия дискового пространства, то в большинстве случаев следует пользоваться типом VARCHAR.

Для таблиц MyISAM поддерживается опция DELAY_KEY_WRITE. Она заставляет программу хранить изменения табличных индексов в памяти, пока таблица не будет закрыта. Это сокращает время записи на диск измененных табличных данных, но также повышает риск повреждения таблицы в случае сбоя сервера. Если используется данная опция, то при каждом перезапуске сервера необходимо проверять таблицы на предмет повреждений.

Процедура analyse() представляет собой удобное средство проверки таблицы после вставки данных, так как она определяет диапазон значений каждого столбца в полученном наборе записей. Ее нужно использовать в инструкции SELECT, которая извлекает все записи отдельной таблицы. На основании анализа таблицы процедура analyse() предложит оптимальный тип данных для каждого столбца.

В некоторых случаях процедура analyse() сообщает о том, что вместо типа CHAR должен применяться тип ENUM. Это происходит, когда столбец содержит небольшое число повторяющихся значений. Столбец типа ENUM занимает гораздо меньше места, поскольку в действительности он хранит лишь номера элементов перечисления.

Многие типы данных допускают регулирование своей размерности. Например, в столбце типа CHAR может храниться столько уникальных значений, что приводить его к типу ENUM нет никакого смысла, и все равно формальная размерность оказывается избыточной. То же самое касается типа INT, у которого существуют более "короткие" эквиваленты: MEDIUMINT, SMALLINT и TINYINT. Но не забудьте учесть будущее пополнение таблицы. Например, если в таблице 16000 записей, то для первичного ключа вполне подойдет тип SMALLINT. Если же предполагается, что в таблице будет более 65535 записей, следует остановиться на типе INT.

Обратите внимание на столбцы, в которых не могут присутствовать значения NULL. Для экономии места такие столбцы нужно объявлять со спецификатором NOT NULL. Числовые столбцы, в которых не могут храниться отрицательные числа, должны иметь спецификатор UNSIGNED.

Оптимизация приложений

Подключение к базе данных MySQL происходит относительно быстро в сравнении с другими СУБД, но это время можно еще уменьшить за счет кэширования соединений. Требуется лишь прикладная среда, позволяющая хранить идентификаторы соединений в памяти во время работы сервера. Например, модуль PHP непрерывно работает на Web-сервере. Он поддерживает функцию mysql_pconnect(), которая создает постоянные соединения. Получив запрос на подключение к серверу, модуль PHP попытается использовать существующее соединение, если это возможно. В протоколах JDBC и ODBC тоже применяется технология кэширования соединений. Она особенно удобна, когда приложение создает большое число соединений за короткий промежуток времени.

Кэшируется и другая информация. К примеру, если приложение вставляет данные в таблицу, можно предварительно помещать данные в буфер, с тем чтобы позднее занести их в таблицу в пакетном режиме. В этом случае лучше сразу же заблокировать таблицу, чтобы не пришлось многократно обновлять табличные индексы.

Приложение может дотировать информацию, извлекаемую из базы данных. Это выгодно, если данные меняются нечасто. Когда изменение данных все же происходит, приложение запрашивает принудительную очистку буфера. Предположим, что в Internet-магазине имеется каталог продаваемых товаров. Этот каталог пополняется или обновляется раз в неделю, а то и меньше. Когда приложение отправляет клиенту HTML-страницу с описанием товара, оно вполне может взять информацию из кэша. Если администратор захочет воспользоваться приложением для обновления цены товара, он должен будет очистить кэш.

То же самое применимо и в отношении программных блоков. Если нужно узнать название, цену и категорию товара, введите один запрос и сохраните полученные значения в программных переменных. Основная работа по выборке данных заключается в поиске нужной записи. Не имеет особого значения, 100 или 1000 байтов извлекаются из нее.

Оптимизация запросов

Незаметно для пользователей программа MySQL оптимизирует предложения WHERE инструкции SELECT. Обычно не нужно заботиться о том, сколько скобок указано в выражении или каков порядок таблиц в объединении. Вместо этого сосредоточьтесь на индексах. Они позволяют ускорить операции выборки данных за счет замедления операций записи. Конечно, индексы занимают дополнительное место на диске, но они незаменимы с точки зрения эффективной организации таблиц.

Когда программа MySQL извлекает данные из таблицы, ей достаточно просмотреть один индексный столбец, чтобы найти нужные записи и не сканировать всю таблицу. Если к объединенной таблице применимы два индекса, программа выбирает из них тот, который позволит прочесть меньшее число записей.

Разрешается создавать индекс, охватывающий несколько столбцов. Программа MySQL может работать с частями индекса, но они должны просматриваться строго слева направо. Например, если индекс включает столбцы имени и фамилии, то при обращении к первому столбцу индекс будет использован, а ко второму — нет (при условии, что перед этим не было обращения к первому столбцу). Это правило применимо и к символам индексируемого столбца, содержащего текстовые данные (тип CHAR, VARCHAR или BLOB). Когда в предложении WHERE присутствует оператор LIKE, индекс задействуется лишь в том случае, если шаблон сравнения содержит все литеральные символы слева, а метасимволы — справа. Так, шаблон `'abc%'` разрешает использование индекса, а шаблон `'abc%xyz'` — нет.

В листинге 26.2 приведены инструкции, создающие две таблицы. Таблица word будет содержать 14346 записей, а таблица dictionary — 104237. В первую таблицу слова заносятся пользователями, а вторая таблица содержит список известных программе слов. Пользователи часто вводят несуществующие слова. Запрос, анализируемый в листинге 26.3, предназначен для выяснения количества распознанных слов. Условию отбора соответствуют 911 записей.

```
mysql> EXPLAIN SELECT word.word, dictionary.word
    ->            FROM word LEFT JOIN dictionary
    ->               ON word.word=dictionary.word
    ->            WHERE word.class =  '_VERBO'  \G
*************************** 1.row ***************************
        table: word
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 14346
        Extra: where used

        table: dictionary
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
```

```
          ref: NULL
         rows: 104237
        Extra:
2 rows in set (0.00 sec)
```

В запросе участвуют три столбца: столбцы word и class таблицы word и столбец word таблицы dictionary. Известно, что тестовому условию отбора соответствуют 911 записей таблицы word, поэтому наша задача состоит в том, чтобы сократить диапазон сканирования первой таблицы до соответствующего уровня. Для этого необходимо создать индекс по столбцу class. Сначала я планировал включить в индекс только упомянутый столбец, но потом подумал о других запросах, которые приходится направлять таким таблицам. Я, например, часто создаю отчет, в который включается все содержимое таблицы, отсортированное сначала по классам, а затем — по словам. Разумнее будет включить в индекс сразу два столбца (листинг 26.4).

```
ALTER TABLE word
  ADD INDEX (class, word)
```

Теперь инструкция EXPLAIN выдает другие результаты (листинг 26.5). В поле key_len сообщается о том, что индекс охватывает 16 символов столбца class. По оценке программы MySQL, ей придется просмотреть 1517 записей, хотя мы знаем, что их всего 911.

```
mysql> EXPLAIN SELECT word.word, dictionary.word
    ->           FROM word LEFT JOIN dictionary
    ->             ON word.word=dictionary.word
    ->          WHERE word.class = '_VERBO' \G

         table: word
          type: ref
possible_keys: class
           key: class
       key_len: 16
           ref: const
          rows: 1517
         Extra: where used; Using index

         table: dictionary
          type: ALL
possible_keys: NULL
           key: NULL
       key_len: NULL
           ref: NULL
          rows: 104237
         Extra:
2 rows in set (0.00 sec)
```

Итак, появление индекса привело к сокращению диапазона сканирования в 15 раз, но инструкция все ее вынуждена просматривать 45 миллионов записей. Осталось еще учесть столбцы word в обеих таблицах. Разберемся сначала с таблицей word.

В процессе объединения таблиц программа MySQL использует не более одного индекса от каждой таблицы. Если появляются варианты, то выбирается индекс с более узким диапазоном. Созданный нами индекс уже охватывает столбец word, к тому же, как видно из листинга 26.5, диапазон поиска существенно сузился.

Теперь перейдем к таблице dictionary. Пока что инструкция SELECT вынуждена сканировать ее целиком. Добавление индекса к столбцу word позволит программе сразу же находить нужную запись (листинг 26.6).

```
ALTER TABLE dictionary
  ADD INDEX (word)
```

Эффект этого действия продемонстрирован в листинге 26.7. Как видите, количество просматриваемых записей таблицы dictionary сократилось до одной!

```
mysql> EXPLAIN SELECT word.word, dictionary.word
    ->           FROM word LEFT JOIN dictionary
    ->             ON word.word=dictionary.word
    ->          WHERE word.class = '_VERBO' \G
*************************** 1. row ***************
        table: word
         type: ref
possible_keys: class
          key: class
      key_len: 16
          ref: const
         rows: 1517
        Extra: where used; Using index

        table: dictionary
         type: ref
possible_keys: word
          key: word
      key_len: 64
          ref: word.word
         rows: 1
        Extra: Using index
2 rows in set (0.26 sec)
```

Оптимизация инструкций

От оптимизации больше всего выигрывают запросы на выборку, но существуют также методики повышения эффективности других инструкций, в частности INSERT. Можно избежать затрат времени на анализ инструкции, если воспользоваться пре-

имуществами значений по умолчанию. Вместо того чтобы указывать значения всех столбцов, задайте лишь те из них, которые отличаются от стандартных установок, а остальное пусть сделает MySQL. Сказанное иллюстрирует листинг 26.8, в котором показаны определение таблицы и инструкция `INSERT`.

```
CREATE TABLE address (
  ID int(11) NOT NULL AUTO_INCREMENT,
  Name_Prefix CHAR(16) default NULL,
  Name_First CHAR(32) default NULL,
  Name_Middle CHAR(32) default NULL,
  Name_Last CHAR(64) NOT NULL default '',
  Name_Suffix CHAR(16) default NULL,
  Company CHAR(64) default NULL,
  Street1 CHAR(64) default NULL,
  Street2 CHAR(64) default NULL,
  Street3 CHAR(64) default NULL,
  City CHAR(64) NOT NULL default '',
  StateProv CHAR(64) NOT NULL default '',
  PostalCode CHAR(16) NOT NULL default '',
  CountryCode CHAR(2) default NULL,
  Phone1 CHAR(32) default NULL,
  Phone2 CHAR(32) default NULL,
  Fax CHAR(32) default NULL,
  Email CHAR(64) NOT NULL default '',
  PRIMARY KEY (ID)
);

INSERT INTO address (Name_First, Name_Last)
    VALUES ('Leon', 'Atkinson');
```

По умолчанию операции записи имеют приоритет над операциями чтения, но программа MySQL не прервет выполнение инструкции SELECT, если в очереди вдруг появится инструкция INSERT. Последняя окажется заблокированной до тех пор, пока инструкция SELECT не завершится. У инструкции INSERT есть также специальный флаг DELAYED, при наличии которого инструкция помещается в очередь без блокирования клиентского приложения, что повышает его оперативность.

Если есть несколько записей, предназначенных для вставки в таблицу, воспользуйтесь многострочной инструкцией INSERT. Еще быстрее работает инструкция LOAD DATA INFILE. Для полной очистки таблицы лучше вызывать инструкцию TRUNCATE TABLE, а не DELETE. В этом случае программа MySQL удалит и снова создаст табличный файл, вместо того чтобы удалять записи одна за другой.

Если в состав инструкции входит сложное выражение, замените его пользовательской функцией. Естественно, это имеет смысл делать только тогда, когда предполагается многократно вызывать инструкцию. О создании собственных функций рассказывается в главе 31, "Расширение возможностей MySQL".

Обслуживание таблиц

Можно ускорить выполнение запросов, если хранить таблицы и индексы в упорядоченном виде. Инструкция OPTIMIZE TABLE улучшает таблицу тремя способами. Во-первых, она устраняет пустые промежутки, оставшиеся после удаления записей. Для таблиц MyISAM это означает возможность одновременного выполнения инструкций INSERT и SELECT. Во-вторых, она соединяет распределенные фрагменты таблиц с динамическими записями. И наконец, она сортирует индексы.

Инструкция ALTER TABLE позволяет отсортировать записи таблицы. Это тоже способствует ускорению некоторых запросов, хотя и не устраняет потребность в индексах.

Если таблица меняется редко, а дисковое пространство ограничено, имеет смысл сжать таблицу с помощью утилиты myisampack (см. главу 14, "Утилиты командной строки"). После этого таблица будет доступна только для чтения. Ее индексы необходимо перестроить, вызвав утилиту myisamchk. Данная методика позволяет уменьшить размер таблицы на 40-70%, в зависимости от формата ее содержимого. В листинге 26.9 показан процесс сжатия таблицы, содержащей названия штатов США.

```
# myisampack.exe state
Compressing state.MYD: (50 records)
- Calculating statistics
- Compressing file
32.42%
Remember to run myisamchk -rq on compressed tables
# myisamchk -rq state
- check key delete-chain
- check record delete-chain
- recovering (with sort) MyISAM-table 'state.MYI'
Data records: 50
- Fixing index 1
- Fixing index 2
```

Настройка конфигурации сервера

Когда речь заходит об объеме оперативной памяти сервера, совет всегда один: чем больше — тем лучше. Увеличение объема памяти способствует ускорению работы программы MySQL, так как в оперативной памяти она хранит свои временные таблицы и буферы записей. В подкаталоге support-files дистрибутива содержатся образцы конфигурационных файлов с различными вариантами настроек, касающихся использования памяти. Выберите тот вариант, который соответствует исходным параметрам сервера. Поработав с сервером какое-то время, можно будет оценить, какие из настроек требуют корректировки.

В листинге 26.10 показана конфигурация сервера, располагающего как минимум 1 Гбайт ОЗУ, четырьмя жесткими дисками и четырьмя центральными процессорами. Обратите внимание на важность индексного буфера. В данной конфигурации предполагается, что сервер хранит табличные данные на первом диске, а временные файлы — на втором. Таблицы InnoDB находятся на третьем диске, а журналы InnoDB — на четвертом.

```
[mysqld]
set-variable    = key_buffer=384M
set-variable    = max_allowed_packet=1M
set-variable    = table_cache=512
set-variable    = sort_buffer=2M
set-variable    = record_buffer=2M
set-variable    = thread_cache_size=8
set-variable    = thread_concurrency=8
set-variable    = myisam_sort_buffer_size=64M
log-bin
server-id       = 1
tmpdir          = /disk2/tmp/

# Таблицы BDB
set-variable = bdb_cache_size=384M
set-variable  =  bdb_max_lock=100000

# Таблицы InnoDB
innodb_data_home_dir = /disk3/
innodb_log_group_home_dir = /disk4/
innodb_log_arch_dir = /disk4/
innodb_data_file_path = ibdata1:250M;ibdata2:500M;ibdata3:1000M
set-variable  =  innodb_mirrored_log_groups=1
set-variable  =  innodb_log_files_in_group=3
set-variable = innodb_log_file_size=5M
set-variable  =  innodb_log_buffer_size=8M
innodb_flush_log_at_trx_commit=1
innodb_log_archive=0
set-variable  =  innodb_buffer_pool_size=16M
set-variable = innodb_additional_mem_pool_size=2M
set-variable  =  innodb_file_io_threads=4
set-variable  =  innodb_lock_wait_timeout=50
```

Когда сервер проработает какое-то время, выполните инструкцию SHOW STATUS, чтобы узнать его производительность. Сравните значения показателей Key_reads и Key_read_requests. Их соотношение будет очень низким, если программа MySQL часто пользуется индексным буфером. В случае необходимости попытайтесь повысить размер буфера.

Проследите изменение показателя Open_tables, сравнивая его созначением серверной переменной table_cache, которое можно узнать с помощью инструкции SHOW VARIABLES. Когда табличный буфер заполняется, программа MySQL вынуждена закрывать одни таблицы, чтобы открывать другие. Показатель Opened_tables отражает число таблиц, открывавшихся с момента запуска сервера. Сравните его с общим числом запросов (показатель Questions). Чем больше будет размер табличного буфера, тем реже придется открывать и закрывать таблицы.

Серверная переменная thread_cache_size задает размер кэша потоков. Как правило, на каждый процессордолжно приходиться два потока. Сравните показатели Threads_created и Connections, чтобы определить, как часто серверу приходилось повторно использовать потоки.

Просмотрите еще раз список переменных демона mysqld, приведенный в главе 14, "Утилиты командной строки". Есть много разных буферов и кэшей, увеличение размера которых способно повысить производительность сервера. После изменения конфигурации обязательно проведите повторные замеры.

Перекомпиляция MySQL

Команда разработчиков MySQL прилагает огромные усилия для оптимизации исполняемых файлов программы. Лучше всего пользоваться бинарными дистрибутивами, которые доступны на Web-узле MySQL. Вряд ли вам удастся получить более качественный исполняемый файл. Например, в дистрибутивы Linux зачастую включаются нестабильные версии компиляторов и библиотек. Разработчики MySQL всегда применяют самые стабильные версии в сочетании с оптимальными опциями компиляции.

Необходимость в компиляции возникает, когда для данной платформы невозможно найти скомпилированную версию программы, хотя эта ситуация маловероятна. Еще одна причина – желание поэкспериментировать с различными библиотеками. Но подобными экспериментами не стоит слишком увлекаться, так как в результате можно получить нестабильно работающий исполняемый файл.

На Web-узле MySQL приведена информация о том, как компилировать программу на различных платформах. Не поленитесь просмотреть рекомендации специалистов, поскольку здесь есть много "подводных камней", особенно в случае старых операционных систем.

Перед началом компиляции убедитесь в наличии утилит gzip и gnutar. Они необходимы для извлечения файлов из tar-архива. Учтите, что версия утилиты tar для Solaris содержит ошибку, которая не позволяет распаковывать некоторые архивы, поэтому желательно иметь GNU-версию утилиты.

Нужен также компилятор языка C++. Вполне подойдет какая-нибудь GNU-версия, включая egcs. Не забудьте и об утилите make.

Те, кто имеют опыт компиляции программ с открытыми кодами, должны быть знакомы со сценариями конфигурации, создаваемыми утилитой autoconf. Саму ее запускать не нужно. Файл Makefile создается сценарием configure. В листинге 26.11 показан вызов этого сценария с установками, которые рекомендованы разработчиками MySQL. Сценарий configure должен запускаться из каталога, содержащего исходные коды программы.

```
CFLAGS="-O3" \
CXX=gcc  \
CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql \
--enable-assembler \
--with-mysqld-ldflags=-all-static
```

В табл. 26.1 перечислены параметры сценария configure. Аналогичную информацию можно получить, вызвав сценарий с опцией --help. Если нужно включить поддержку таблиц Berkeley DB или InnoDB, не забудьте указать соответствующие оп-

ции. В исходные дистрибутивы MySQL входят все необходимые для этого файлы, поэтому путь к библиотекам Berkeley DB задается только в том случае, когда требуется использовать их альтернативные версии.

Общая конфигурация	
`--cache-file=`*файл*	**Кэшировать** результаты **в** *файле*
`--help`	Вывести список опций
`--no-create`	Не создавать выходные файлы
`--quiet, --silent`	Не отображать сообщения `"checking..."`
`--version`	Отобразить номер версии утилиты `autoconf`, которая создала данный сценарий
Имена файлов и каталогов	
`--prefix=`*префикс*	Помещать архитектурно-независимые файлы в каталог *префикс* (по умолчанию — `/usr/local`)
`--exec-prefix=`*префикс1*	Помещать архитектурно-зависимые файлы в каталог *префикс1* (по умолчанию — *префикс*)
`--bindir=`*каталог*	Помещать пользовательские исполняемые файлы в *каталог* (по умолчанию — *префикс1*`/bin`)
`--sbindir=`*каталог*	Помещать административные исполняемые файлы в *каталог* (по умолчанию — *префикс1*`/sbin`)
`--libexecdir=`*каталог*	Помещать исполняемые файлы программ в *каталог* (по умолчанию — *префикс1*`/libexec`)
`--datadir=`*каталог*	Помещать архитектурно-независимые данные, доступные только для чтения, в *каталог* (по умолчанию — *префикс*`/share`)
`--sysconfdir=`*каталог*	Помещать системные данные, доступные только для чтения, в *каталог* (по умолчанию — *префикс*`/etc`)

Имена файлов и каталогов

`--sharedstatedir=`*`каталог`*
Помещать модифицируемые архитектурно-независимые данные в *каталог* (по умолчанию — *префикс*`/com`)

`--localstatedir=`*`каталог`*
Помещать модифицируемые системные данные в *каталог* (по умолчанию — *префикс*`/var`)

`--libdir=`*`каталог`*
Помещать объектные библиотеки в *каталог* (по умолчанию — *префикс*`/lib`)

`--includedir=`*`каталог`*
Помещать файлы заголовков в *каталог* (по умолчанию — *префикс*`/include`)

`--oldincludedir=`*`каталог`*
Помещать файлы заголовков, не предназначенные для компиляторов `gcc`, в *каталог* (по умолчанию — `/usr/include`)

`--infodir=`*`каталог`*
Помещать документацию системы Info в *каталог* (по умолчанию — *префикс*`/info`)

`--mandir=`*`каталог`*
Помещать документацию системы Man в *каталог* (по умолчанию — *префикс*`/man`)

`--srcdir=`*`каталог`*
Искать исходные файлы в *каталоге* (сначала — в каталоге сценария)

`--program-prefix=`*`префикс`*
Добавлять *префикс* к именам инсталлированных программ

`--program-suffix=`*`суффикс`*
Добавлять *суффикс* к именам инсталлированных программ

`program-transform-name=`*`программа`*
Запускать *программу для* преобразования имен инсталлированных программ

Тип узла

`--build=`*`узел_построения`*
Рабочий узел (по умолчанию - guessed)

`--host=`*`узел`*

`--target=`*`целевой_узел`*
Целевой узел (по умолчанию — *узел*)

Свойства и пакеты

`--disable-`*`свойство`*
Не включать *свойство* (то же, что `--enable-`*`свойство`*`=no`)

Свойства и пакеты	
`--enable-`*`свойство`*`[=`*`аргумент`*`]`	Включить *свойство* (по умолчанию — yes)
`--with-`*`пакет`*`[=`*`аргумент`*`]`	Использовать *пакет* (по умолчанию — yes)
`--without-`*`пакет`*	Не использовать *пакет* (то же, что `--enable-`*`пакет`*`=no`)
`--x-includes=`*`каталог`*	Искать включаемые файлы X Window в *каталоге*
`--x-libraries=`*`каталог`*	Искать библиотеки X Window в *каталоге*
Дополнительные возможности	
`--enable-maintainer-mode`	Разрешить правила и зависимости утилиты make, лишние (а ин огда непонятные) для обычного инсталлятора
`--enable-shared[=`*`аргумент`*`]`	Создавать совместно используемые библиотеки
`--enable-static[=`*`аргумент`*`]`	Создавать статические библиотеки
`--enable-fast-install[=`*`аргумент`*`]`	Оптимизировать установки для быстрой инсталляции
`--with-gnu-ld`	Предполагать, что компилятор языка C использует GNU-утилиту `ld` (по умолчанию отключена)
`--disable-libtool-lock`	Избегать блокировок (могут помешать параллельной работе потоков компилятора)
`--with-other-libc=`*`каталог`*	Подключать библиотеку `libc` и другие стандартные библиотеки, расположенные в нестандартном каталоге. Изначально эта опция появилась, чтобы компилятор мог подключать библиотеку `glibc` версии 2.2, не требуя от пользователей обновлять стандартную библиотеку `libc`
`--with-server-suffix`	Добавлять идентификатор сервера в строку версии

Дополнительные возможности

`--with-mit-threads`	Использовать библиотеку MIT-pthreads
`--with-pthread`	Использовать библиотеку `pthread`
`--with-named-thread-libs=`*`аргумент`*	Использовать указанную потоковую библиотеку, а не ту, которая автоматически обнаруживается сценарием
`--with-named-curses-libs=`*`аргумент`*	Использовать указанную библиотеку `curses`, а не ту,которая автоматически обнаруживается сценарием
`--with-named-z-libs=`*`аргумент`*	Использовать указанную библиотеку `zlib`, а не ту, которая автоматически обнаруживается сценарием
`--enable-thread-safe-client`	Компилировать клиент с поддержкой потоков
`--enable-assembler`	Использовать ассемблерные версии некоторых строковых функций, если это возможно
`--with-raid`	Включить поддержку RAID-дисков
`--with-unix-socket-path=`*`сокет`*	Использовать указанный UNIX-сокет; параметр *сокет* должен представлять собой абсолютное путевое имя
`--with-tcp-port=`*`порт`*	Использовать заданный *порт* для сервисов MySQL (по умолчанию — 3306)
`--with-mysqld-user=`*`пользователь`*	Запускать демон `mysqld` от имени *пользователя*
`--disable-large-files`	Отключить поддержку больших файлов
`--with-libwrap[=`*`каталог`*`]`	Компилировать программу с поддержкой библиотеки `libwrap` (TCP-оболочки)
`--without-debug`	Не включать в программу код отладки
`--with-mysqld-ldflags=`*`аргумент`*	Использовать дополнительные параметры компоновки демона `mysqld`
`--with-client-ldflags=`*`аргумент`*	Использовать дополнительные параметры компоновки клиентов

Дополнительные возможности

`--with-low-memory`	Стараться использовать меньше памяти на этапе компиляции
`--with-comment`	Выдавать комментарии о среде компиляции
`--without-server`	Создавать только клиентскую программу
`--without-docs`	Не создавать документацию
`--without-bench`	Не создавать набор тестов производительности
`--without-readline`	Использовать системную версию библиотеки `readline`, а не ту, что входит в дистрибутив
`--with-charset=`*`набор_символов`*	Сделать указанный набор символов основным (по умолчанию — `latin1`)
`--with-extra-charsets=`*`список`*	Использовать дополнительные наборы символов помимо основного; аргумент *список* может содержать перечень наборов символов, ключевые слова complex (включать все наборы символов, которые нельзя загрузить динамически) или all (включать все наборы символов)
`--with-berkeley-db[=`*`каталог`*`]`	Использовать библиотеку Berkeley DB, расположенную в указанном *каталоге*
`--with-berkeley-db-includes=`*`каталог`*	Искать файлы заголовков Berkeley DB в *каталоге*
`--with-berkeley-db-libs=`*`каталог`*	Искать библиотечные файлы Berkeley DB в *каталоге*
`--with-innodb`	Использовать библиотеку InnoDB
`--with-gemini[=`*`каталог`*`]`	Использовать библиотеку Gemini, расположенную в указанном *каталоге*

Когда сценарий configure окончит свою работу, запустите утилиту make, чтобы создать исполняемый файл. После этого необходимо выполнить команду make `install`, которая поместит созданные файлы в соответствующие каталоги. Подробнее об этом рассказывалось в главе 2, "Инсталляция MySQL".

БЕЗОПАСНОСТЬ

В этой главе.

Схема привилегий
Задание привилегий
Обеспечение безопасности

Глава 27

В этой главе рассказывается о том, как программа MySQL управляет привилегиями и как можно обеспечить безопасность сервера баз данных. В MySQL применяются списки управления доступом, традиционные для многих систем. На сервере хранятся таблицы пользовательских привилегий. Привилегии могут относиться к столбцам, таблицам, базам данных или быть глобальными.

Программа MySQL выполняется в виде системного сервиса или демона, поэтому нужно учитывать средства безопасности, предоставляемые операционной системой или сетевым программным обеспечением. Повысить безопасность сервера можно, ограничив к нему доступ посредством сетевого фильтра (брандмауэра). Если в базе данных хранится особо важная информация, рассмотрите возможность найма профессионального консультанта по вопросам безопасности.

Схема привилегий

В списке управления доступом (Access Control List, ACL) указывается, какие инструкции разрешено выполнять тем или иным пользователям. Такой список можно связать с пользователем, узлом, базой данных, таблицей или столбцом. Программа MySQL сверяет каждое обращение к базе данных с имеющимися списками управления доступом и определяет, есть ли у пользователя право выполнить запрашиваемое действие.

Пользователи идентифицируют себя по имени, паролю и адресу узла. Пользовательские имена и пароли MySQL не связаны напрямую с именами и паролями операционной системы. Просто большинство клиентов MySQL по умолчанию берет имя пользователя, которое было указано при регистрации в системе. Это довольно удобно, хотя и не является обязательным правилом.

Пользовательские имена и пароли могут быть длиной до 16 символов. Пароль разрешается оставлять пустым. Это самый низкий уровень безопасности. Он допустим только в том случае, когда доступ ограничивается по каким-то другим критериям, например по адресу узла.

Перечень привилегий хранится в базе данных mysql. Сценарий mysql_install_db, который запускается в ходе инсталляции программы, создает в этой базе данных пять таблиц с описаниями привилегий (табл. 27.1).

Таблица	*Содержимое*
columns_priv	Привилегии отдельных столбцов
db	Привилегии всей базы данных
host	Привилегии всех пользователей того или иного узла
tables_priv	Привилегии отдельных таблиц
user	Глобальные привилегии

В таблице user описываются глобальные права доступа и хранятся пользовательские пароли. Алгоритм шифрования паролей отличается от того, который применяется операционной системой. Пароль можно создать с помощью функции PASSWORD() (см. главу 12, "Встроенные функции") или путем прямого изменения таблицы user, но удобнее всего это делать с помощью инструкции GRANT.

При попытке подключения к серверу программа MySQL обращается к таблице user и проверяет, имеет ли пользователь право на подключение. Имя, пароль и адрес узла пользователя должны соответствовать как минимум одной записи таблицы. Если этого не происходит, программа отказывает пользователю в запросе. Но даже когда подключение легитимно, пользователю может быть разрешено выполнять лишь ограниченный набор SQL-инструкций. Права доступа к данным контролируются остальными четырьмя таблицами базы mysql. Следует, правда, отметить, что любому зарегистрированному пользователю разрешено вводить инструкции SELECT с литералами в списке возвращаемых столбцов, например SELECT NOW().

В листинге 27.1 показаны определения всех пяти таблиц привилегий.

```
CREATE TABLE columns_priv (
   Host CHAR(60) BINARY NOT NULL DEFAULT '',
   Db CHAR(64) BINARY NOT NULL DEFAULT '',
   User CHAR(16) BINARY NOT NULL DEFAULT '',
   Table_name CHAR(64) BINARY NOT NULL DEFAULT '',
   Column_name CHAR(64) BINARY NOT NULL DEFAULT '',
   TIMESTAMP TIMESTAMP(14) NOT NULL,
   Column_priv SET('Select', 'Insert', 'Update', 'References')
      NOT NULL DEFAULT '',
   PRIMARY KEY (Host, Db, User, Table_name, Column_name)
)
TYPE=MyISAM
COMMENT='Column privileges';

CREATE TABLE db (
   Host CHAR(60) BINARY NOT NULL DEFAULT '',
   Db CHAR(64) BINARY NOT NULL DEFAULT '',
```

```
    User CHAR(16) BINARY NOT NULL DEFAULT '',
    Select_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Insert_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Update_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Delete_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Create_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Drop_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Grant_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    References_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Index_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Alter_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    PRIMARY KEY (Host, Db, User),
    KEY User (User)
)
TYPE=MyISAM
COMMENT='Database privileges';

CREATE TABLE host (
    Host CHAR(60) BINARY NOT NULL DEFAULT '',
    Db CHAR(64) BINARY NOT NULL DEFAULT '',
    Select_priv ENUM('N 'Y') NOT NULL DEFAULT 'N',
    Insert_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Update_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Delete_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Create_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Drop_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Grant_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    References_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Index_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Alter_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    PRIMARY KEY (Host, Db)
)
TYPE=MyISAM
COMMENT='Host privileges; Merged with database privileges';

CREATE TABLE tables_priv (
    Host CHAR(60) BINARY NOT NULL DEFAULT '',
    Db CHAR(64) BINARY NOT NULL DEFAULT '',
    User CHAR(16) BINARY NOT NULL DEFAULT '',
    Table_name CHAR(60) BINARY NOT NULL DEFAULT '',
    Grantor CHAR(77) NOT NULL DEFAULT '',
    TIMESTAMP TIMESTAMP(14) NOT NULL,
    Table_priv set('Select', 'Insert', 'Update', 'Delete',
        'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter')
        NOT NULL DEFAULT '',
    Column_priv set('Select', 'Insert', 'Update', 'References')
        NOT NULL DEFAULT '',
    PRIMARY KEY (Host, Db, User, Table_name),
    KEY Grantor (Grantor)
)
TYPE=MyISAM
COMMENT='Table privileges';

CREATE TABLE user (
    Host CHAR(60) BINARY NOT NULL DEFAULT '',
    User CHAR(16) BINARY NOT NULL DEFAULT '',
```

```
    Password CHAR(16) BINARY NOT NULL DEFAULT  '',
    Select_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Insert_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Update_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Delete_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Create_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Drop_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Reload_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Shutdown_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Process_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    File_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Grant_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    References_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Index_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Alter_priv ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    PRIMARY KEY (Host, User)
)
TYPE=MyISAM
COMMENT='Users and global privileges';
```

За каждым пользователем каждого узла в таблице `user` закреплен свой набор привилегий, заданных в виде перечислений со значениями `'Y'/'N'`. В столбце Host содержится IP-адрес либо доменное имя компьютера. Разрешается использовать метасимволы % и _ для задания диапазона адресов. Например, выражение `%.leonatkinson.com` соответст-

именем пользователя leon. Поскольку подобный выбор не всегда очевиден, лучше избегать пустых имен пользователей.

Таблицы db и host совместноконтролируют доступ к базам данных. Подобно таблице user, в таблице db не допускаются имена пользователей с метасимволами. При выполнении операций сравнения программа MySQL сортирует записи этой таблицы в таком порядке: имя узла, имя базы данных, имя пользователя. Другими словами, сначала выбирается наиболее точное имя узла, затем — наиболее точное имя базы данных, а затем — наиболее точное имя пользователя. Записи таблицы host сортируются сначала по имени узла, потом — по имени базы данных.

Таблицы columns_priv и tables_priv допускают наличие метасимволов в столбце Host, но не вдругих столбцах. Поэтому, чтобы предоставить пользователю права доступа к трем столбцам таблицы, нужно задать три записи в таблице columns_priv.

В табл. 27.2 перечислены столбцы привилегий, встречающиеся в таблицах. Смысл каждого из них определяется таблицей, в которой он присутствует. Запись таблицы db применима ко всем таблицам базы данных. Привилегии, заданные в таблице user, являются глобальными. Есть привилегии, которые определены только на глобальном уровне. Например, право перезагрузки позволяет пользователю очищать кэш-буферы, совместно используемые всеми соединениями и базами данных.

Столбец	*Описание*
Alter_priv	Пользователь может вводить инструкцию ALTER TABLE
Create_priv	Пользователь может создавать таблицы и базы данных
Delete_priv	Пользователь может вводить инс трукцию DELETE
Drop_priv	Пользователь может удалять таблицы и базы данных
File_priv	Пользователь может обращаться к файлам локальной файловой системы
Grant_priv	Пользователь может передавать свои привилегии другим пользователям
Index_priv	Пользователь может добавлять или удалять индексы
Insert_priv	Пользователь может вводить инструкцию INSERT
Process_priv	Пользователь может просматривать список соединений и удалять соединения
Reload_priv	Пользователь может очищать буферы
References_priv	Соответствующая привилегия еще не реализована
Select_priv	Пользователь может создавать запросы к таблицам (инструкция SELECT)
Shutdown_priv	Пользователь может останавливать сервер
Update_priv	Пользователь может вводить инструкцию UPDATE

Названия привилегий в большинстве своем соответствуют названиям инструкций SQL, поэтому легко понять, какие права получает пользователь благодаря той или иной привилегии. Привилегия Alter позволяет пользователю менять определения таблиц. Привилегия Create дает возможность, в зависимости от контекста, создавать таблицы или базы данных. Привилегия Delete разрешает удалять записи таблиц, а привилегия Drop — сами таблицы или базы данных. Привилегия Index позволяет создавать и удалять индексы, привилегия Insert — вставлять записи в таблицы, а привилегии Select и Update — выполнять инструкции SELECT и UPDATE соответственно.

Привилегия File разрешает пользователям выполнять инструкцию LOAD DATA INFILE, а также инструкцию SELECT с предложением INTO OUTFILE. Она определяет лишь возможность чтения и записи файлов на сервере. Естественно, помимо этого пользователь должен иметь право чтения и записи самой таблицы. Файловые операции осуществляются от имени пользователя, запустившего демон mysqld. Операционная система может налагать на него свои ограничения, чтобы демон не повредил важные системные файлы.

Привилегия Grant позволяет пользователю вызывать инструкцию GRANT для передачи своих привилегий другим пользователям. В инструкции GRANT разрешается указывать только те привилегии, которыми владеет текущий пользователь. Например, если пользователь имеет только право создавать таблицы, то он не сможет выполнить инструкцию GRANT RELOAD.

Привилегия Process дает пользователю возможность просматривать список соединений с помощью инструкции SHOW PROCESSLIST и удалять любое соединение при помощи инструкции KILL. Подобной привилегией владеют только администраторы, имеющие доступ к утилите mysqladmin.

Привилегия Reload позволяет пользователю вводить любую из инструкций FLUSH, описанных в главе 13, "Инструкции SQL". Обычно кэш-буферы не требуют очистки, за исключением ряда случаев. При непосредственном редактировании табличных файлов необходимо очистить табличный буфер. Если таблицы привилегий меняются в обход инструкций GRANT и REVOKE, то нужно очистить буфер привилегий. Рядовым пользователям данная привилегия не нужна.

Привилегия Shutdown разрешает пользователю завершать работу демона mysqld. Привилегия на запуск сервера не существует, ведь если нет серверного процесса, то и некому проверять привилегию! Для перезапуска сервера необходим пользователь, обладающий соответствующими правами на уровне операционной системы. Например, в Linux пользователь root может вызвать сценарий safe_mysqld.

Программа MySQL проверяет привилегии для каждой инструкции, вводимой пользователем. Процесс проверки таблиц проиллюстрирован на рис. 27.1. Как только программа находит нужную привилегию, она прекращает дальнейший поиск. В первую очередь просматривается таблица user. Обычно у рядовых пользователей нет глобальных привилегий, поэтому программа переходит к таблицам db и host. Каждая запись таблицы db описывает права доступа к заданной базе данных. Если поле Host является пустым, соответствующая запись должна присутствовать в таблице host. В данном случае программа выбирает только те привилегии, которые определены в обеих таблицах. Например, если в таблице db пользователю выдается привилегия Update, а в таблице host - нет, программа не разрешит пользователю обновлять все таблицы базы данных.

Если искомые привилегии в таблицах db и host отсутствуют, программа проверяет таблицу table s_priv. В ней определены привилегии доступа к отдельным таблицам. В последнюю очередь проверяется таблица column s_priv. Если и там ничего не найдено, пользовательский запрос отклоняется.

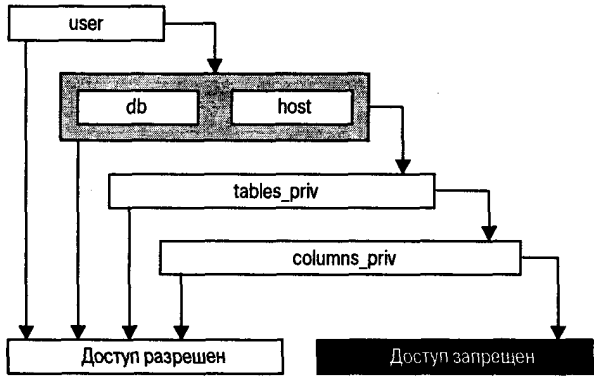


Рис. 27.1. Проверка привилегий

Задание привилегий

Для задания привилегий предназначены инструкции GRANT и REVOKE (см. главу 13, "Инструкции SQL"). Можно также непосредственно манипулировать таблицами привилегий. После внесения изменений следует выполнить инструкцию FLUSH PRIVILEGES . Пароли для таблицы user создаются с помощью функции PASSWORD() .

В дистрибутив MySQL входит сценарий проверки привилегий mysqlaccess , написанный **Ивом Карлье** (Yves **Carlier**). Его синтаксис был описан в главе 14, "Утилиты командной строки". Данный сценарий поможет вам понять, какими возможностями обладает тот или иной пользователь. Он также выдаст предупреждение в случае обнаружения опасной конфигурации, например когда любой пользователь может зарегистрироваться под именем root без ввода пароля. В листинге 27.2 показаны типичные результаты работы этого сценария.

```
Access-rights
for USER 'leon', from HOST 'localhost', to DB 'freetrade'
        +-----------------+---+ +-----------------+---+
        | Select_priv     | N | | Shutdown_priv   | N |
        | Insert_priv     | N | | Process_priv    | N |
        | Update_priv     | N | | File_priv       | N |
        | Delete_priv     | N | | Grant_priv      | N |
        | Create_priv     | N | | References_priv | N |
        | Drop_priv       | N | | Index_priv      | N |
        | Reload_priv     | N | | Alter_priv      | N |
        +-----------------+---+ +-----------------+---+
NOTE:    A password is required for user 'leon' :-|
```

```
The following rules are used:
 db    :  'No matching rule'
 host  :  'Not processed: host-field is not empty in db-table.'
 user  :  'localhost','leon','2b62070b42bbaffa','N','N','N','N',
          'N','N','N','N','N','N','N','N','N','N'

Access-rights
for USER 'leon', from HOST 'localhost', to DB 'test'
        +-----------------+---+ +-----------------+---+
        | Select_priv     | Y | | Shutdown_priv   | N |
        | Insert_priv     | Y | | Process_priv    | N |
        | Update_priv     | Y | | File_priv       | N |
        | Delete_priv     | Y | | Grant_priv      | N |
        | Create_priv     | Y | | References_priv | Y |
        | Drop_priv       | Y | | Index_priv      | Y |
        | Reload_priv     | N | | Alter_priv      | Y |
        +-----------------+---+ +-----------------+---+
NOTE:     A password is required for user 'leon' :-(
BEWARE:   Any user with the appropriate permissions has access to
          your DB!
       :   Check your users!

The following rules are used:
db    :  '%','test','','Y','Y','Y','Y','Y','Y','N','Y','Y','Y'
host  :  'Not processed: host-field is not empty in db-table.'
user  :  'localhost','leon','2b62070b42bbaffa','N','N','N','N',
         'N','N','N','N','N','N','N','N','N','N'


Access-rights
for USER 'leon', from HOST 'localhost', to DB 'ANYNEW DB'
        +-----------------+---+ +-----------------+---+
        | Select_priv     | N | | Shutdown_priv   | N |
        | Insert_priv     | N | | Process_priv    | N |
        | Update_priv     | N | | File_priv       | N |
        | Delete_priv     | N | | Grant_priv      | N |
        | Create_priv     | N | | References_priv | N |
        | Drop_priv       | N | | Index_priv      | N |
        | Reload_priv     | N | | Alter_priv      | N |
        +-----------------+---+ +-----------------+---+
NOTE:     A password is required for user 'leon' :-(

The following rules are used:
db    :  'No matching rule'
host  :  'Not processed: host-field is not empty in db-table.'
user  :  'localhost','leon','2b62070b42bbaffa','N','N','N','N',
         'N','N','N','N','N','N','N','N','N','N'
```

Проверить существующие привилегии позволяет также инструкция SHOW GRANTS. Она возвращает результаты в виде инструкций `GRANT`, которые были выполнены при задании привилегий указанного пользователя (листинг 27.3). Такие результаты проще понять, чем таблицы с обозначениями `'Y'` и `'N'`.

```
mysql> SHOW GRANTS FOR leon@localhost;
+-----------------------------------------------------+
| Grants for leon@localhost                           |
+-----------------------------------------------------+
| GRANT SELECT ON mysql.* TO 'leon'@'localhost'       |
| GRANT SELECT ON coremysql.* TO 'leon'@'localhost'   |
| GRANT ALL PRIVILEGES ON leon.* TO 'leon'@'localhost' |
+-----------------------------------------------------+
3 rows in set (0.00 sec)
```

Лучший принцип управления привилегиями — свести их к минимуму. Позволяйте пользователям выполнять только те действия, которые от них ожидаются. Например, если есть группа Web-приложений, обращающихся к серверу MySQL, создайте для каждого приложения отдельную пользовательскую учетную запись и предоставьте этому пользователю право манипулировать таблицами только одной базы данных. Не разрешайте пользователю менять определения таблиц или создавать индексы, если только не предполагается выполнять эти действия регулярно. Таким образом, ошибка в приложении, открывающая пользователям возможность вводить произвольные запросы, не даст им разрушить другие базы данных.

Большинству пользователей не нужны глобальные привилегии. Наличие глобальной привилегии Select позволит им просматривать содержимое всех баз данных, в том числе `mysql`, где хранятся описания привилегий и пароли. Привилегия Process дает пользователю возможность следить за чужими запросами, включая те, в которых задаются пароли. Короче говоря, глобальные привилегии предназначены для администраторов баз данных.

Ограничьте количество узлов, от которых могут поступать запросы. Не разрешайте пользователям регистрироваться на сервере с произвольного компьютера. Даже если пользователи подключаются к системе посредством коммутируемых каналов связи, разрешите соединения только от одного провайдера. Иногда имеет смысл допускать подключения только с локального узла. В этом случае пользователи вынуждены будут сначала регистрироваться на компьютере, где установлена база данных, т.е. проходить двойную проверку: при входе в операционную систему и при входе в программу MySQL.

Доступ к базе привилегий нужен только администратору. Не позволяйте другим пользователям работать с этой базой данных. Если они смогут прочесть содержимое таблицы `user`, то смогут зарегистрироваться на сервере от имени любого пользователя, так как увидятвсе пароли. Пароли хранятся в зашифрованном виде, но клиенты MySQL передают серверу именно зашифрованные пароли.

Обеспечение безопасности

Чтобы обеспечить безопасность сервера баз данных, необходимо тщательно спланировать работу системы. Старайтесь использовать пароли, которые невозможно ни угадать, ни подсмотреть. Избегайте хранения паролей в незашифрованном виде как в самой базе данных, так и записанными **где-то** на бумажке. Если такая бумажка при-

клеена к монитору, что нередко происходит, то любой сможет в отсутствие хозяина пароля зарегистрироваться в системе под его именем. Многократное использование паролей тоже нежелательно, так как от взлома одной системы не должныстрадать другие системы, даже если пользователям лень запоминать "лишние" пароли. Меры предосторожности лишними не бывают!

Выработайте политику выбора надлежащих паролей. Пароли, представляющие собой обычные слова, элементарно взламываются специальными программами методом последовательного перебора. Самыми эффективными являются случайные комбинации букв, чисел и знаков пунктуации, но такие пароли трудно запоминать. Существует популярная методика выбора паролей. Ее суть заключается в том, что пароль составляется из первых букв какой-нибудь легко запоминающейся фразы. Например, пароль `tlotr-J.T. (1955)` соответствует фразе "The Lord ofthe Rings – John Tolkien (1955)". Такой пароль содержит смесь строчных и прописных букв, чисел и знаков пунктуации и имеет максимальную длину — 16 символов.

Никогда не задавайте пароль в командной строке. Пусть клиентская программа сама попросит ввести его. Команды, вводимые в интерпретаторе, видны любому пользователю, просматривающему список процессов. Функции клиентской библиотеки MySQL стараются максимально быстро очищать командную строку, но все равно в течение короткого промежутка времени она остается общедоступной. Пароли можно также хранить в личном конфигурационном файле (см. главу 14, "Утилиты командной строки").

Бывают ситуации, когда пароли приходится хранить в исходном виде. Один из примеров — Web-приложение. Программа должна регистрироваться на сервере, поэтому она хранит строку пароля, передаваемую клиентской библиотеке. Утилита `strings`, имеющаяся в большинстве версий UNIX, позволяет легко просматривать текстовые строки, включенные в исполняемый файл. Ограничьте доступ к таким файлам. Убедитесь, что файл, в котором содержится пароль, не находится в каталогах, доступных Web-серверу. Если Web-приложение выполняется от имени конкретного пользователя, поместите пароль в конфигурационный файл.

Для особо недоверчивых существует такое средство, как опция `--secure` демона `mysqld`. При ее наличии демон выполняет обратное преобразование доменных имен, проверяя совпадение IP-адресов. Например, если IP-адресу 192.168.100.128 соответствует имя `safe.your.net`, но поиск этого имени в DNS оканчивается неудачей или оказывается, что за этим именем закреплен другой адрес, то программа MySQL отказывается устанавливать соединение. Можно также воспользоваться опцией `--skip-name-resolve`, которая вообще отключит процедуру поиска доменных имен. В этом случае в столбцах Host таблиц привилегий будут разрешены только IP-адреса.

Если в системе поддерживаются UNIX-сокеты и работа ведется с базой данных, расположенной на локальномузле, задайте опцию `--skip-networking`. Она отключит поддержку Internet-соединений. Соединения можно будет устанавливать только через сокет.

Не стоит слепо доверять данным, поступающим из внешнего мира. Пользователи могут передавать цепочки байтов, нарушающие работоспособность системы или приводящие к неожиданным результатам. Представим себе приложение, заносящее в базу данных запись с помощью инструкции INSERT, аргумент для которой предоставляется пользователем. Если в состав аргумента входит одинарная кавычка, она послужит сигналом к завершению строкового литерала. Это открывает возможность для видоизменения инструкции. Чтобы этого избежать, защищайте специальные символы от

интерпретации, например с помощью PHP-функции `addslashes()`, или пользуйтесь параметризованными запросами, как в MySQL++. Нельзя защитить от интерпретации специальные символы числовых литералов, но можно удалить их в приложении, выполнив операцию приведения типов, или взять в одинарные кавычки, чтобы позволить программе MySQL самой осуществить нужное преобразование. Если числовой литерал заключен в одинарные кавычки, программа преобразует строку в число в случае необходимости.

Безопасность можно обеспечивать на физическом уровне. Очень часто под сервер баз данных выделяют отдельный компьютер. В этом случае можно будет запретить все подключения к серверу, кроме тех, которые устанавливаются через порт 3306, контролируемый программой MySQL.

Еще более радикальное средство — создание двух сетей. В такой схеме общедоступный компьютер принимает запросы от внешних пользователей через один сетевой интерфейс, а другой интерфейс служит для подключения к локальной сети, в которой работает сервер баз данных. В результате сервер освобождается от обслуживания внешнего трафика. Рассмотрим рис. 27.2. Пользователи запрашивают HTML-страницы у Web-сервера, выполняющегося на общедоступном компьютере. В ответ на это Web-приложение, написанное на PHP, подключается к серверу баз данных, используя IP-адрес из частного диапазона, начинающийся на 10. Чтобы дополнительно защитить Web-сервер, можно поставить на пути во внешний мирбрандмауэр.

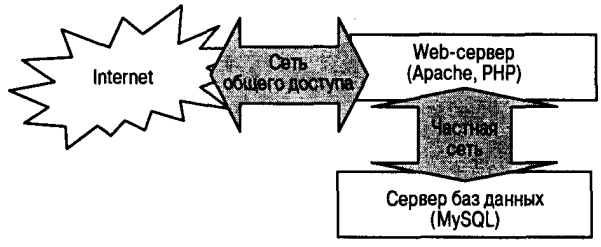


Рис. 27,2. Вынесение сервера в отдельную сеть

Программа MySQL поддерживает обмен сжатыми и зашифрованными данными между клиентом и сервером. Сжатие позволяет уменьшить трафик и представить данные в более защищенном виде. Благодаря шифрованию генерируются данные, которые сможет прочитать только авторизованный получатель. Для шифрования применяется протокол SSL (Secure Sockets Layer — протокол защищенных сокетов).

Как объяснялось в главе 2, "Инсталляция MySQL", сервер MySQL должен выполняться от имени пользователя, учетная запись которого создана специально для этих целей. Нельзя предоставлять серверу права суперпользователя, чтобы избежать ситуации, когда из-за ошибки разработчиков MySQL злоумышленники получают возможность запускать программы на сервере.

В дистрибутив MySQL входит утилита `mysqld_multi`, позволяющая запускать несколько серверов на одном компьютере. Это обеспечивает дополнительную безопасность системы, в которой работает много пользователей. Каждый пользователь может управлять собственной базой данных, что избавляет администратора от необходимости тщательно продумывать схему передачи привилегий. Подробнее об организации многосерверной среды рассказывается в главе 29, "Распределенные базы данных".

ПЕРЕНОС ДАННЫХ В РАЗНЫЕ СУБД

В этой главе…

Переключение между СУБД
Устранение несовместимостей
Использование режима ANSI
Уникальные свойства MySQL

Глава 28

В этой главе рассказывается о том, чем MySQL отличается от других СУБД и в чем особенности имеющейся в данной СУБД реализации языка SQL. Изложенная информация должна помочь читателям в создании таких баз данных, которые легко переносить между разными серверами. Приводятся также сведения о переносе существующих баз данных в MySQL.

Программа MySQL реализует начальный уровень стандарта SQL-92, известный как SQL2. Это самый новый из официальных стандартов, хотя ожидалось, что в 2001 г. будет принят стандарт SQL3. В MySQL реализован также протокол ODBC 2. Близкое соответствие этим стандартам упрощает написание инструкций и программ, которые одинаково интерпретируются разными СУБД.

Переключение между СУБД

Многие факторы заставляют пользователей менять СУБД. К этому могут подтолкнуть технические причины, например потребность в функциональной возможности, которой нет у текущей СУБД. Иногда просто выбирают более современный и производительный сервер.

Компании, как правило, предпочитают работать с официально признанными продуктами. Их службы технической поддержки имеют опыт обслуживания таких систем, поэтому пользователи вынуждены подчиняться политике компании. Нередки случаи, когда компания работает с программными средствами единственного производителя, например Microsoft. Есть и компании, которые могут себе позволить только ПО с открытыми кодами.

Выбор СУБД зачастую диктуется сугубо финансовыми соображениями. Например, компания может предпочесть конкретную СУБД, потому что у нее заключен договор об обмене услугами с производителем этой СУБД. Маркетологи компании могут прийти к выводу, что ее продукт будет продаваться лучше, если он будет работать с популярной СУБД. Широкий интерес к программам с открытыми кодами возник в 1998 г., когда компания Netscape объявила о выпуске своего Web-броузера на условиях

открытой лицензии. С тех пор многие разработчики ПО стали поддерживать линию открытых продуктов, так как это выгодно с точки зрения распространения программ.

Аргумент, который люди часто приводят против той или иной программы, заключается в отсутствии технической поддержки. Мне доводилось слышать объяснения, что, мол, для программ с открытыми кодами невозможно найти надежные каналы технической поддержки в случае возникновения проблем. На самом деле у любой программы с относительно широким кругом приверженцев есть сопутствующие интерактивные форумы, где пользователи бесплатно делятся информацией друг с другом. Это касается как открытых программ, так и коммерческого ПО. Тем не менее многие по-прежнему считают, что надежной может считаться только оплаченная техническая поддержка. MySQL, как и многие другие СУБД, располагает службой технической поддержки, предоставляющей профессиональные услуги за деньги. Наличие такой службы является одним из основных преимуществ MySQL, поскольку этим занимаются сами создатели программы — компания MySQL AB. Как следствие, качество оказываемых услуг оказывается выше, чем у более крупных компаний, которые не могут себе позволить сосредоточить все усилия только на базах данных.

Наконец, наиболее эффективный аргумент в пользу той или иной СУБД — профессиональный опыт разработчиков. Если команда разработчиков раньше имела дело, скажем, с серверами Sybase, то им проще будет работать в знакомой среде. Различия между СУБД на уровне SQL понять несложно, но средства обслуживания серверов разнятся очень сильно. Впрочем, учитывая время, затрачиваемое на освоение нового сервера, любая команда разработчиков сможет справиться с этой задачей.

Устранение несовместимостей

Программа MySQL не поддерживает некоторые функциональные возможности, присутствующие у других реляционных СУБД. Ниже рассказывается о том, что нужно учесть при переносе в MySQL приложений, написанных для другой системы.

В MySQL таблицы реализованы в виде файлов, располагающихся в каталогах, имена которых соответствуют именам баз данных. Есть СУБД, в которых табличные данные хранятся в больших файлах. Иногда каждой базе данных соответствует всего один файл. Системный каталог базы данных (т.е. набор ее схем) может называться system или master. Для доступа к системному каталогу MySQL предназначены инструкции семейства SHOW. Например, инструкция SHOW TABLES выводит список таблиц базы данных.

В большинстве СУБД при ссылке на таблицы применяется точечная нотация. Скажем, в Microsoft SQL Server полное имя столбца может выглядеть так: `server_c.store_db.store_admin.invoice.id`. В MySQL полные имена столбцов записываются в формате *`база_данных.таблица.столбец`*.

В некоторых СУБД временные таблицы обрабатываются только приложениями. Приложение должно само удалить таблицу, когда в ней больше нет необходимости. В других СУБД реализована система автоматической очистки временных таблиц. В MS SQL Server временные таблицы помечаются символом `#`. В MySQL этому соответствует флаг TEMPORARY инструкции CREATE TABLE (листинг 28.1). Доступ к временным таблицам имеет лишь тот поток, который их создал. Б MySQL существуют также резидентные таблицы, хранящиеся в памяти и уничтожаемые при завершении работы сервера. Тип таблицы задается с помощью опции TYPE инструкции CREATE TABLE.

```
CREATE TEMPORARY TABLE scratch (
    ID INT(11),
    Name CHAR(16),
    Score FLOAT
)
```

В MySQL реализованы концепции пользователей и привилегий, но не поддерживается понятие принадлежности базы данных или таблицы. Пользователю можно предоставить исключительное право доступа к таблице, но сама она не хранит связь с пользователем. В MySQL применяется сложная система привилегий, основанная на инструкциях GRANT и REVOKE, однако права доступа нельзя объединять в группы или роли.

В MySQL информация о привилегиях хранится в базе данных. При удалении таблицы или базы данных привилегии, относящиеся к удаленному объекту, остаются до тех пор, пока не будет выполнена инструкция REVOKE. Не полагайтесь на то, что сервер самостоятельно удалит привилегии.

Стандартные комментарии SQL начинаются с двух дефисов. В MySQL такой тип комментариев поддерживается, но за дефисами обязательно должен следовать пробел. Это позволяет избежать путаницы с математическими выражениями. Обратимся к табл. 28.1. Третья строка левой инструкции выглядит как комментарий, но ее можно рассматривать и как часть выражения 1 - (-1), что даст 2 в результате. В правой инструкции используются комментарии в стиле языка C.

в стиле языка SQL

```
UPDATE scratch
          =
   --1 (отладка)
 WHERE ID=3
```

Комментарии в стиле языка C

```
UPDATE scratch
          =
   /* 1 (отладка) */
 WHERE ID=3
```

Синтаксис комментариев рассматривался в главе 13, "Инструкции SQL". При переносе SQL-сценариев из другой СУБД в MySQL может потребоваться вставлять пробел в каждый комментарий.

В MySQL любое выражение со значением NULL равно NULL. NULL — это не то же самое, что False. Выражение NULL AND FALSE равно NULL, а не FALSE.

В некоторых СУБД поддерживается инструкция SELECT, позволяющая вставлять результаты запроса в другую таблицу. В MySQL для этих целей используется инструкция INSERT. В ней можно указывать не список значений в скобках, а подчиненную инструкцию SELECT.

Некоторые СУБД позволяют создавать таблицы с помощью инструкции SELECT. В табл. 28.2 демонстрируется создание временной копии таблицы. Левая инструкция записана в соответствии с синтаксисом MS SQL Server. Правая инструкция делает то же самое в MySQL.

Oracle

```
SELECT 'a' || 'b' || 'c'
```

Помимо внутренних объединений, создаваемых с помощью предложения WHERE, в MySQL используются специальные операторы объединений в предложении FROM. Например, поддерживается оператор LEFT JOIN. В других СУБД подобного рода операторы, такие как *= или (+), применяются в предложении WHERE.

Имена функций существенно различаются в зависимости от СУБД. Встроенные функции MySQL были перечислены вглаве 12, "Встроенные функции". Замену обычно найти несложно. Например, в Oracle текущее время хранится в переменной SYSDATE, в MS SQL Server оно определяется с помощью функции GETDATE(), а в MySQL—с помощью функции NOW().

Для совместимости с другими СУБД в MySQL создан ряд псевдонимов. В частности, имеется функция SYSDATE(), эквивалентная одноименной системной переменной Oracle. Поддерживаются также функции ODBC.

В MySQL реализован механизм транзакций, но лишь для некоторых типов таблиц. К ним не относится стандартный тип MyISAM, по крайней мере в версии 3.23. Отдельные SQL-инструкции выполняются в атомарном режиме, поэтому, если формировать собственные транзакции не нужно, таблицы MylSAM вполне подойдут.

Для таблиц тех типов, которые поддерживают транзакции, ведутся журналы транзакций. На их основе работают инструкции COMMIT и ROLLBACK. Если сервер внезапно зависает и его приходится принудительно перезапускать, прежнее состояние таблиц автоматически восстанавливается по журналу и целостность данных не нарушается. Таблицы MylSAM могут повреждаться из-за сбоев сервера. Обычно администраторы MySQL регулярно выполняют соответствующие проверки. Можно также настроить сервер на проверку всех таблиц при каждом запуске.

В MySQL таблицы блокируются при помощи инструкции LOCK TABLES. Есть СУБД, которые позволяют блокировать таблицу в инструкции SELECT. О вопросах блокировки рассказывалось в главах 9, "Транзакции и параллельные вычисления", и 13, "Инструкции SQL".

В главе 9 речь также шла об имитации последовательностей. В MySQL нет отдельного объекта, отвечающего за выделение уникальных идентификаторов. Вместо этого первичный ключ помечается специальным флагом, благодаря которому ему будут автоматически присваиваться целые числа. Кроме того, можно смоделировать последовательность с помощью таблицы, состоящей из одной ячейки. Обратимся к табл. 28.4. В левой части приведена инструкция Oracle, которая создает последовательность с интервалом приращения *5,* начинающуюся с числа 50. В правой части показаны эквивалентные инструкции MySQL.

Oracle

```
/* Создание последователь-
ности */
CREATE SEQUENCE invoice_seq
   INCREMENT BY 5
     START WITH  50;
```

MySQL

```
/* Создание последовательности */
CREATE TABLE invoice_seq (
   nextvalINT
);
INSERT INTO invoice_seq
     VALUES  (50);
```

Oracle	*MySQL*

`/* Получение следующего значения */`
 	
`SELECT invoice_seq.nextval;`
	`/* Получение следующего значения */`
	`UPDATE invoice_seq`
	`SET nextval = LAST_IN-`
	`SERT_ID(nextval+5);`
	`SELECT LAST INSERT ID();`

В MySQL не поддерживаются подчиненные запросы, т.е. инструкции SELECT, взятые в скобки и входящие в состав других инструкций SELECT. Обычно сложную инструкцию можно переписать так, чтобы удалить подчиненный запрос. Воспользуйтесь временными таблицами и блокировками. Если подчиненный запрос применяется для проверки того, входит ли запись в заданное множество, используйте методику, описанную выше для предиката EXISTS.

При работе с утилитой mysql подчиненный запрос можно сымитировать, создав файл сценария с текстом запроса. Представим себе таблицу, в которой хранятся сообщения интерактивного форума. Раз в месяц старые сообщения переносятся в архив и удаляются из главной таблицы. Таблица archive содержит идентификаторы всех записей таблицы message, которые сохраняются в архивном файле. Чтобы получить список удаляемых сообщений, нужно извлечь их идентификаторы из таблицы archive. В листинге 28.3 демонстрируется удаление записей таблицы message, соответствующих записям таблицы archive. Сначала с помощью функции CONCAT() создается группа инструкций, содержащих идентификаторы сообщений. В результате формируется таблица, состоящая из одного столбца. Затем благодаря предложению INTO OUTFILE этатаблица сохраняется в файле /tmp/mydel.sql. Таким образом, будет получен SQL-сценарий, содержащий одну инструкцию в каждой строке. В конец каждой строки добавляется точка с запятой. Сценарий загружается с помощью команды source, которая является частью интерпретатора mysql. Она загружает указанный файл и последовательно выполняет содержащиеся в нем инструкции, как если бы они были набраны в командной строке. При взаимодействии с сервером программным путем этот этап необходимо реализовать самостоятельно.

```
/* Сохранение таблицы в текстовом файле. */
SELECT CONCAT('DELETE FROM message WHERE ID=', MessageID, ';')
  FROM archive
  INTO OUTFILE '/tmp/mydel.sql';
/* Загрузка и выполнение файла инструкций. */
source /tmp/mydel.sql
```

Некоторые СУБД позволяют непосредственно управлять указателями наборов записей. Такие указатели можно перемещать с помощью инструкций SQL. В MySQL указатели используются самой программой и недоступны на уровне инструкций. В то же время результаты запроса можно извлекать построчно с помощью функций библиотеки языка C.

В большинстве реляционных СУБД поддерживаются хранимые процедуры и пользовательские функции. Они пишутся на специальных языках сценариев, таких как PL/SQL или Transact-SQL. В MySQL аналогичного языка нет. Существуют два способа имитации хранимых процедур. Во-первых, соответствующие функции можно реализовать в приложениях, а во-вторых, можно писать функции MySQL на языке C (об этом рассказывается в главе 31, "Расширение возможностей MySQL").

С хранимыми процедурами связана и концепция триггеров. СУБД выполняет триггерные процедуры при наступлении определенных событий. Это позволяет накладывать на таблицу произвольные ограничения целостности. Как правило, это каскадные удаления, когда удаление родительской записи приводит к удалению всех ее "потомков". Ограничения налагаются также на значения столбцов. Такие столбцы удобно определять с помощью типа ENUM, задавая набор допустимых строк. Соответствующий пример показан в листинге 28.4. Разработчики MySQL планируют в будущем включить в программу поддержку каскадных удалений.

```
CREATE TABLE member (
    ID INT(5) NOT NULL AUTO_INCREMENT,
    Name CHAR(16) NOT NULL,
    Department ENUM(
        'Command',
        'Engineering',
        'Science') NOT NULL,
    PRIMARY KEY(ID)

)
```

В MySQL распознаются инструкции CREATE TABLE, содержащие определения внешних ключей, хотя эти определения отбрасываются при создании таблицы. Программа не пытается контролировать целостность внешних ключей. В документации приводятся веские доводы в пользу такого шага. Считается, что целостность межтабличных связей должна контролироваться приложениями.

Представление — это инструкция SELECT, которая получила имя и выглядит как таблица. Представления необходимы для того, чтобы заставить пользователей обращаться к некоторым таблицам строго определенным образом и без возможности обновления. Планируется, что представления появятся в MySQL версии 4.1, а пока что запрет записи в таблицу реализуется только посредством привилегий.

Некоторые СУБД используют фиксированный порядок сортировки записей, а в других СУБД этот порядок указывается перед созданием базы данных. MySQL не учитывает регистр символов в операциях сравнения, но эту установку можно отменить, задав в определении столбца флаг BINARY. Существует также оператор приведения типа BINARY, описанный в главе 12, "Встроенные функции". Порядок сортировки двоичных строк определяется используемым набором символов. По умолчанию это ISO-8859-1.

В MySQL существуют операторы REGEXP и NOT REGEXP, позволяющие сравнивать строку с регулярным выражением. Некоторые СУБД разрешают указывать регулярные выражения в операторе LIKE, но в MySQL концепция регулярных выражений реализована в полном объеме.

Использование режима ANSI

Демон `mysqld` имеет опцию командной строки `--ansi`, позволяющую включить режим ANSI. В этом режиме поведение программы несколько меняется с целью улучшения совместимости со стандартом SQL-92. Ниже перечислены отличительные особенности режима ANSI.

Две вертикальные черты (`||`) теперь представляют собой оператор конкатенации строк, а не оператор логического сложения. Обычно для совместимости с другими СУБД приходится пользоваться функцией `CONCAT()`.

Между именем функции и открывающей скобкой разрешается ставить пробел. Обычно программа MySQL запрещает это делать, чтобы можно было отличать имена функций от имен столбцов. В режиме ANSI имена всех функций становятся зарезервированными словами, т.е. их нельзя использовать в качестве имен таблиц или столбцов.

В режиме ANSI строковые литералы разрешается заключать только в одинарные кавычки. Обычно MySQL разрешает использовать двойные кавычки, но стандарт требует, чтобы в такие кавычки брались только идентификаторы с пробелами. Например, если таблица называется `my table`, то ее имя нужно заключать в кавычки, иначе оно будет воспринято как два разных имени. В обычном режиме такие имена берутся в обратные кавычки.

Тип `REAL` в MySQL обычно преобразуется в тип `DOUBLE`. Однако в режиме ANSI ему соответствует тип `FLOAT`.

Стандартным уровнем изоляции транзакций в режиме ANSI является уровень `SERIALIZABLE` (см. главу 9 "Транзакции и параллельные вычисления").

Уникальные свойства MySQL

Подобно всем реляционным СУБД, MySQL содержит целый ряд расширений стандарта SQL. Если в будущем планируется переход на другую СУБД, постарайтесь не задействовать эти расширения. Правда, я придерживаюсь противоположного принципа: использовать все, что возможно. Большинство расширений обеспечивает повышение производительности, что иногда приводит к отказу от перехода на якобы более мощную СУБД. Кроме того, некоторые приложения имеют короткое время жизни. Это особенно справедливо в отношении Web-приложений. Их универсализация бесполезна, если их все равно придется писать заново.

В MySQL поддерживаются комментарии особого вида, позволяющие скрывать код от других СУБД. Такие комментарии записываются в стиле языка C, но текст комментария начинается с восклицательного знака. Программа MySQL распознает код, заключенный в такие комментарии, тогда как другие СУБД игнорируют их. После знака ! можно указать минимально допустимый номер версии MySQL. Таким способом помечаются "спорные" фрагменты инструкций. В листинге 28.5 приведена инструкция, некоторые фрагменты которой выходят за рамки стандарта SQL. В последней строке содержится требование, чтобы таблица MyISAM создавалась только в MySQL версии 3.23 или выше. Можно указывать более точный номер версии, например 32335 (т.е. 3.23.35).

```
CREATE TABLE /*! IF NOT EXISTS */ player (
    /* Столбцы */
    ID INT(11) /*! UNSIGNED */ NOT NULL /*! AUTO_INCREMENT */,
    Nickname CHAR(8) NOT NULL,
    Password CHAR(8) NOT NULL,
    Rank FLOAT(4,2) NOT NULL /*! DEFAULT 50.0 */,
    PRIMARY KEY (ID)
) /*!323 TYPE=MyISAM */
```

Программа MySQL придерживается правил операционной системы в отношении чувствительности к регистру имен баз данных и таблиц. В Windows регистр не важен, а в UNIX— важен. Большинство СУБД придерживается первого варианта. Имена столбцов в MySQL тоже нечувствительны к регистру. Правда, в сравнении с другими СУБД, MySQL допускает большую гибкость. К примеру, разрешаются имена, начинающиеся с цифр, а максимальная длина имен обычно больше, чем в других СУБД.

Все СУБД по-разному создают базы данных, поэтому о длине имен баз данных можно не беспокоиться. Инструкция CREATE DATABASE в MySQL не является частью стандарта. Что касается длины имен таблиц и столбцов, то многие СУБД ограничивают ее тридцатью двумя символами. Желательно не выходить за рамки этого ограничения, иначе при переносе базы данных придется переименовывать ее объекты. Убедитесь также, что никакие две базы данных или две таблицы одной базы данных не названы одинаково, за исключением регистра символов, например `StoreDB` и `storedb`.

Стандарт требует, чтобы при ссылке на столбцы использовалась точечная нотация вида *таблица.столбец*. Реляционные СУБД по-разному расширяют это требование. MySQL позволяет добавлять слева имя базы данных: *база_данных.таблица.столбец*. В соответствующем контексте можно ссылаться на имя таблицы, перед которым стоит имя базы данных. Чтобы избежать проблем с другими СУБД, старайтесь придерживаться стандарта.

В стандарте SQL одинарная кавычка защищается от интерпретации удвоением (''). В MySQL для этой цели можно использовать символ \. О защите специальных символов рассказывалось в главе 9, "Транзакции и параллельные вычисления". Строки можно заключать в одинарные или двойные кавычки.

В MySQL операторы || и && обозначают операции логического сложения и умножения соответственно. Вместо них всегда можно использовать ключевые слова OR и AND. В стандарте SQL оператор || обозначает конкатенацию строк. Если забыть об этом, то при переносе базы данных в другую СУБД могут возникнуть трудно обнаруживаемые ошибки.

В MySQL поддерживается оператор `%`, который эквивалентен функции `MOD()`.

В списке возвращаемых столбцов инструкции SELECT можно использовать операторы сравнения, описанные в главе 10, "Типы данных, переменные и выражения". Результатом сравнения будет нуль или единица, в зависимости от того, является выражение истинным или ложным.

Стандартный способ присваивания переменных заключается в использовании инструкции SET. В MySQL разрешается также задавать значения переменных в списке возвращаемых столбцов инструкции SELECT. Для этого предназначен оператор `:=`.

Операторы REGEXP, RLIKE, NOT REGEXP и NOT RLIKE позволяют сравнивать значение с регулярным выражением. В некоторых СУБД для этих целей расширены возможности оператора LIKE.

MySQL разрешает применять оператор LIKE к любым столбцам, даже числовым. Перед сравнением с шаблоном числовые значения приводятся к строковому типу. Например, можно записать LIKE '2%', чтобы найти целые числа, начинающиеся с цифры 2.

В MySQL есть функция LAST_INSERT_ID(), возвращающая последнее значение поля-счетчика. Если первичный ключ является целым числом, на него всегда можно сослаться по специальному имени _rowid. Если же первичного ключа нет, этому имени соответствует столбец типа UNIQUE, при условии, что таковой имеется.

В стандарте SQL определено, что функция CONCAT() принимает два аргумента. В тех СУБД, которые соответствуют этой части стандарта, нужно использовать вложенные вызовы функции при объединении группы строк. MySQL не ограничивает число аргументов данной функции. Функция CHAR() тоже имеет произвольное число аргументов.

Помимо описанных выше функций в MySQL поддерживаются следующие нестандартные функции: BIT_AND(), BIT_COUNT(), BIT_OR(), CASE(), DECODE(), ELT(), ENCODE(), ENCRYPT(), FORMAT(), FROM_DAYS(), IF(), MD5(), PASSWORD(), PERIOD_ADD(), PERIOD_DIFF(), STD(), TO_DAYS() и WEEKDAY().

В MySQL есть несколько нестандартных типов столбцов. Типы ENUM и SET позволяют определить диапазон допустимых значений столбца. Поддерживаются также производные типы, расширяющие или, наоборот, сокращающие размерность стандартного типа данных. Например, значение типа MEDIUMINT занимает три байта вместо четырех, а тип LONGBLOB поддерживает строки длиной до 4 Гбайт.

Строки типа VARCHAR в MySQL теряют хвостовые пробелы при записи в таблицу, в то время как стандарт требует их сохранять. В будущих версиях MySQL данная установка может измениться.

При обсуждении инструкции CREATE TABLE в главе 13, "Инструкции SQL", рассказывалось о том, каким образом программа MySQL меняет определения столбцов. В таблицах с динамическими записями столбцы CHAR превратятся в VARCHAR, если их размерность составляет более четырех символов. И наоборот, более короткие столбцы типа VARCHAR будут приведены к типу CHAR в целях экономии места на диске.

В MySQL поддерживается флаг AUTO_INCREMENT, позволяющий реализовывать уникальные идентификаторы записей. Если столбец помечен таким флагом, то при вставке записи программа поместит в этот столбец следующее значение по порядку. В Oracle для этих целей применяются последовательности, которые можно имитировать с помощью таблицы, состоящей из одной ячейки.

В листинге 28.6 демонстрируется универсальный способ создания последовательности при помощи механизма транзакций. Это не самый эффективный способ получения уникальных идентификаторов в MySQL, но он должен работать в тех СУБД, где поддерживаются транзакции. Стандартный тип таблиц – MyISAM – несовместим с транзакциями, поэтому в примере создается таблица Berkeley DB. Последовательность начинается с нуля.

```
/* Создание последовательности. */
CREATE TABLE invoice_seq (
   nextval INT
) TYPE=Berkeley_DB;
INSERT INTO invoice_seq VALUES (0);

/* Получение следующего значения. */
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
BEGIN;
UPDATE invoice_seq
   SET nextval = nextval+1;
SELECT nextval
  FROM invoice_seq;
COMMIT;
```

Инструкция SET TRANSACTION подчеркивает необходимость сериализации в данной методике. В рамках единой транзакции сценарий увеличивает порядковый номер и тут же запрашивает его. Если все потоки будут придерживаться такой методики, можно будет использовать идентификаторы, не опасаясь того, что какой-то из них встретится дважды.

Столбцы, не помеченные флагом NOT NULL, могут содержать значения NULL. В MySQL можно также использовать флаг NULL, чего нет в стандарте. Флаги UNSIGNED и ZEROFILL тоже являются расширениями стандарта. Первый из них модифицирует диапазон возможных значений столбца за счет знакового разряда. Например, столбец SMALLINT допускает значения от -32768 до 32767, а столбец UNSIGNED SMALLINT– от 0 до 65535. Значения, выходящие за границы знакового диапазона, могут отвергаться или неправильно интерпретироваться другими СУБД.

MySQL разрешает указывать флаг DISTINCT внутри статистической функции COUNT(). Например, запрос, показанный в листинге 28.7, является допустимым. Другие СУБД могут не понимать такой синтаксис.

```
SELECT COUNT(DISTINCT Producer, Contact)
  FROM project;
```

В MySQL разрешается определять произвольное число индексов в инструкции CREATE TABLE, тогда как в стандарте допускаются только первичный и внешние ключи. Другие СУБД требуют, чтобы индексы создавались отдельно с помощью инструкции CREATE INDEX. В MySQL индексы могут быть именованными. Можно индексировать фрагменты столбцов, начиная с самого левого символа. Это называется частичными индексами. Поддерживаются также полнотекстовые индексы столбцов BLOB и TEXT.

ЕСЛИ В инструкции CREATE TABLE присутствует флаг TEMPORARY, программа MySQL создаст временную таблицу, которая будет уничтожена по окончании сеанса. В других СУБД применяется иной синтаксис работы с временными таблицами.

При наличии флага IF NOT EXISTS инструкция CREATE TABLE не будет создавать таблицу, если она уже существует. У инструкции DROP TABLE есть похожий флаг IF EXISTS. В отличие от других СУБД, MySQL разрешает удалять несколько таблиц в одной инструкции.

В MySQL у инструкции SELECT есть предложение INTO OUTFILE, позволяющее сохранять результаты запроса в файле. Инструкция LOAD DATA INFILE загружает данные из файла в таблицу.

В других СУБД столбец, указанный в предложении GROUP BY, должен входить в таблицу результатов запроса. В MySQL данное ограничение отсутствует. В предложении GROUP BY допускаются флаги ASC и DESC.

Флаг STRAIGHT_JOIN заставляет программу выполнить объединение таблиц в указанном порядке. В Oracle пользователи могут включать в комментарии подсказки для модуля оптимизации, служащие аналогичным целям. Инструкция EXPLAIN возвращает информацию о том, как именно выполняется объединение. На основании этого можно делать выводы о необходимости добавления индексов (см. главу 26, "Оптимизация").

Возможности модификации таблиц в MySQL заметно превосходят возможности других СУБД. К примеру, многие СУБД позволяют добавлять столбцы, но не разрешают удалять их или менять их определения. В MySQL все это возможно. Если изменение затрагивает существующие значения, программа попытается их преобразовать. Инструкция RENAME TABLE непосредственно меняет имя таблицы. Инструкция ALTER TABLE позволяет вносить множественные изменения в определение таблицы.

Инструкция DELETE содержит предложение LIMIT, которое ограничивает количество удаляемых записей. Это позволяет эффективнее организовать многопользовательскую работу с базой данных. На время выполнения инструкции DELETE таблица блокируется, поэтому большие объемы записей лучше удалять не целиком, а небольшими "порциями", чтобы другие потоки не блокировались надолго.

Запросы на чтение и на запись заносятся в разные очереди, причем запросы на запись имеют более высокий приоритет. Эту установку можно отменить с помощью флага DELAYED инструкции INSERT, флага LOW_PRIORITY инструкции UPDATE или флага HIGH_PRIORITY инструкции SELECT.

Инструкция REPLACE является особой разновидностью инструкции INSERT. Она удаляет существующие записи, если значения их первичных ключей противоречат первичным ключам вставляемых записей. В результате пропадает необходимость в связке DELETE/INSERT.

Инструкции ANALYZE TABLE, CHECK TABLE, OPTIMIZE TABLE и REPAIR TABLE позволяют контролировать таблицы и индексы и восстанавливать их в случае повреждений.

Инструкции семейства SHOW возвращают различную информацию о структуре баз данных и о состоянии сервера. В других СУБД информация о схемах хранится в главной базе данных, но в MySQL центральная база данных используется только для привилегий и загруженных функций. Имея соответствующие привилегии, можно, к примеру, просмотреть список активных соединений с помощью инструкции SHOW PROCESSLIST. Обычно это приходится делать средствами операционной системы.

Инструкция SHOW VARIABLES возвращает перечень серверных установок. С помощью инструкции SET можно менять параметры сервера, не редактируя конфигурационные файлы. Например, инструкция SET CHARACTER SET задает новый стандартный набор символов.

Для повышения производительности в MySQL применяются различные кэш-буферы. Иногда приходится очищать их содержимое. Для этого предназначена инструкция FLUSH.

Администраторы MySQLмогут создавать базы данных не с помощью специального интерфейса, а посредством инструкции CREATE DATABASE. Инструкция DROP DATABASE удаляет базуданных и все ее таблицы.

РАСПРЕДЕЛЕННЫЕ БАЗЫ ДАННЫХ

В этой главе…

Концепции распределенных баз данных
Отложенная синхронизация
Репликация в MySQL
Запуск нескольких серверов

Глава 29

В этой главе рассматриваются концепции распределенных баз данных. Такие базы данных хранятся и обрабатываются на нескольких компьютерах, что повышает их производительность, доступность и устойчивость к сбоям. Чтобы все компьютеры оставались синхронизированными, по сети должны рассылаться обновления. В идеальной ситуации распределенная база данных выглядит для клиентов как единый, монолитный сервер.

В MySQL поддерживается репликация данных, что позволяет запускать несколько серверов, синхронизирующих свои базы данных. Сервер может быть главным, подчиненным или и тем и другим одновременно. Главный сервер фиксирует все изменения и делает их доступными для одного или нескольких подчиненных серверов. Последние, в свою очередь, запрашивают обновления у одного главного сервера.

В этой главе рассказывается также о том, как запустить несколько серверов MySQL на одном компьютере. Данная методика позволяет повысить безопасность серверов, которые работают в многопользовательском режиме.

Концепции распределенных баз данных

Обычный сервер хранит у себя все данные и обслуживает все клиентские запросы. Схема взаимодействия между сервером и клиентами изображена на рис. 29.1. Серверные данные располагаются на одном или нескольких физических дисках. Чтобы сделать запрос, клиент устанавливает соединение с сервером. Сервер анализирует инструкции, выполняет их, извлекает данные и возвращает результаты запроса.

По мере возрастания нагрузки производительность сервера снижается. Чтобы избежать этого, задействуют дополнительные ресурсы, например наращивают память, ставят дополнительные процессоры и даже сетевые платы. Это эффективная стратегия, если клиенты расположены в непосредственной близости от сервера, например несколько серверов приложений взаимодействуют с одной СУБД. Но в тех архитектурах, где сервер и клиенты удалены друг от друга, производительность обратно пропорциональна расстоянию.

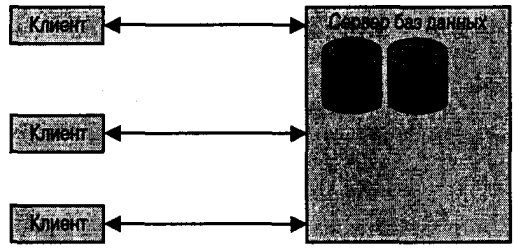


Рис. 29.1. Обычный сервер баз данных

Решением этой проблемы являются распределенные базы данных (РБД), которые сегментируют хранимую информацию и перемещают отдельные ее блоки ближе к нужным клиентам. Способов организации таких баз данных много. Можно разместить таблицы на разных компьютерах или использовать несколько идентичных хранилищ. Во втором случае серверы взаимодействуют друг с другом для поддержания синхронизации. Если на одном из серверов происходит обновление данных, оно распространяется и на все остальные серверы.

К недостаткам распределенных баз данных можно отнести то, что возрастает сложность управления ими. Но преимуществ все же больше. Главное из них – повышение производительности. Данные быстрее обрабатываются несколькими серверами, а кроме того, данные располагаются ближе к тем пользователям, которые чаще с ними работают.

Система становится более устойчивой, если она способна выдержать сбой одного из своих компонентов. В распределенной базе данных с симметричной схемой хранения исчезновение одного из серверов приводит к замедлению работы пользователей, находящихся ближе к этому серверу, но в целом система остается работоспособной. К тому же, она легко масштабируется, так как ее не нужно останавливать при добавлении еще одного сервера.

В несимметричной системе можно оптимизировать схему расположения данных. Чем ближе пользователи находятся к нужным им данным, тем меньше на их работу влияют сетевые задержки. В результате серверам приходится обрабатывать меньшие объемы данных. Это также способствует повышению безопасности данных, поскольку их можно физически хранить в тех системах, где пользователи имеют право работать с соответствующими данными. В целом, однако, применение распределенных баз данных связано с достаточно высоким риском. Требуется обеспечить соблюдение мер безопасности сразу на нескольких узлах, что не так-то просто реализовать.

Распределенные базы данных трудно проектировать и обслуживать. Порядок работы в системе может со временем поменяться, что повлечет за собой изменение схемы хранения данных. Как клиенты, так и серверы должны уметь обрабатывать запросы к данным, которые не расположены в ближайшей системе. Плохо спроектированная РБД может демонстрировать меньшую производительность, чем одиночный сервер.

РБД состоит из трех основных частей: клиентов, модуля обработки транзакций и хранилища данных. Сервер обычно берет на себя задачи обработки транзакций и хранения данных, хотя в полностью распределенной базе данных за решение этих задач отвечают разные аппаратные компоненты. Обратимся к рис. 29.2. Здесь три клиента взаимодействуют с двумя модулями обработки транзакций, которые, в свою очередь, работают с двумя хранилищами. Клиенты посылают свои запросы модулям, а те определяют, в каком из хранилищ находятся требуемые данные.

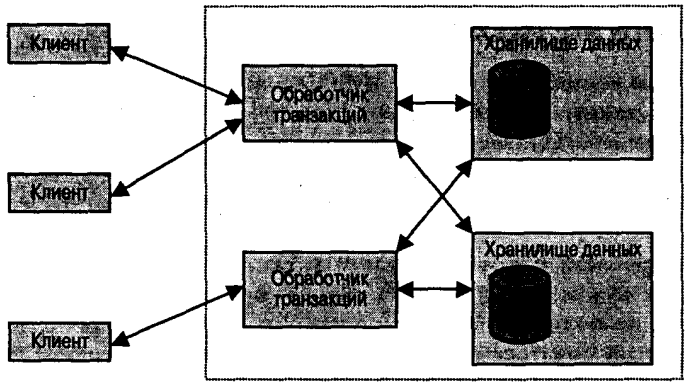


Рис. 29.2. Распределенные серверы

В идеале клиенты не знают, является система распределенной или нет. Они лишь посылают ей запросы, а система возвращает клиентам результаты этих запросов. Как она это делает, клиентов не интересует. На практике распределенные базы данных проявляют разную "прозрачность". В крайнем случае РБД хранится на нескольких независимых серверах, а клиентскому приложению приходится выбирать сервер в зависимости от того, какую информацию требуется получить. Это подразумевает, что таблицы, находящиеся на разных серверах, не имеют никаких внутренних связей. Естественно, такая организация РБД лишь изредка оказывается полезной.

Система управления распределенными базами данных, или **РСУБД**, предоставляет клиентам унифицированный интерфейс доступа к данным, благодаря которому возникает иллюзия единого сервера. Если данные находятся в разных местах, РСУБД посылает запросы и обновления в соответствующие хранилища. В зависимости от того, с каким хранилищем ведется работа, производительность системы может оказаться разной, но, по крайней мере, клиентам не приходится самим заниматься выбором сервера.

Если данные реплицируются между несколькими серверами, клиент в общем случае может предпочесть тот или иной сервер. В подобной схеме все серверы хранят одни и те же данные. Специальный модуль может помогать клиентам в выборе серверов, осуществляя выравнивание нагрузки. РСУБД отвечает за выполнение транзакций в многосерверной среде, но в любой момент времени два сервера не могут быть синхронизированы между собой.

В РБД применяется несколько схем распределения данных. В случае репликации каждый сервер хранит весь объем данных. Для этого требуется, чтобы РСУБД дублировала транзакции, позволяя всем клиентам видеть согласованный образ базы данных. В случае несимметричного разделения данных выбирается уровень сегментации. На самом высоком уровне расщеплению подвергаются отдельные базы данных, но не таблицы. Каждая таблица целиком находится в каком-то одном месте. На более низком уровне таблицы расщепляются по строкам или столбцам. Например, при горизонтальном расщеплении отдельные подмножества записей помещаются в разные хранилища, а при вертикальном расщеплении подмножества формируются на основании столбцов.

Отложенная синхронизация

Создать точную копию базы данных MySQL довольно просто. Способы резервного копирования баз данных описывались в главе 25, "Устранение последствий катастроф". Что касается восстановления данных, то это можно сделать на любом сервере. Располагая такими средствами, несложно реализовать распределенную базу данных, которая синхронизируется через достаточно большие промежутки времени, например раз в день.

Рассмотрим проблему обновления данных. Если обновления происходят сразу на двух серверах, их нужно согласовывать. Чтобы не возникали неразрешимые ситуации, необходимо позволить вносить изменения только на одном сервере. Тогда синхронизация будет заключаться в дублировании содержимого сервера, доступного для записи, на все остальные серверы. Их полезность зависит от того, насколько важна актуальность данных. Во многих случаях база данных, содержащая все записи, кроме тех, которые были созданы за последние 24 часа, вполнеприемлема.

Методика отложенной синхронизации идеально подходит для баз данных, содержащих результаты ночных отчетов. Например, Web-узел, предоставляющий доступ к MP3-файлам, может регистрировать названия запрашиваемых песен и составлять рейтинги популярности. Раз в день все серверы посылают свои журнальные файлы главному серверу, который корректирует рейтинги согласно новой статистике. Схема такой базы данных приведена в листинге 29.1.

Листинг 29.1. Схема распределенной

```
/* Каталог MP3-файлов. */
CREATE TABLE song (
    ID INT NOT NULL AUTO_INCREMENT,
    Name CHAR(40) NOT NULL,
    Artist CHAR(16) NOT NULL,
    Filename CHAR(80) NOT NULL,
    PRIMARY KEY(ID),
    INDEX (Name),
    INDEX (Artist)
);

/* Журнал запрашиваемых файлов. */
CREATE TABLE log (
    Server TINYINT UNSIGNED NOT NULL,
    Song INT NOT NULL,
    DownloadTime DATETIME NOT NULL
);
```

Каждый сервер хранит информацию о доступных песнях в таблице song. В таблицу log заносится запись всякий раз, когда кто-то загружает очередной MP3-файл. У этой таблицы нет первичного ключа, так как в обычном режиме она используется лишь для вставкизаписей. Наличие индекса только замедлит работу с таблицей.

Раз в день таблицы log всех серверов объединяются по следующему сценарию. Один из серверов прекращает обслуживать запросы Web-приложений. Остальные серверы создают копии таблицы log, извлекают из них последние записи и посылают

их серверу, генерирующему отчет. На время этой процедуры каждый сервер должен заблокировать свою таблицу log. Чтобы слишком много пользовательских запросов не оказалось заблокировано, процедуру желательно выполнять в период минимальной активности сервера.

Сервер, генерирующий отчет, загружает полученные данные в свою таблицу log, после чего выполняет сценарий, показанный в листинге 29.2. В этом сценарии создаются два рейтинга популярности: по всем песням и за последние 24 часа. Временный индекс таблицы log ускоряет ее просмотр.

```
/* Создание индекса. */
CREATE INDEX song ON log (Song);

/* Определение рейтинга всех песен. */
DROP TABLE IF EXISTS popular_alltime;
CREATE TABLE popular_alltime (
   Rank INT NOT NULL,
   Song INT NOT NULL,
   Hits INT NOT NULL
);
SET @id = 0;
INSERT INTO popular_alltime
   SELECT (@id := @id+1), Song, COUNT(*)
     FROM log
    GROUP BY 2
    ORDER BY 3 DESC;
ALTER TABLE popular_alltime
  ADD PRIMARY KEY(Rank);

/* Определение рейтинга песен за последние 24 часа. */
DROP TABLE IF EXISTS popular_last24;
CREATE TABLE popular_last24 (
   Rank INT NOT NULL,
   Song INT NOT NULL,
   Hits INT NOT NULL
);
SET @id = 0;
INSERT INTO popular_last24
   SELECT (@id := @id+1), Song, COUNT(*)
     FROM log
    WHERE DownloadTime > DATE_SUB(NOW(), INTERVAL 24 HOUR)
    GROUP BY 2
    ORDER BY 3 DESC;
ALTER TABLE popular_last24
  ADD PRIMARY KEY(Rank);

/* Удаление индекса. */
DROP INDEX song ON log;
```

Таблицы рейтингов не содержат **поле-счетчик,** вместо него используется **инкрементная** переменная. Будучи созданной, рейтинговая таблица уже не меняется. Сценарий удаляет существующие таблицы и формирует их заново. Имеет смысл сжимать такие таблицы с помощью утилиты `myisampack` (см. главу 14, "Утилиты командной строки"). Созданные таблицы загружаются остальными серверами, где они заменяют старые рейтинги.

Изменения в таблицу song должны вноситься на одном сервере. Тогда исчезают проблемы, связанные с дублированием песен и первичных ключей. Главный сервер публикует новую редакцию каталога песен либо после каждого изменения, либо по определенному графику.

Отложенная синхронизация удобна там, где пользователям не нужны отчеты в реальном времени. В приведенном выше примере число операций записи в журнальную таблицу превышает число обращений к рейтингам, если учесть, что пользователь, просматривающий список десяти лучших песен, наверняка захочет загрузить хотя бы одну из них. Методика срабатывает, поскольку информация, представленная в рейтинге, не имеет непосредственной ценности.

В описанной схеме требуется, чтобы приложения знали архитектуру всей системы. Эта система не является монолитной и легко справляется со сбоями отдельных **компонентов.** Каждый из составляющих ее серверов хранит идентичную информацию.

Репликация в MySQL

Программа MySQL реализует функции автоматической репликации данных между главным и подчиненными серверами. Главный сервер ведет журнал изменений, который делается доступным для одного или нескольких подчиненных серверов. Простейшая схема репликации с участием одного главного и двух подчиненных серверов изображена на рис. 29.3.

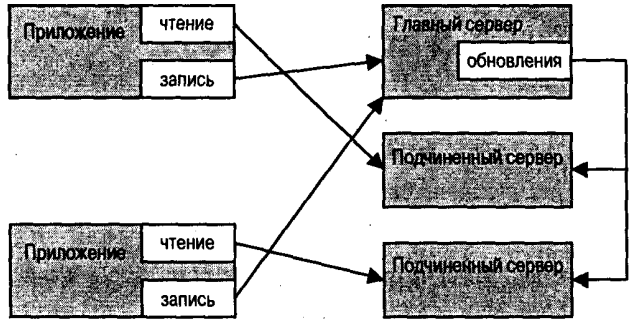


Рис. 29.3. Схема репликации с одним сервером для записи и двумя серверами для чтения

Все серверы работают под управлением MySQL на разных **компьютерах,** подключенных к сети. В целях синхронизации подчиненные серверы регулярно связываются с главным сервером. Последний фиксирует каждое изменение в двоичном журнале (см. главу 24, "Физическое хранение данных"). Если не считать короткие промежутки времени, в течение которых происходит синхронизация, подчиненные серверы содержат зеркальные копии базы данных.

Клиенты могут посылать запросы на выборку как **главному,** так и любому подчиненному серверу. Запись данных должна происходить только на главном сервере, иначе изменения останутся локальными. В схеме на рис. 29.3 запросы на выборку направляются только подчиненным серверам, что позволяет главному серверу **сосредоточиться** на обновлениях.

На подчиненном сервере запускается отдельный поток, который периодически опрашивает главный сервер на предмет наличия изменений. Однако бывает так, что сервер одновременно является и главным, и подчиненным. Тогда становится возможной схема репликации, изображенная на рис. 29.4. Здесь есть четыре сервера, каждый из которых играет двойную роль. Приложение направляет "своему" серверу запросы как на выборку, так и на обновление данных. Обновление, произошедшее на одном сервере, передается по цепочке остальным серверам.

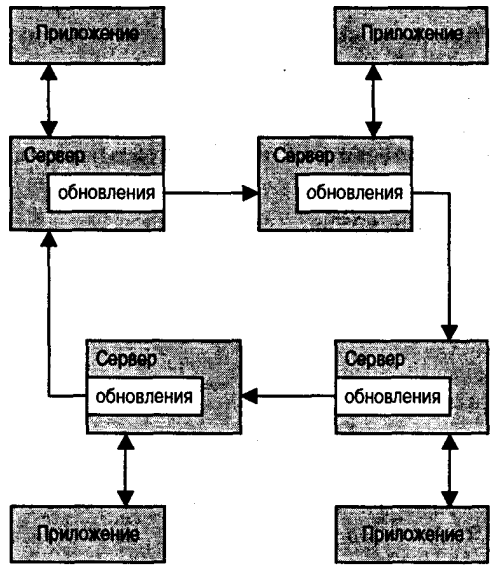


Рис. *29.4.* Круговая репликация

Приложение должно предотвращать несогласованные обновления. Рассмотрим такой случай. Серверы А и Б являются друг для друга главным и подчиненным сервером. Оба используют таблицу сообщений с первичным **ключом-счетчиком.** В определенный момент два пользователя одновременно создают новые сообщения, по одному на каждом сервере. Если таблицы раньше были синхронизированы, то теперь в каждой из них будет новая запись с идентификатором, совпадающим с идентификатором другой записи на другом сервере. Можно избежать этой неприятности, используя внешний генератор идентификаторов, при условии, что он будет **однопотоковым.**

В схеме на рис. 29.4 каждое приложение взаимодействует с одним сервером, хотя это необязательно. Если серверы расположены рядом, можно формировать пул **серверов.** Тогда при выборе сервера приложение сможет учитывать возможность сбоя или применять алгоритм выравнивания нагрузки.

Подчиненный сервер контролирует, какое изменение было получено от главного сервера последним. По умолчанию сведения об этом находятся в файле `master.info`.

Если обновление приводит к появлению ошибки, то поток, управляющий синхронизацией, останавливается. В этом случае сервер записывает сообщение в журнал ошибок.

В MySQL версии 3.23.39 механизм репликации имеет ряд **ограничений**. В основном все они связаны с тем, что изменения фиксируются только в двоичном журнале. Функция RAND() по умолчанию инициализирует генератор псевдослучайных чисел значением системных часов, но это значение не сохраняется в двоичном журнале. В подобной ситуации можно инициализировать генератор самостоятельно с помощью функции UNIX_TIMESTAMP().

Вспомните, что изменения таблиц привилегий в базе данных mysql не вступят в силу, пока не будут очищены буферы привилегий. Но дело в том, что в двоичном журнале не отслеживаются инструкции FLUSH, поэтому старайтесь не менять таблицы напрямую, а пользуйтесь инструкциями GRANT и REVOKE.

Значения переменных не реплицируются. Если в обновлении строки участвовала переменная, то на подчиненном сервере будет использовано локальное, а не реальное значение этой переменной.

Чтобы включить функции репликации, нужно отредактировать конфигурационные файлы сервера. Ниже перечислены опции демона mysqld, имеющие отношение к репликации. Все они были описаны в главе 14, "Утилиты командной строки".

```
--binlog-do-db=база_данных
--binlog-ignore-db=база_данных
--log-bin-index=каталог
--log-bin[=каталог]
--log-slave-updates
--master-connect-retry=секунды
--master-host=узел
--master-info-file=файл
--master-password=пароль
--master-port=порт
--master-user=пользователь
--replicate-do-db=база_даных
--replicate-do-table=база_данных.таблица
--replicate-ignore-db=база_данных
--replicate-ignore-table=база_данных.таблица
--replicate-rewrite-db=главный->подчиненный
--replicate-wild-do-table=шаблон
--replicate-wild-ignore-table=шаблон
--server-id=идентификатор
```

Прежде чем редактировать конфигурационные файлы, создайте на главном сервере учетную запись, чтобы подчиненный сервер мог получать обновления. В листинге 29.3 показана инструкция, создающая пользователя user с привилегией FILE. Никакие другие привилегии подчиненному серверу не нужны. В данном примере пользователь может посылать запросы с любого узла, но на практике обычно указывают конкретное доменное имя.

```
GRANT FILE ON *.* TO slave IDENTIFIED BY 'password';
```

Теперь нужно остановить главный сервер или заблокировать все таблицы, подлежащие репликации. Создайте точные копии реплицируемых баз данных и лишь затем редактируйте конфигурационные файлы. Вспомните, что таблицы MySQL имеют системно-независимый формат. Это позволяет копировать и перемещать файлы, не меняя их формат. В листинге 29.4 создается tar-архив одной базы данных. Можно также выполнить на подчиненном сервере инструкцию LOAD TABLE, чтобы получить точную копию нужной таблицы.

```
[/usr/local/var]# tar cvfz freetimedb.tar.gz ./freetime/
```

Далее требуется включить двоичный журнал на главном сервере, внеся изменения в конфигурационный файл `/etc/my.cnf`. Соответствующая опция называется log-bin. С помощью опции `server-id` серверу присваивается уникальный идентификатор. В листинге 29.5 показаны две строки, которые добавляются в раздел `[mysqld]` конфигурационного файла. После внесения изменений нужно перезапустить главный сервер.

```
log-bin
server-id=1
```

Скопируйте на подчиненный сервер созданную выше копию базы данных и отредактируйте его конфигурационный файл. Добавляемые в него опции перечислены в листинге 29.6. Поскольку реплицируется лишь одна база данных, с помощью опции `replicate-do-db` задается конкретное имя: `freetime`. Данному подчиненному серверу присваивается идентификатор 2.

```
master-host=192.168.123.194
master-user=slave
master-password=password
master-port=3306
replicate-do-db=freetime
server-id=2
```

После изменения конфигурации запустите подчиненный сервер. Он свяжется с главным севером и проверит, изменилась ли база данных `freetime` с момента получения ее последней копии. Пока подчиненный сервер работает, он регулярно получает интересующие его изменения. Задержка обновления зависит от контекста, но обычно она составляет несколько секунд.

В листинге 29.7 перечислены инструкции, используемые в процессе репликации. Их описание можно найти в главе 13, "Инструкции SQL". Инструкция CHANGE MASTER приводит к временной смене главного сервера. Она полезна в том случае, ко-

```
mysql> SHOW MASTER STATUS;
+-------------+----------+--------------+------------------+
| File        | Position | Binlog_do_db | Binlog_ignore_db |
+-------------+----------+--------------+------------------+
| red-bin.002 | 312      |              |                  |
+-------------+----------+--------------+------------------+
1 row in set (0.00 sec)
```

```
mysql> SHOW SLAVE STATUS \G
*************************** 1. row ***************************
             Master_Host: 192.168.123.194
             Master_User: slave
             Master_Port: 3306
           Connect_retry: 60
                 Log_File: red-bin.002
                     Pos: 312
          Slave_Running: Yes
        Replicate_do_db: freetime
    Replicate_ignore_db:
             Last_errno: 0
             Last_error:
           Skip_counter: 0
1 row in set (0.00 sec)
```

Репликация пока еще является относительно новым механизмом в MySQL, хотя и хорошо работающим. По крайней мере, функции репликации легко конфигурировать. В MySQL версии 4.0 будут внесены многочисленные улучшения, в том числе направленные на устранение ограничений репликации. В настоящее время репликация лучше всего реализуется в виде синхронизированных резервных копий. Все подчиненные серверы постоянно поддерживают свои данные в синхронизированном состоянии, поэтому при сбое главного сервера нужно лишь переключить приложения на новый сервер.

Если репликация применяется для улучшения производительности работы в организации, следуйте схеме, представленной на рис. 29.3. Приложение должно взять на себя заботу о том, чтобы изменения д елались лишь на главном сервере. Во всем остальном репликация должна быть абсолютно прозрачной. Если серверы, на которых хранится распределенная база данных, находятся на большом расстоянии друг от друга, попытайтесь внедрить схему круговой репликации. Например, можно закрепить за каждым сервером свой набор таблиц и позволить обновлять их только на одном сервере.

Каждый узел в схеме круговой репликации важен для нормальной работы системы. Чтобы свести к минимуму потери от сбоев, создайте дополнительные подчиненные серверы, не входящие в "цепочку". Они предназначены для экстренной замены вышедшего из строя главного сервера.

Запуск нескольких серверов

Обычно сценарий `safe_mysqld` запускает сервер MySQL от имени специального пользователя. Но на самом деле любой пользователь системы может запустить свой сервер параллельно с другими серверами, при условии, что все онибудут работать с разными портами или сокетами. Таким способом часто обеспечивается повышенный уровень безопасности.

К примеру, провайдеры Internet продают дисковое пространство Web-серверов множеству пользователей. В такой открытой среде нельзя рассчитывать на то, что пользователи не будут злоупотреблять своими привилегиями, имея доступ к общим ресурсам. Вместо того чтобы предоставлять каждому пользователю отдельную базу данных и заниматься администрированием всего множества баз данных, провайдеры создают для каждого пользователя свой сервер. Это позволяет пользователям самостоятельно заниматься администрированием, не мешая остальным.

Неудобством такого подхода является то, что приложения должны подключаться к серверам, используя нестандартные установки. Все их нужно свести в персональный конфигурационный файл `~/my.cnf`.

Нестандартные значения номера порта или имени сокета, с которыми работает сервер, тоже нужно вынести в отдельный конфигурационный файл, который передается демону `mysqld` при запуске. Каждому серверу необходим и свой каталог данных. Все это можно сделать вручную, но лучше воспользоваться специально предназначенным для этих целей сценарием `mysqld_multi`.

Данный сценарий работает с одним конфигурационным файлом, в котором у каждого сервера есть своя группа опций, объединенных под общим заголовком. В группу `[mysqld_multi]` входят опции самого сценария, а названия остальных групп состоят из имени сервера и его номера, например `[mysqld13]`. Опции каждой группы передаются соответствующему демону mysqld при его запуске.

Сценарий mysqld_multi может запускать и останавливать любой сервер, указанный по номеру, но для этого сценарий должен иметь привилегию SHUTDOWN. Ею не должен владеть кто угодно, поэтому нужно создать специальную учетную запись на каждом сервере, где подобные действия разрешены (листинг 29.10). Имя пользователя и пароль должны быть везде одинаковыми. Поместите их в группу [mysqld_multi], как показано в листинге 29.11.

```
GRANT SHUTDOWN ON *.*
    TOmulti_admin@localhost
    IDENTIFIED BY 'password'
```

У каждого сервера должна быть своя группа опций. В названии группы нужно указать положительное целое число, уникальное в пределах файла. Группы не обязаны располагаться по порядку. Для каждого сервера нужно задать файл сокета, номер порта и каталог данных. Что касается имени пользователя, то разрешается, чтобы один и тот же пользователь запускал несколько серверов. В листинге 29.11 создаются три сервера для трех пользователей.

```
[mysqld_multi]
mysqld     = /usr/local/libexec/mysqld
mysqladmin = /usr/local/bin/mysqladmin
user       = multi_admin
password   = password
log        = /usr/local/var/multi.log

[mysqld1]
socket     = /tmp/mysql.sock
port       = 3306
pid-file   = /usr/local/mysql/var/mysqld1/myserver.pid
datadir    = /usr/local/mysql/var/mysqld1
user       = jgalt

[mysqld2]
socket     = /tmp/mysqld2.sock
port       = 3307
pid-file   = /usr/local/mysql/var/mysqld2/myserver.pid
datadir    = /usr/local/mysql/var/mysqld2
user       = dtaggart

[mysqld3]
socket     = /tmp/mysqld3.sock
port       = 3308
pid-file   = /usr/local/mysql/var/mysqld3/myserver.pid
datadir    = /usr/local/mysql/var/mysqld3
user       = hreardon
```

РАБОТА С ОБЪЕКТАМИ

В этой главе…

Объектно-ориентированная модель
Сериализация объектов
Объектно-реляционные связи

Глава 30

В этой главе рассказывается о том, как использовать **объекты** в базах данных MySQL. **Объектно-ориентированная** модель пытается отражать сущности реального мира с помощью структур данных. Не существует простого способа установить соответствие между объектами и таблицами реляционных баз данных. Решением этой проблемы занимаются уже не один год. В настоящее время наиболее популярной методикой является так называемый *уровень постоянства* (persistence layer). Это особый набор функций, обеспечивающих преобразование данных при их передаче между приложением и базой данных. Большинство реализаций методики основано на работах Скотта **Амблера** (Scott Ambler). Он публикует многочисленные статьи и спецификации на своем Web-узле ***www.ambysoft.com.***

Компания **SourceForge** *(www.sourceforge.net)* ведет несколько открытых проектов, посвященных реализации уровней постоянства. Наиболее популярный среди них — проект **Osage** *(http://osage.sourceforge.net).* Программистов, работающих на языке Perl, заинтересует проект **Tangram** *(www.soundobjectlogic.com/tangram).* Компания Sun разработала интерфейс **JDO** (Java Data **Objects** – объекты данных Java), обеспечивающий прозрачный доступ к объектам базы данных.

В этой главе будут описаны лишь основные концепции объектно-ориентированной модели и рассмотрены решения, имеющиеся в программе MySQL. Приводимые примеры сценариев написаны на **PHP.**

Объектно-ориентированная модель

В **объектно-ориентированной** модели структуры данных называются объектами. Они имеют свойства и методы. *Свойства*— это отдельные атрибуты объекта. У каждого свойства есть имя и значение в строго заданном диапазоне. Например, у объекта может быть свойство "Age" (возраст), содержащее неотрицательное целое число. Набор значений всех свойств определяет состояние объекта. *Методы*— это **действия,** выполняемые объектом. Как и функция, метод принимает список аргументов и возвращает значение.

Методы и свойства могут быть открытыми либо закрытыми. Открытые методы разрешается вызывать кому угодно, тогда как вызов закрытого метода может осуществить лишь сам объект. То же самое относится и к свойствам.

Объект представляет собой уникальный экземпляр класса. Класс содержит определения методов и свойств и является своего рода шаблоном объектов. Таким образом, термины "объект" и "класс" представляют собой разные понятия.

С помощью механизма наследования можно создавать иерархии классов. Класс, входящий в иерархию, наследует часть свойств и методов своих родительских классов, дополняя их собственными уникальными атрибутами. При одиночном наследовании у класса есть только один непосредственный предок, а при множественном наследовании — несколько предков.

Объекты называются полиморфными, если их функционирование зависит от контекста. Пример полиморфизма — разная реакция на вызов одного и того же метода. Представим, к примеру, что существует метод `calculateSpeed()`, предназначенный для вычисления скорости объекта. Неизвестно заранее, к какому именно объекту в иерархии классов будет применен данный метод. Если это объект класса `Car` (автомобиль), скорость вычисляется в километрах в час. Если же это объект класса Computer (компьютер), скорость вычисляется в мегагерцах.

Здесь упомянута важнейшая особенность полиморфных объектов: они способны принимать "облик" любого из родительских классов. Рассмотрим рис. 30.1. В этой иерархии существует класс Sedan (седан), порождающийся от класса `Car`, который, в свою очередь, наследует класс Vehicle (транспортное средство). Поскольку класс Sedan тоже является потомком класса `Vehicle`, объект класса Sedan можно использовать везде, где допустим класс Vehicle. Например, у транспортного средства есть определенное число колес. Предположим, в классе Vehicle создан метод `getWheels()`, возвращающий число колес объекта. Для автомобиля метод вернет значение 4, а для велосипеда— 2. Оба класса, `Car` и Bicycle, наследуют метод `getWheels()` от родительского класса Vehicle.

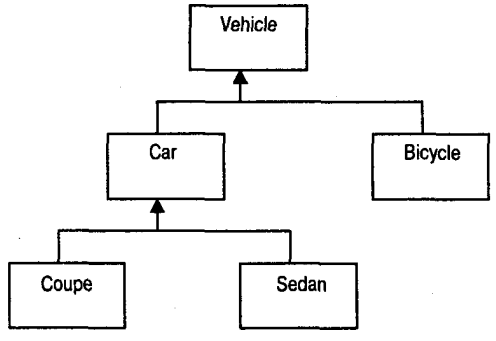


Рис. 30.1. Иерархия наследования

Сериализация объектов

Концепция **сериализации** существует во многих языках программирования. Функции **сериализации** позволяют преобразовать объект в упорядоченную последовательность символов. Эту последовательность можно посылать по сети и записывать на диск.

Для хранения **сериализованных** объектов удобно применять столбцы типа TEXT. В листинге 30.1 приведена инструкция CREATE TABLE, создающая таблицу таких объектов.

```
CREATE TABLE object (
    ID INT NOT NULL AUTO_INCREMENT,
    Data TEXT,
    PRIMARY KEY(ID)
)
```

В листинге 30.2 демонстрируется Сериализация объекта на языке **PHP.** В сценарии определяются два класса: Building (здание) и Room (комната). В объект home класса Building добавляются два объекта класса Room, после чего объект home сохраняется в таблице object. Функция serialize() преобразует содержимое объекта в длинную строку, которая записывается в столбец Data.

```
<?php

    // класс зданий
    class Building
    {
        var $name;

        // массив комнат
        var $room;

        function getRoom($id)
        {
            return($this->room[$id]
        }

        function addRoom($room)
        {
            $this->room[] = $room;
        }
    }

    // класс комнат
    class Room
    {
        var $name;

        // конструктор
```

```
        function Room($name="unnamed")
        {
            $this->name = $name;
        }

        // метод для определения названия комнаты
        function getName()
        {
            return($this->name);
        }
    }

    // создание объекта здания
    $home = new Building;

    // добавление комнат
    $home->addRoom(new Room("kitchen"));
    $home->addRoom(new Room("bedroom"));

    // установление соединения
    if(!($dbLink = mysql_pconnect("localhost", "httpd", "")))
    {
        print("Could not connect to server!<br>\n");
        print(mysql_errno() . ": " . mysql_error() . "<br>\n");
        exit();
    }

    // выбор базы данных 'test'
    if(!mysql_select_db("test", $dbLink))
    {
        print("Could not select database!<br>\n");
        print(mysql_errno() . ": " . mysql_error() . "<br>\n");
        exit ();
    }

    // выполнение запроса
    $Query = "INSERT INTO object (Data) " .
             "VALUES ('" . serialize($home) . "')";
    if(!($dbResult = mysql_query($Query, $dbLink)))
    {
        print("Could not execute query!<br>\n");
        print(mysql_errno() . ": " . mysql_error() . "<br>\n");
        exit();
    }

    // определение идентификатора объекта
    $oid = mysql_insert_id();

    // извлечение объекта из базы данных
    $Query = "SELECT Data FROM object WHERE ID=$oid";
    if(!($dbResult = mysql_query($Query, $dbLink)))
    {
        print("Could not execute query!<br>\n");
        print(mysql_errno() . ": " . mysql_error() . "<br>\n");
        exit();
    }
```

```
    $row = mysql_fetch_row($dbResult);

    $h = unserialize($row[0]);

    // добавление еще одной комнаты
    $h->addRoom(new Room("bathroom"));

    // определение названия второй комнаты
    $r = $h->getRoom(1);
    print($r->getName());
?>
```

После того как объект был сохранен в базе данных, он немедленно извлекается и в него добавляются сведения еще об одной комнате. Последние несколько строк сценария демонстрируют, что методы объекта остались неизменными. Если запустить сценарий, то можно убедиться, что функция `print()` отображает строку "bedroom". Интерпретатор PHP не кодирует методы в сериализованном объекте. Кодируются лишь свойства, а также имя класса. После восстановления объекта интерпретатор проверяет имя класса и делает доступными его методы.

Сериализация — это простое решение, но оно полностью противоречит реляционной модели. Объекты, хранящиеся в базе данных, не могут иметь никаких отношений с записями таблиц. Таким образом, сериализация в MySQL представляет собой всего лишь надежный способ сетевого хранения данных.

Объектно-реляционные связи

Выявление связей между свойствами объектов и столбцами таблиц помогает сохранить преимущества реляционной модели при работе с объектами. Каждому свойству объекта должны быть поставлены в соответствие один или несколько столбцов таблицы. Отдельному классу может соответствовать одна или несколько таблиц, в зависимости от контекста. Реляционная модель требует, чтобы отношение "многие ко многим" формировалось посредством промежуточной таблицы. В объектно-ориентированной модели в этом нет необходимости, хотя вполне допускается инкапсулировать такую таблицу в объекте. В то же время лучше создать специальный метод класса, который скроет промежуточную таблицу и будет играть ту же самую роль вместо нее.

Методы классов не имеют эквивалентов в базеданных. Они остаются элементами программного кода, что создает тесную связь между информацией, хранимой в базе данных, и классами приложения. Это не проблема, если все приложения придерживаются единого интерфейса доступа к данным. Проблема возникает при непосредственном обновлении данных с помощью SQL или другой программной среды. Например, объект может ограничивать значения числового свойства диапазоном от 0,0 до 100,0, но в MySQL столбцы типа FLOAT имеют гораздо более широкий диапазон. Если забыть о существующем ограничении, то, выполняя инструкцию UPDATE, можно легко нарушить целостность столбца. Необходимость учитывать подобного рода ограничения приводит к существенному усложнению работы с объектами.

В общем случае требуется, чтобы для каждого постоянного объекта существовали методы чтения, записи и удаления. Если набор объектов невелик, создать необходи-

мые методы вручную несложно. Метод первого типа извлекает из таблицы запись и сохраняет значения ее полей в свойствах объекта. Метод второго типа помещает значения свойств в столбцы таблицы, создавая новую запись в случае необходимости. Метод третьего типа удаляет объект из базы данных.

Основная проблема в этой простой методике— огромный объем SQL-кода, встраиваемого в каждый объект. Возникает слишком тесная связь между приложением и базой данных. Когда в класс добавляется новое свойство, приходится менять схему базы данных. Решением проблемы является уже упоминавшийся уровень постоянства, инкапсулирующий код взаимодействия с базой данных. Функции этого уровня обеспечивают прозрачную работу с объектами, генерируя все необходимые инструкции SQL.

Создание уровня **постоянства** – это весьма сложная задача, выходящая за рамки данной книги. Ею стоит заниматься, когда предполагается работать с более чем десятью объектами. В приводимом ниже примере используются простые методы доступа к данным.

На рис. 30.2 приведена **UML-диаграмма** двух классов: Building и Room. Связь "один ко многим" отражает тот факт, что в каждом здании есть какое-то число комнат. В простейшем случае одному классу соответствует одна таблица. Такая таблица может содержать столбцы, не сопоставленные свойствам класса. С их помощью реализуются отношения между таблицами.

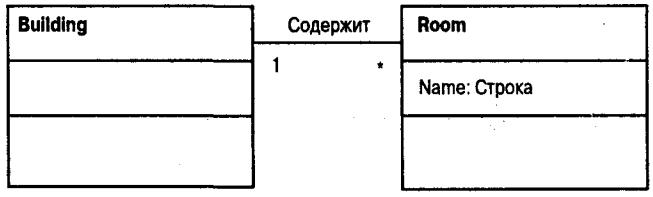


Рис. *30.2.* Диаграмма классов

На рис. 30.3 изображена диаграмма отношений между таблицами классов. Руководствуясь принципами, изложенными в главе 7, "Проектирование баз данных", несложно догадаться, что таблицы Building и Room должны содержать целочисленные первичные ключи, а таблица Room— еще и внешний ключ. Эти идентификаторы не важны с точки зрения использования объектов, но они необходимы для связи таблиц базы данных. Они также позволят приложению ассоциировать друг с другом объекты и записи таблиц. В листинге 30.3 показаны инструкции, требуемые для создания упомянутых таблиц.

Рис. *30.3.* Диаграмма базы данных

```
CREATE TABLE Building (
   ID INT NOT NULL AUTO_INCREMENT,
   Name CHAR(32) NOT NULL,
   PRIMARY KEY(ID)
);

CREATE TABLE Room (
   ID INT NOT NULL AUTO_INCREMENT,
   Building INT NOT NULL,
   Name CHAR(32) NOT NULL,
   PRIMARY KEY(ID),
   FOREIGN KEY (Building) REFERENCES Building(ID)
);
```

В листинге 30.4 демонстрируется реализация классов Building и Room на PHP с использованием методов, обеспечивающих постоянство объектов. Чтобы пример стал немного короче, я не включил в него код проверки ошибок, возникающих при подключении к базе данных. Естественно, в реальном приложении такой код необходим.

```
<?php

    class Building
    {
        var $name;
        var $room;
        var $id;
        var $dbLink;

        function Building($dbLink, $id=0)
        {
            $this->dbLink = $dbLink;
            $this->id = $id;
            $this->room = array();

            // если задан идентификатор, загружаем объект
            if($this->id)
            {
                $this->sqlGet();
            }
        }

        function sqlGet()
        {
            if($this->id <= 0)
            {
                return(FALSE);
            }

            // извлечение данных из таблицы Building
```

```
        $Query = "SELECT Name  " .
                 "FROM Building " .
                 "WHERE ID = $this->id";
        $dbResult = mysql_query($Query, $this->dbLink);
        $row = mysql_fetch_row($dbResult);
        $this->name = $row[0];

        // создание массива комнат
        $this->room = array();
        $Query = "SELECT ID " .
                 "FROM Room " .
                 "WHERE Building = $this->id";
        $dbResult = mysql_query($Query, $this->dbLink);
        while($row = mysql_fetch_row($dbResult))
        {
            $this->room[] = new Room($this->dbLink, $row[0]);
        }
    }

    function sqlPut()
    {
        global $dbLink;
        if($this->id <= 0)
        {
            // создание новой записи в таблице Building
            $Query = "INSERT INTO Building (Name) " .
                     "VALUES ('" . $this->name . "')";
            $dbResult = mysql_query($Query, $this->dbLink);
            $this->id = mysql_insert_id();

            // занесение в базу данных массива комнат
            for($r=0; $r < count($this->room); $r++)
            {
                $this->room[$r]->building = $this->id;
                $this->room[$r]->sqlPut();
            }
        }
        else
        {
            // обновление существующей записи
            $Query = "UPDATE Building " .
                     "SET Name = '" . $this->name . "'" .
                     "WHERE ID = $this->id";
            $dbResult = mysql_query($Query, $this->dbLink);

            // обновление массива комнат
            for($r=0; $r < count($this->room); $r++)
            {
                $this->room[$r]->sqlPut();
            }
        }
    }

    function sqlDelete()
    {
        // удаление всех объектов комнат
```

```
        for($r=0; $r < count($this->room); $r++)
        {
            $this->room[$r]->sqlDelete();
        }

        // удаление объекта здания
        $Query = "DELETE FROM Building " .
                 "WHERE ID = $this->id";
        $dbResult = mysql_query($Query, $this->dbLink);
        $this->id = 0;
    }
}

class Room
{
    var $building;
    var $name; ,
    var $id;
    var $dbLink;

    function Room($dbLink, $id=0)
    {
        $this->dbLink = $dbLink;
        $this->id = $id;

        // если задан идентификатор, загружаем объект
        if($id)
        {
            $this->sqlGet();
        }
    }

    function sqlGet()
    {
        if($this->id <= 0)
        {
            return(FALSE);
        }

        // извлечение данных из таблицы Room
        $Query = "SELECT Building, Name " .
                 "FROM Room " .
                 "WHERE ID = $this->id";
        $dbResult = mysql_query($Query, $this->dbLink) ;
        $row = mysql_fetch_row($dbResult) ;
        $this->building = $row[0];
        $this->name = $row[1];
    }

    function sqlPut()
    {
        if($this->building <= 0)
        {
            return(FALSE);
        }
        if ($this->id <= 0)
```

```
        {
            // создание новой записи в таблице Room
            $Query = "INSERT INTO Room (Building, Name) " .
                     "VALUES ($this->building, " .
                     "'" . $this->name . "')";
            $dbResult = mysql_query($Query, $this->dbLink);
            $this->id = mysql_insert_id();
        }
        else
        {
            // обновление существующей записи
            $Query = "UPDATE Room " .
                     "SET Building = $this->building, " .
                     "Name = '" . $this->name . "'" .
                     "WHERE ID = $this->id";
            $dbResult = mysql_query($Query, $this->dbLink);
        }
    }

    function sqlDelete()
    {
        // удаление объекта комнаты
        $Query = "DELETE FROM Room " .
                 "WHERE ID = $this->id";
        $dbResult = mysql_query($Query, $this->dbLink);
        $this->id = 0;
        $this->building = 0;
    }
}

// установление соединения
$dbLink = mysql_pconnect("localhost", "httpd", "");

// выбор базы данных 'test'
mysql_select_db("test", $dbLink);

// создание объекта здания и сохранение его в базе данных
$b = new Building($dbLink);
$b->name = "Home";

$r = new Room($dbLink);
$r->name = "Bedroom";
$b->room[] = $r;
$r->name = "Kitchen";
$b->room[] = $r;
$r->name = "Bathroom";
$b->room[] = $r;

$b->sqlPut();

// извлечение объекта здания
$home = new Building($dbLink, $b->id);

// удаление его из базы данных
$home->sqlDelete();
```

```
    // повторное занесение объекта в базу данных
    $home->sqlPut();

    // повторное удаление объекта
    $home->sqlDelete();
?>
```

Класс Building содержит четыре свойства и четыре метода. В свойстве name записано имя здания, а свойство room представляет собой массив объектов класса Room. Свойство id — это уникальный идентификатор объекта, соответствующий значению первичного ключа в таблице Building. Свойство dbLink хранит идентификатор сеанса, необходимый функциям PHP, которые работают с базами данных MySQL.

Первый метод класса Building — это конструктор, вызываемый при создании конкретного экземпляра класса. Данному методу передается идентификатор сеанса и необязательный идентификатор объекта. Если второй аргумент задан, конструктор загружает указанный объект из базы данных.

Метод sqlGet() запрашивает из базы данных название здания, соответствующее указанному идентификатору. Если бы у класса Building были дополнительные свойства, пришлось бы запрашивать и дополнительные столбцы. В таблице Room есть внешний ключ, связывающий ее с таблицей Building, поэтому метод sqlGet() по имеющемуся идентификатору здания извлекает список идентификаторов комнат. Для каждого найденного идентификатора создается объект класса Room. Выполнять отдельный запрос с целью извлечения каждого объекта этого класса — не самое эффективное решение, но в данном случае оно обеспечивает слабую связность. Метод sqlGet() класса Room лишь извлекает записи из базы данных и помещает значения их полей в соответствующие свойства объекта, но в более сложном классе может потребоваться дополнительная обработка данных. Например, объект может отдельно хранить номер года, месяца и дня, а в базе данных всем этим свойствам соответствует один столбец типа DATE.

Метод sqlPut() сохраняет текущее состояние объекта в записях базы данных. На основании значения свойства id делается выбор в пользу инструкции UPDATE либо INSERT. Если идентификатор равен нулю, предполагается, что нужно создать новую запись в таблице Building, иначе обновляется существующая запись. Функция mysql_insert_id() возвращает идентификатор, сгенерированный для поля-счетчика, и этот идентификатор сохраняется в свойстве id.

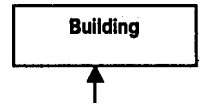
У объектов класса Room есть свой метод sqlPut(), по очереди вызываемый для каждой комнаты здания. PHP-программистам советую обратить внимание на использование цикла for. Цикл foreach в данном случае не подходит, поскольку он будет работать с копиями элементов массива room, а не с самими элементами. Вызов метода sqlPut() класса Room приведет к созданию записи, но обновление свойства id потеряется, если изменению подвергнется копия.

Метод sqlDelete() класса Building сначала вызывает одноименный метод каждого объекта массива room, после чего выполняет инструкцию DELETE для удаления записи из таблицы Building. Свойство id объекта устанавливается равным нулю, что указывает на независимость объекта от какой бы то ни было записи базы данных.

Код класса Room написан по тому же образцу, что и класс Building. После определений классов идет основное тело сценария. Обратите внимание на способ добав-

ления объектов комнат к объекту здания. В сценарии используется один объект класса Room, копии которого с разными названиями комнат записываются в массив room. В результате не приходится создавать три разных объекта и устанавливать свойство dbLink каждого из них.

Классы Building и Room представляют собой пример отношения включения, когда один класс является членом другого. Другой тип отношения между объектами — это наследование. Ему стоит уделить особое внимание. Рассмотрим диаграмму, представленную на рис. 30.4. На ней изображены два потомка класса Building: классы Store (магазин) и Residence (жилой дом). Код для их реализации приведен в листинге 30.5.



S1 R

Рис. 30.4. Класс Building и его подклассы

```
class Building
{
    var $name;
    var $room;
}

class Residence extends Building
{
    var $occupants;
}

class
{
    var $parkingSpaces;
}
```

Преобразовать эти три класса в таблицы можно двумя способами. Первый из них заключается в создании одной таблицы со всеми необходимыми столбцами, а второй — в создании отдельной таблицы для каждого класса, включая родительский.

В листинге 30.6 показана инструкция, которая создает таблицу, охватывающую все свойства родительского класса и двух его подклассов. Если в конкретном экземпляре класса то или иное свойство отсутствует, в соответствующую ячейку записывается значение NULL. Например, у объекта класса нет свойства, определяющего число жильцов, поэтому столбец occupants для него будет равен NULL. Чтобы можно было различать между собой объекты классов Residence и S таблицу добавлен дополнительный столбец buildingType, содержащий всего два возможных значения.

```
CREATE TABLE Building (
    ID INT NOT NULL AUTO_INCREMENT,
    name CHAR(32) NOT NULL,
    buildingType ENUM('Residence', 'Store') NOT NULL,
    occupants INT,
    parkingSpaces INT,
    PRIMARY KEY(ID)
)
```

Недостатком такого подхода является сильная связность. Если возникнет необходимость поменять определение одного из подклассов, придется соответствующим образом изменить и саму таблицу, а это затронет все ее записи. Кроме того, в одной таблице смешиваются пространства имен нескольких классов, что зачастую приводит к чрезмерному удлинению имен столбцов. Например, если у обоих подклассов есть свойство name, потребуется создать два разных столбца, в имена которых в качестве префикса входит имя подкласса.

Рассмотренный подход плохо работает также в случае множественного наследования. В листинге 30.6 можно вместо типа ENUM использовать тип SET, но это не компенсирует тот факт, что в таблице будут присутствовать столбцы для всех возможных классов. Таблицы, в которых слишком много значений NULL, свидетельствуют о том, что разработчики базы данных не придерживались требований реляционной модели.

Преимуществом однотабличного подхода является простота реализации. Нет необходимости выполнять объединения таблиц, тем самым ускоряется доступ к данным. Конечно, придется просматривать большее число записей, но наличие индексов компенсирует этот недостаток.

Второй подход, как уже говорилось, заключается в создании отдельной таблицы для каждого класса, в том числе родительского. Связь подклассов с родительским классом будет реализовываться посредством внешних ключей. Чтобы собрать всю информацию об объекте, придется выполнить объединение как минимум двух таблиц. В листинге 30.7 показаны инструкции, создающие таблицы для нашего примера.

```
CREATE TABLE Building (
    ID INT NOT NULL AUTO_INCREMENT,
    name CHAR(32) NOT NULL,
    PRIMARY KEY(ID)
)

CREATE TABLE Residence (
    ID INT NOT NULL AUTO_INCREMENT,
    Building INT NOT NULL,
    occupants INT,
    PRIMARY KEY(ID),
    FOREIGN KEY (Building) REFERENCES Building (ID)
)
```

```
CREATE TABLE Store (
    ID INT NOT NULL AUTO_INCREMENT,
    Building INT NOT NULL,
    parkingSpaces INT,
    PRIMARY KEY(ID),
    FOREIGN KEY (Building) REFERENCES Building (ID)
)
```

Данная методика решает проблему сильной связности. Добавление нового подкласса сводится к добавлению новой таблицы. Легко реализуется и множественное наследование. Например, если здание одновременно является и магазином, и жилым домом, то в таблицах Residence и Store просто будут записи с одинаковыми внешними ключами.

Реализация такого рода базы данных требует определенных усилий и приводит к созданию большого количества таблиц. Следовательно, возрастает число операций объединения, что сказывается на скорости работы с базой данных. Тем не менее описанная методика обеспечивает слабую связность и высокую степень полиморфизма.

РАСШИРЕНИЕ ВОЗМОЖНОСТЕЙ MYSQL

В этой главе..

- Библиотека функций отладки
- Создание наборов символов
- Создание функций
- Создание процедур

Глава 31

В этой главе рассматриваются способы расширения функциональных возможностей сервера MySQL. Сюда входит написание собственных функций и процедур, а также создание наборов символов. Описываются также средства отладки, имеющиеся в MySQL.

Библиотека функций отладки

В MySQL входит библиотека функций отладки, первоначально созданная Фредом Фишем (Fred Fish). Чтобы разрешить ее использование, нужно на этапе компиляции программы вызвать сценарий `configure` с опцией `--with-debug`. Если в распоряжении имеется бинарный дистрибутив, проверьте версию какого-либо исполняемого файла. Программы, скомпилированные с поддержкой отладки, имеют суффикс `--debug`. Библиотека функций отладки является частью библиотеки `mysqlclient`, о которой рассказывалось в главе 16, "Использование библиотеки языка C". Макросы библиотеки объявлены в файле `dbug.h`.

Названия всех макросов начинаются с префикса DBUG_ (табл. 31.1). Если нужно отключить отладку, определите макроконстанту DBUG_OFF. При ее наличии все остальные макросы игнорируются.

Макрос	*Описание*
`DBUG_ENTER(`*функция*`)`	Этот макрос принимает имя функции, в которую входит программа. Его нужно указывать после объявления локальных переменных, но перед вызовом каких-либо инструкций. `DBUG ENTER("main");`

Макрос	*Описание*
DBUG_EXECUTE(*ключевое_слово*, *инструкция*)	Этот **макрос** помечает **указанную инструкцию** меткой. `DBUG_EXECUTE("where", print_where(tmp,"cache")););`
DBUG_FILE	**Эта** макроконстанта **инкапсулирует** дескриптор выходного **файла, в** который **записывается** отладочная информация. `fprintf(DBUG_FILE, "\nWHERE:(%s) ",info);`
DBUG_LONGJMP(*среда*, *значение*)	**Если в** программе используется **функция** `longjmp()`, **замените ее данным** макросом. `DBUG_LONGJMP(env, val);`
DBUG_POP()	Этот макрос восстанавливает **предыдущее** состояние **отладки.** `DBUG_POP();`
DBUG_PRINT(*ключевое_слово*, (*формат*[, *аргументы*]))	Этот макрос **записывает** отладочную **информацию в** файловый поток, **как если бы была вызвана функция** `fptintf()` **с** константой DBUG_FILE **в качестве** дескриптора. **Первый аргумент –** это ключевое **слово**, которое можно использовать **с** описанным **ниже** флагом d. Второй **аргумент –** это набор параметров, **передаваемый функции** `fprintf()`. `DBUG_PRINT("mfunkt", ("name: '%s'",name));`
DBUG_PROCESS(*имя*)	Этот макрос **задает имя** текущего процесса. `DBUG_PROCESS(argv[0]);`
DBUG_PUSH(*формат*)	Этот макрос **задает** новые **параметры для** текущего **сеанса** отладки. **Все** они помещаются **в стек,** поэтому можно восстанавливать **предыдущие** состояния **с** помощью макроса DBUG_POP().
	`DBUG_PUSH("d:t");`

Макрос	*Описание*
DBUG_RETURN(*значение*)	Этот макрос заменяет инструкцию return. Если функция ничего не возвращает, пользуйтесь макросом DBUG_VOID_RETURN(). DBUG_RETURN(TRUE);
DBUG_SETJMP(*среда*)	Этот макрос заменяет функцию setjmp(). DBUG_SETJMP(env);
DBUG_VOID_RETURN	Этот макрос указывает на то, что функция не возвращает никаких значений

Отладка функции начинается с того, что в ее начало помещается макрос DBUG_ENTER(). Затем все вызовы инструкции return заменяются либо макросом DBUG_RETURN(), либо DBUG_VOID_RETURN(). Это позволяет отладчику определять, когда управление передается той или иной функции.

Макрос DBUG_EXECUTE() помечает отдельную строку кода ключевым словом. Макрос DBUG_PRINT() записывает сообщение в отладочный файл. Можно напрямую работать с этим файлом благодаря макроконстанте DBUG_FILE, в которой хранится его дескриптор.

В листинге 31.1 показан пример отладки функции, которая вычисляет факториал заданного целого числа.

```
ttinclude <dbug.h>

int factorial(register int value)
{
   DBUG_ENTER("factorial");
   DBUG_PRINT("find", ("find %d factorial", value));
   if (value > 1)
   {
      value *= factorial(value - 1);
   }
   DBUG_PRINT("result", ("result is %d", value));
   DBUG_RETURN(value);
}
```

Программа, работающая с библиотекой функций отладки, обычно начинает отладку, вызывая макрос DBUG_PUSH(). Утилиты MySQL включают отладку, если получен соответствующий аргумент командной строки или если установлена специальная переменная среды. Об опциях --debug и -I рассказывалось в главе 14, "Утилиты командной строки".

В табл. 31.2 перечислены **флаги**, понимаемые отладчиком и передаваемые макросу `DBUG_PUSH()`. Они определяют, какая информация должна быть представлена в выходных данных. Строка формата выглядит как последовательность флагов, разделенных двоеточиями. Некоторые флаги требуют наличия параметров. Например, флаг `d` принимает список ключевых слов, разделенных запятыми.

Флаг	*Описание*
`d[,`*`ключевые_слова`*`]`	Этот флаг разрешает выводить информацию макросам с именами вида `DBUG_`*`ключевое_слово`*. Если список **ключевых** слов не указан, то подразумеваются все макросы. Ключевые слова должны задаваться без префикса `DBUG_`
`D[,`время]	Этот флаг свидетельствует о том, что вывод отладочной информации должен быть задержан на указанное число десятых долей секунды. Например, флаг `D,25` означает, что при выводе каждой строки будет выдерживаться пауза длительностью 2,5 секунды
`f[,`*функции*]	Этот флаг разрешает выводить отладочную информацию только из указанных функций. Например, флаг `f,main` означает, что будут включены макросы, находящиеся в теле функции main `()`
`F`	Этот флаг указывает на то, что каждую строку отладочной информации необходимо сопровождать именем **исходного** файла
`g`	Этот флаг включает режим профилирования. В результате будет создан файл `dbugmon.out`. В качестве аргумента может быть указан список функций, для которых выполняется профилирование. В противном случае подразумеваются все функции. Более подробную информацию об этом можно найти в файле `dbug/readme.prof` дистрибутива MySQL
`i`	Этот флаг указывает на то, что каждую строку отладочной информации необходимо сопровождать **идентификатором** процесса или потока, в зависимости от контекста
`L`	Этот флаг указывает на то, что каждую строку отладочной информации необходимо сопровождать номером строки исходного файла
`n`	Этот флаг указывает на то, что каждую строку отладочной информации необходимо сопровождать информацией о глубине вызова текущей функции

Флаг	*Описание*
N	Этот **файллаг** включает нумерацию строк в файле отладки
o[,*файл*]	Этот флаг говорит о том, что отладочная информация должна направляться в указанный файл. По умолчанию эта информация отображается на экране, но многие клиентские программы изменяют данную**установку,** создавая в каталоге `/tmp` файл с именем программы и расширением `.trace`
O[,*файл*]	Этот флаг аналогичен флагу `o`, но после каждой записи в файл будет очищаться файловый буфер
p[,процессы]	Этот флаг разрешает выводить отладочную информацию только указанным процессам. Имя процесса должно быть задано с помощью макроса DBUG_PROCESS
P	Этот флаг указывает на то, что каждую **строку** отладочной информации необходимо сопровождать именем процесса
r	Этот флаг заставляет выравнивать выводимую информацию по левому краю экрана после вызова макроса `DBUG_PUSH()`
S	При наличии этого флага отладчик будет вызывать функцию `_sanity(_file_, _line_)` для каждой отлаживаемой функции, пока первая невернет значение, отличное от нуля
t[,*уровень*]	Этот макрос включает вывод строк, помечающих точки вызова и завершения функций. Через запятую может быть указан максимальный уровень трассировки, по достижении которого отладочная и трассировочная информация перестает выводиться

Исходная документация, написанная Фредом Фишем, находится в файле `dbug/user.r` в исходном каталоге MySQL. С помощью утилиты `nroff` этот файл можно преобразовать в формат Postscript или в текстовый формат.

Создание наборов символов

Допускается включать в программу MySQL новые наборы символов. Для простого, однобайтового набора требуется лишь один файл с четырьмя таблицами преобразований. В случае сложного набора необходимо также написать функцию, выполняющую сортировку строк.

```
# Конфигурационный файл для набора символов latin1.

# Массив ctype[]  (должен содержать 257 элементов).
  00
  20  20  20  20  20  20  20  20  20  28  28  28  28  28  20  20
  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20  20
  48  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10
  84  84  84  84  84  84  84  84  84  84  10  10  10  10  10  10
  10  81  81  81  81  81  81  01  01  01  01  01  01  01  01  01
  01  01  01  01  01  01  01  01  01  01  01  10  10  10  10  10
  10  82  82  82  82  82  82  02  02  02  02  02  02  02  02  02
  02  02  02  02  02  02  02  02  02  02  02  10  10  10  10  20
  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00
  48  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10
  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10  10
  01  01  01  01  01  01  01  01  01  01  01  01  01  01  01  01
  01  01  01  01  01  01  01  10  01  01  01  01  01  01  01  02
  02  02  02  02  02  02  02  02  02  02  02  02  02  02  02  02
  02  02  02  02  02  02  02  10  02  02  02  02  02  02  02  02

# Массив to_lower  (должен содержать 256 элементов).
  00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
  10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
  20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
  30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
  40  61  62  63  64  65  66  67  68  69  6A  6B  6C  6D  6E  6F
  70  71  72  73  74  75  76  77  78  79  7A  5B  5C  5D  5E  5F
  60  61  62  63  64  65  66  67  68  69  6A  6B  6C  6D  6E  6F
  70  71  72  73  74  75  76  77  78  79  7A  7B  7C  7D  7E  7F
  80  81  82  83  84  85  86  87  88  89  8A  8B  8C  8D  8E  8F
  90  91  92  93  94  95  96  97  98  99  9A  9B  9C  9D  9E  9F
  AO  A1  A2  A3  A4  A5  A6  A7  A8  A9  AA  AB  AC  AD  AE  AF
  BO  B1  B2  B3  B4  B5  B6  B7  B8  B9  BA  BB  BC  BD  BE  BF
  E0  E1  E2  E3  E4  E5  E6  E7  E8  E9  EA  EB  EC  ED  EE  EF
  F0  F1  F2  F3  F4  F5  F6  D7  F8  F9  FA  FB  FC  FD  FE  DF
  EO  E1  E2  E3  E4  E5  E6  E7  E8  E9  EA  EB  EC  ED  EE  EF
  FO  Fl  F2  F3  F4  F5  F6  F7  F8  F9  FA  FB  FC  FD  FE  FF
```

```
# Массив to_upper (должен содержать 256 элементов).
  00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
  10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
  20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
  30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
  40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
  50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
  60  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
  50  51  52  53  54  55  56  57  58  59  5A  7B  7C  7D  7E  7F
  80  81  82  83  84  85  86  87  88  89  8A  8B  8C  8D  8E  8F
  90  91  92  93  94  95  96  97  98  99  9A  9B  9C  9D  9E  9F
  AO  A1  A2  A3  A4  A5  A6  A7  A8  A9  AA  AB  AC  AD  AE  AF
  BO  B1  B2  B3  B4  B5  B6  B7  B8  B9  BA  BB  BC  BD  BE  BF
  CO  C1  C2  C3  C4  C5  C6  C7  C8  C9  CA  CB  CC  CD  CE  CF
  DO  D1  D2  D3  D4  D5  D6  D7  D8  D9  DA  DB  DC  DD  DE  DF
  CO  C1  C2  C3  C4  C5  C6  C7  C8  C9  CA  CB  CC  CD  CE  CF
  DO  D1  D2  D3  D4  D5  D6  F7  D8  D9  DA  DB  DC  DD  DE  FF

# Массив sort_order (должен содержать 256 элементов).
  00  01  02  03  04  05  06  07  08  09  OA  0B  OC  OD  OE  OF
  10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
  20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
  30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
  40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
  50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
  60  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
  50  51  52  53  54  55  56  57  58  59  5A  7B  7C  7D  7E  7F
  80  81  82  83  84  85  86  87  88  89  8A  8B  8C  8D  8E  8F
  90  91  92  93  94  95  96  97  98  99  9A  9B  9C  9D  9E  9F
  AO  A1  A2  A3  A4  A5  A6  A7  A8  A9  AA  AB  AC  AD  AE  AF
  BO  B1  B2  B3  B4  B5  B6  B7  B8  B9  BA  BB  BC  BD  BE  BF
  41  41  41  41  5C  5B  5C  43  45  45  45  45  49  49  49  49
  44  4E  4F  4F  4F  4F  5D  D7  D8  55  55  55  59  59  DE  DF
  41  41  41  41  5C  5B  5C  43  45  45  45  45  49  49  49  49
  44  4E  4F  4F  4F  4F  5D  F7  D8  55  55  55  59  59  DE  FF
```

Десятичное значение	***Шестнадцатеричное** значение*	*Описание*
1	0x01	Прописная буква
2	0x02	Строчная буква
4	0x04	Цифра
8	0x08	Символ пробела
16	0x10	Знак пунктуации
32	0x20	Управляющий символ
64	0x40	Пусто
128	0x80	**Шестнадцатеричная** цифра

Вторая группа значений представляет собой таблицу ASCII, предназначенную для перевода символов в нижний регистр. Например, символ в позиции 0x41 — это прописная буква **'A'**, но ей соответствует значение 0x61, т.е. строчная **'a'**. Третья группа значений определяет таблицу ASCII для перевода символов в верхний регистр. Последняя таблица задает порядок сортировки и обычно совпадает с третьей таблицей.

Если истинный порядок сортировки невозможно отразить в столь простой таблице, то необходимо написать специальные функции сортировки. Для этого нужно создать файл в каталоге `strings` дерева MySQL. Здесь же находятся файлы всех остальных наборов символов, например ctype-big5`.c`. Программа MySQL ищет в этом файле пять функций и четыре массива. Имена всех функций и массивов включают стандартный префикс и название набора. В листинге 31.3 показаны прототипы функций и определения массивов для набора символов big5.

```
uchar NEAR ctype_big5[257]

uchar NEAR to_lower_big5[]

uchar NEAR to_upper_big5[]

uchar NEAR sort_order_big5[]

my_bool my_like_range_big5(
   const char *ptr,
   uint ptr_length,
   pchar escape,
   uint res_length,
   char *min_str,
   char *max_str,
   uint *min_length,
   uint *max_length)

int my_strcoll_big5(
   const uchar * s1,
   const uchar * s2)

int  my_strnncoll_big5(
   const uchar * s1,
   int len1,
   const uchar * s2,
   int  len2)

int  my_strnxfrm_big5(
   uchar * dest,
   uchar * src,
   int len,
   int  srclen)

int  my_strxfrm_big5(
   uchar * dest,
   uchar * src,
   int len)
```

Четыре функции осуществляют сравнение строк. Функция с префиксом my_like_range_ находит наименьшую и наибольшую строки (с учетом регистра), соответствующие выражению в операторе LIKE. Само выражение передается в аргументе ptr. Аргумент ptr_length определяет длину выражения. Содержимое наименьшей и наибольшей строк заносится в аргументы min_str и max_str соответственно.

Функции с префиксами my_strcoll_ и my_strncoll_ служат аналогами обычных функций strcoll() и strncoll() языка C, которые, в свою очередь, являются версиями функций strcmp() и strncmp(), учитывающими региональные установки. Функции с префиксами my_strxfrm_ и my_strnxfrm_ эмулируют стандартные функции strxfrm() и strnxfrm(). Получить о них более подробную информацию можно в man-файлах UNIX.

В начало файла нужно добавить комментарий, подобный тому, что показан в листинге 31.4. На основании строк, приведенных в комментарии, сценарий configure включает набор символов в клиентскую библиотеку. Переменная strxfrm_multiply_*набор* задается в том случае, когда у набора символов есть свои функции сортировки строк. Она определяет максимальный коэффициент удлинения строки при ее прохождении через функцию с префиксом my_strxfrm_. Как следует из листинга, строки в кодировке big5 не растягиваются.

Если в набор входят многобайтовые символы, потребуется определить переменную mbmaxlen_*набор*. Она задает максимально возможное число байтов в представлении символа. Например, в набор big5 входят двухбайтовые символы.

```
/*
  Эти строки анализируются сценарием configure при создании
  файла ctype.c, поэтому не меняйте их без веских оснований.

  .configure. strxfrm_multiply_big5=1
  .configure. mbmaxlen_big5=2
 /
```

Все, что осталось теперь сделать, — это добавить имя набора в списки CHARSETS_AVAILABLE и COMPILED_CHARSETS в файле configure.in и перекомпилировать программу. Активизировать доступные наборы символов можно с помощью опций командной строки, конфигурационного файла или SQL-инструкций, как рассказывалось в главах 13, "Инструкции SQL", и 14, "Утилиты командной строки".

Создание функций

В программу MySQL можно добавить новые функции, которые будут использоваться точно так же, как и встроенные функции, описанные в главе 12, "Встроенные функции". Существуют два способа создания таких функций. Первый — это включение функции непосредственно в исходный код MySQL, второй — определение функции в формате UDF (User-Definable Function — пользовательская функция). Второй способ подходит, когда функцию требуется хранить и отлаживать отдельно от утилит MySQL. Код функции компилируется в виде библиотечного модуля, который загружа-

ется с помощью инструкции CREATE FUNCTION. Первый способ менее удобен, поскольку приходится останавливать сервер и заменять его исполняемый файл. Так обычно поступают с функциями, которые планируется сделать частью проекта MySQL. Ниже будет рассмотрен второй подход.

В дистрибутив MySQL входит пример UDF-функций. Он находится в файле sql/udf_example.cc. В этом файле содержатся определения шести функций. Я скопировал из него строку, используемую утилитой make для правильного вызова компилятора языка C. В комментариях к файлу рекомендуется выполнить команду make udf_example.cc, чтобы посмотреть параметры компиляции статического объектного файла, а затем заменить аргумент -c аргументом -shared -o udf_example.so. В листинге 31.5 показана строка компиляции файла в моей системе RedHat в каталоге sql программы MySQL.

```
c++ \
    -DMYSQL_SERVER \
    -DDEFAULT_MYSQL_HOME="\"/usr/local\"" \
    -DDATADIR="\"/usr/local/var\"" \
    -DSHAREDIR="\"/usr/local/share/mysql\"" \
    -DHAVE_CONFIG_H \
    -DDBUG_OFF \
    -I../bdb/build_unix \
    -I../innobase/include \
    -I./../include \
    -I./../regex \
    -I. \
    -I../include \
    -I.. \
    -O3 \
    -fno-implicit-templates \
    -shared \
    -o udf_example.so \
    udf_example.cc
```

После компиляции совместно используемой библиотеки нужно скопировать ее в один из каталогов, перечисленных в файле /etc/ld.so.conf. Если каталог будет другим, укажите его имя в переменной среды LD_LIBRARY_PATH.

Для активизации функции нужно выполнить инструкцию CREATE FUNCTION. В листинге 31.6 демонстрируется загрузка функций METAPHON и AVGCOST из библиотеки udf_example. В результате в таблице mysql.func будут созданы две новые записи, и пока инструкция DROP FUNCTION их не удалит, функции останутся доступны всем пользователям даже в случае перезапуска сервера.

```
CREATE FUNCTION METAPHON
    RETURNS STRING SONAME "udf_example.so";
CREATE AGGREGATE FUNCTION AVGCOST
    RETURNS REAL SONAME "udf_example.so";
```

В библиотечном файле может содержаться одна или несколько функций. Язык реализации — C или C++. Каждой **SQL-функции** в этом файле соответствует как минимум одна функция с аналогичным именем. Кроме того, могут быть созданы функции с суффиксами `_init` и `_deinit`. Например, в файле `udf_example.cc` содержатся определения функций `metaphon()`, metaphon_init() и `metaphon_deinit()`. Когда вводится инструкция, содержащая вызов **UDF-функции**, сначала происходит обращение к функции с суффиксом `_init`. Затем для каждой записи выполняется основная функция. В конце вызывается функция с суффиксом `_deinit`. Все три функции должны быть безопасны для потоков. Это означает, что в них нельзя использовать глобальные переменные, меняющие свои значения. Функция с суффиксом `_init` предназначена для динамического выделения памяти, а функция с **суффиксом** `_deinit` освобождает выделенную память.

Основная функция может возвращать значение с плавающей запятой, целое число или строку. В первом случае тип результата должен быть double, во втором — long, а в третьем — char *. В листинге 31.7 показано несколько прототипов функций.

```
char *metaphon(
    UDF_INIT *initid,
    UDF_ARGS *args,
    char *result,
    unsigned long *length,
    char *is_null,
    char *error);

long long sequence(
    UDF_INIT *initid,
    UDF_ARGS *args,
    char *is_null,
    char *error);

double myfunc_double(
    UDF_INIT *initid,
    UDF_ARGS *args,
    char *is_null,
    char *error);

my_bool metaphon_init(
    UDF_INIT *initid,
    UDF_ARGS *args,
    char *message);

void metaphon_deinit(
    UDF_INIT *initid);
```

Числовые значения возвращаются непосредственно, а строковые — через указатели. Программа MySQL резервирует **255-символьный** буфер для аргумента `result`. В аргументе lengthдолжен быть указан размер возвращаемого значения. Если размер превышает 255 байтов, нужно создать собственный буфер в инициализирующей функции и передать указатель на него в поле `ptr` структуры UDF_INIT. Описание полей структуры приведено в табл. 31.4.

Поле	*Описание*
my_bool maybe_null	Указывает на то, может ли функция возвращать пустое значение. Если один из аргументов функции может быть пустым, в это поле будет записана единица
unsigned int decimals	Содержит количество цифр после запятой, если функция возвращает числовое значение. Будет указана максимальная точность среди всех аргументов
unsigned int max_length	Определяет максимальную длину возвращаемой строки
char *ptr	С помощью этого указателя осуществляется обмен данными между инициализирующей функцией и другими двумя функциями. Например, можно выделить блок памяти и записать сюда адрес этого блока

Если функция возвращает пустое значение, аргумент is_null должен быть равен 1. В случае ошибки в аргумент error записывается значение 1. В результате текущая запись и все последующие станут пустыми.

Описание структуры UDF_ARGS приведено в табл. 31.5. Через эту структуру программа MySQL передает аргументы функции.

Поле	*Описание*
unsigned int arg_count	Содержит число аргументов функции. Если это значение фиксировано, проверьте его в инициализирующей функции
enum Item_result *arg_type	Содержит массив типов аргументов. Возможные значения массива таковы: INT_RESULT, REAL_RESULT и STRING_RESULT. Можно осуществлять проверку типов и в случае несовпадения либо возвращать признак ошибки, либо корректировать содержимое массива, приводя аргументы к нужному типу
char **args	Содержит массив значений аргументов. Если аргумент является строкой, в массиве будет храниться указатель на строку. Длины строковых аргументов приведены в массиве lengths. Если аргумент представляет собой целое число или число с плавающей запятой, приведите значение к типу long long или double соответственно

Поле	*Описание*
unsigned long *lengths	Содержит массив длин аргументов. В инициализирующей функции эти значения устанавливаются по максимуму на основании определений столбцов. В основной функции длины строковых аргументов являются точными

Инициализирующая функция возвращает ненулевое значение в случае ошибки. Если обнаружена ошибка, скопируйте текст поясняющего значения в аргумент message. Максимальная длина сообщения хранится в переменной `MYSQL_ERRMSG_SIZE`, но нужно стараться, чтобы длина сообщения не превышала 80 символов. Тогда оно сможет быть отображено на любом стандартном терминале.

Если вы хотите поделиться с другими пользователями своими функциями, зарегистрируйте их на узле *http://empyrean.lib.ndsu.nodak.edu/~nem/mysql/udf*, к примеру, нашел там очень интересную функцию, осуществляющую вызов методов Corba.

Создание процедур

Процедуры MySQL выполняют операции над результатами запросов. Процедура активизируется при наличии в конце инструкции SELECT ключевого слова PROCEDURE. В настоящий момент в MySQL входит единственная процедура `analyse()`, описанная в главе 12, "Встроенные функции". Разрешается создавать собственные процедуры, включая их в программу на этапе компиляции.

Процедуры появились в MySQL версии 3.21, но пока что не вызвали особого энтузиазма. Писать их оказалось слишком сложно для большинства пользователей. Например, для создания процедуры на C++ требуется определить класс, производный от класса Procedure. Код последнего находится в файлах `sql/procedure.h` и `sql/procedure.cc`. Процедура `analyse()` реализована в файле `sql/sql_analyse.cc`.

С другой стороны, можно воспользоваться библиотекой `mylua` *(www.fastflow.it/mylua)*, которая позволяет динамически загружать процедуры, написанные на языке LUA (*www.lua.org*). Для запуска сценария LUA в MySQL нужно вызвать функцию `LUA()`, указав ей имя исходного файла, где содержится определение процедуры.

РЕСУРСЫ В INTERNET

В этом приложении.

- Официальные списки рассылки
- Архивы списков рассылки
- Web-узлы
- Отчеты об ошибках

Приложение A

В этом приложении приведен перечень ресурсов, касающихся MySQL, которые можно найти в Internet.

Официальные списки рассылки

Компания MySQL AB ведет несколько списков рассылки, посвященных MySQL. Большинство из них доступно в виде дайджестов, т.е. каждый день присылается письмо, содержащее список сообщений за последние 24 часа. Чтобы подписаться на список, отправьте пустое письмо по специальному адресу список-subscribe `@lists.mysql.com`. Если нужно подписаться на получение дайджеста, добавьте к имени списка суффикс `-digest`. В ответ сервер рассылки пришлет подтверждающее письмо, на которое нужно ответить. Свои сообщения шлите на сервер, указывая имя списка, например `mysql@lists.mysql.com`. Чтобы отказаться от рассылки, отправьте пустое письмо по адресу *список*`-unsubscribe@lists.mysql.com`.

announce-subscribe@lists.mysql.com

В рамках этого списка распространяются анонсы о появлении новых версий MySQL.

mysql-subscribe@lists.mysql.com

Это общий список обсуждения всех аспектов программы MySQL, трафик в котором весьма высок. Чтобы получать ежедневные дайджесты, отправьте запрос по адресу `mysql-digest-subscribe@lists.mysql.com`.

bugs-subscribe@lists.mysql.com

В этот список направляются отчеты о найденных ошибках в программе. Адрес для получения дайджестов следующий: bugs-digest-subscribe@lists.mysql.com.

internals-subscribe@lists. mysql. com

Это список для разработчиков MySQL. Адрес для получения дайджестов следующий: `internals-digest-subscribe@lists.mysql.com`.

java-subscribe@lists. mysql. com

Это список для обсуждения вопросов совместного применения MySQL и Java, преимущественно посвященный JDBC. Адрес для получения дайджестов следующий: `java-digest-subscribe@lists.mysql.com`.

win32-subscribe@lists.mysql. com

Это список для обсуждения вопросов работы MySQL на платформе Windows. Адрес для получения дайджестов следующий: win32-digest-subscribe@lists `.mysql.com`.

myodbc-subscribe@lists.mysql. com

Это список для обсуждения драйвераODBC, имеющегося в MySQL. Адрес для получения дайджестов следующий: `myodbc-digest-subscribe@li.mysql.com`.

plusplus-subscribe@lists. mysql. com

Это список для обсуждения библиотеки MySQL++. Адрес для получения дайджестов следующий: `plusplus-digest-subscribe@lists.mysql.com`.

msql-mysql-modules-subscribe@lists.mysql. com

Это список для обсуждения средств языка Perl и модуля DBI, предназначенных для взаимодействия с MySQL. Адрес для получения дайджестов следующий: `msql-mysql-modules-digest-subscribe@lists.mysql.com`.

Архивы списков рассылки

Те, кто пытаются решить какую-то проблему, связанную с MySQL, могут заглянуть в архивы списков рассылки, где часто можно найти ответы на самые каверзные вопросы.

Официальные архивы

http://lists.mysql.com/

AIMS

http://marc.theaimsgroup.com/

Geocrawler

www.geocrawler.com/lists/3/Databases/

NexTrieve

http://www.nextrieve. com/cgi-bin/mysql

Yahoo

http://groups.yahoo. com/group/myodbc/

Web-узлы

Ниже приведен перечень Web-узлов, которые могут заинтересовать пользователей MySQL.

MySQL

www.mysql.com

Это официальный Web-узел компании MySQL AB. Отсюда можно начинать поиск информации.

www. devshed. com/Server_Side/MySQL/

Здесь публикуются статьи, посвященные различным Web-технологиям.

http://empyrean.lib. ndsu. nodak.edu/~nem/mysql/udf/

Это реестр UDF-функций.

http://netgraft.com/~mbac/research/mysqlmyths. html

Здесь обсуждаются различные аспекты MySQL, которые традиционно вызывали недоверие у пользователей других СУБД.

Реляционные базы данных

www.palslib.com

Здесь содержится перечень информационных ресурсов, посвященных реляционным СУБД.

www. dbdebunk.com

Здесь публикуются различные критические статьи о базах данных.

www. object-relational.com

Здесь рассматриваются вопросы объектно-реляционной модели.

www. ambysoft.com

Это Web-узел Скотта Амблера, где содержится много интересных документов, касающихся реляционных базданных.

*www.jcc. com/SQLPages/jccs_sql*htm

Эта Web-страница посвящена стандартам языка SQL.

www.sql.org

Здесь можно найти официальную информацию о языке SQL.

Отчеты об ошибках

Разработчики MySQL ведут специальный список рассылки (bugs@l ists .mysql.com), в котором публикуются отчеты об ошибках программы. Можно самостоятельно посылать туда свои сообщения, но лучше пользоваться сценарием `mysqlbug`, который способен автоматически генерировать информацию о системе. Чем точнее будут сведения о конфигурации, тем проще будет разработчикам разобраться в проблеме.

Прежде чем сообщать об ошибке, проверьте интерактивную документацию. Некоторые проблемы уже известны и описаны, особенно те, которые касаются особенностей конкретных операционных систем. На Web-узлах можно также встретить комментарии пользователей.

Далее обратитесь в архивы списков рассылки. Не исключено, что с подобной проблемой уже кто-то сталкивался. Ответы на жалобы пользователей быстро публикуются в списках.

В разделе 1.6.2.3 интерактивной документации описано, какую информацию необходимо включить в отчет об ошибке.

ПРАВОВЫЕ АСПЕКТЫ

В этом приложении…

Лицензирование программы MySQL
Общая лицензия GNU
Стабильность
Поддержка

Приложение Б

В этом приложении рассматриваются вопросы, касающиеся коммерческого использования программы MySQL.

Лицензирование программы MySQL

Текущие версии MySQL доступны на условиях общей лицензии GNU (GNU General Public License), которая широко популярна при распространении программ с открытыми исходными кодами. Любой пользователь может свободно использовать и модифицировать программу без каких-либо ограничений, при условии, что программа не передается другим пользователям. В случае распространения программы может потребоваться придерживаться определенных ограничений. Как правило, если программа модифицируется, то наряду с откомпилированной версией необходимо распространять также ее исходные коды. Этого можно избежать, если приобрести у компании MySQL AB коммерческую лицензию. Информация о процедуре получения лицензии доступна на Web-узле компании. Следует отметить, что стоимость коммерческой лицензии невысока в сравнении с другими СУБД.

Ниже приведен текст общей лицензии GNU. Дополнительную информацию о ней можно найти на Web-узле *www.gnu.org*.

Общая лицензия GNU

Версия 2, июнь 1991 года

Преамбула

Для большинства программных продуктов лицензии разрабатываются с целью запрещения их свободного распространения и внесения в них изменений. Данная общая лицензия GNU (GNU General Public License), наоборот, призвана гарантировать всем пользователям свободное обладание и изменение свободно распространяемых программных продуктов. Эта лицензия применяется к большинству программных продуктов организации FSF и любым другим программам, авторы которых берут на себя обязательство придерживаться ее. (Существует также ряд программных продуктов FSF, которые подчиняются правилам "библиотечной" лицензии GNU (GNU Library General Public License).) Вы можете применять ее к своим собственным программам.

Когда мы говорим о свободно распространяемом программном продукте, мы имеем в виду свободу, а не цену. Наши общие лицензии разрабатываются для того, чтобы предоставить вам свободу в распространении копий свободно распространяемых программных продуктов (при желании вы можете назначить цену за этот сервис); чтобы предоставить вам гарантию или возможность получения исходных кодов; чтобы вы могли вносить изменения в продукт или использовать его фрагменты для создания новых свободно распространяемых программ; а также для того, чтобы вы знали о возможности реализации всех перечисленных выше действий.

Для защиты ваших прав нам необходимо очертить круг ограничений, которые запрещают кому бы то ни было отказать вам в этих правах или просить вас отказаться от своих прав. Эти ограничения налагают на вас определенные обязательства, если вы распространяете копии некоторой программы или модифицируете ее.

Например, при распространении копии такой программы (бесплатно или за некоторую плату) вы должны передать получателям все права, которыми обладаете сами. Вы должны убедиться в том, что они также получат или могут получить исходный код. Кроме того, вы должны показать им текст данной лицензии, чтобы они тоже знали свои права.

Мы защищаем ваши права двумя способами: обеспечивая авторское право на программный продукт и предлагая вам эту лицензию, которая дает вам легальное разрешение на копирование, распространение и/или модификацию программы.

Кроме того, в целях защиты всех авторов (и нас в том числе) мы хотим, чтобы все четко понимали, что свободно распространяемый программный продукт свободен от каких бы то ни было гарантий. Иными словами, если программа модифицируется другими людьми и передается для последующих модификаций, то те, кому она попадает в руки, должны знать, что они получили не оригинал. Поэтому любые изменения, внесенные другими пользователями, не должны отразиться на репутации авторов.

И последнее. Любая свободно распространяемая программа постоянно находится под угрозой получения патента. Мы хотим избежать ситуации, при которой распространители могли бы приобрести лицензию на такую программу, став ее владельцем. Во избежание этого мы подчеркиваем, что любой патент должен быть лицензирован для беспрепятственного использования всеми желающими или не лицензирован вообще.

Ниже приводится подробное описание условий для копирования, распространения и модификации программных продуктов.

Условия копирования, распространения и модификации программных продуктов

Данная лицензия применяется к любой программе или другому продукту, который содержит замечание, внесенное владельцем авторских прав, где указано, что данный продукт может распространяться только на условиях общей лицензии GNU. Термин "Программа" относится к любой такой программе или продукту, а фраза "продукт, основанный на Программе" означает либо программу, либо любой производный продукт, для которого соблюдается авторское право, т.е. продукт, содержащий Программу или ее часть, в неизменном виде либо с модификациями и/или в переводе на другие языки (в дальнейшем возможность перевода подразумевается в термине "модификация"). Ко всем владельцам лицензий используется обращение во втором лице: вы, вам, ваш и т.д.

Данная лицензия не распространяется на действия, отличные от копирования, распространения и модификации. На запуск программы ограничения не накладываются, а результатов ее работы они касаются только в том случае, если эти результаты представляют собой продукт, основанный на Программе. Справедливость этого положения зависит от того, что делает Программа.

1. Вы можете копировать или распространять точные копии полученного вами исходного кода программы на любом носителе с нанесением на каждую копию соответствующего замечания об авторских правах и отказе от предоставления гарантии при условии, что вы сохраните в точности все замечания, в которых дается ссылка на эту лицензию и на отсутствие гарантии, а также передадите другим лицам копию этой лицензии вместе с Программой.

   Вы можете назначить цену за услугу по передаче копии, а также на ваше усмотрение предложить гарантийную защиту в обмен на определенную сумму.

2. Вы имеете право модифицировать свою копию (копии) Программы или любую ее часть, создавая таким образом продукт, основанный на Программе, а также копировать и распространять такие модификации или продукт при соблюдении условий п. 1, выполняя при этом следующие требования.

   а) Вы должны внести в модифицированные файлы замечания, которые нельзя не заметить и в которых сообщается о том, что вы изменили файлы, с обязательным указанием даты каждого изменения.

   б) Согласно условиям лицензии, вы должны бесплатно лицензировать любой продукт, который вы распространяете или публикуете и который содержит Программу целиком или частично либо является продуктом, производным от Программы или от одной из ее частей.

   в) Если при работе модифицированной программы обычно выполняется чтение команд в интерактивном режиме, вы должны сразу после запуска программы распечатать или отобразить на экране объявление, содержащее соответствующее замечание об авторских правах и отсутствии гарантии (или другое сообщение, если вы предоставляете какую-то гарантию), извещение о том, что пользователи могут распространять данную программу при соблюдении этих условий, а также указание, разъясняющее,

каким образом пользователь может пол учить копию данной лицензии. (Исключение: если Программа сама по себе работает в интерактивном режиме, но обычно не выводит подобные объявления, ваш продукт, основанный на Программе, не должен в обязательном порядке выводить указанное объявление.)

Эти требования относятся в целом к модифицированному продукту. Если в этом продукте есть разделы, которые не являются производными от Программы и могут рассматриваться как независимые продукты, работающие отдельно от других частей, то данная лицензия и ее условия не относятся к таким разделам, если вы распространяете их как отдельные продукты. Но если вы распространяете те же самые разделы в качестве составных частей одного целого, которые образуют продукт, основанный на Программе, распространение такого "целого" должно происходить на условиях данной лицензии, ограничения которой в этом случае расширяются на все составные части продукта, независимо от того, кто является их автором.

Цель этого раздела лицензии состоит не в том, чтобы оспаривать ваши права на продукт, полностью написанный вашими руками, а в заявлении права на осуществление контроля над распространением унаследованных или коллективных продуктов, основанных на Программе.

Необходимо также отметить, что простое объединение других продуктов, не основанных на Программе, с Программой (или с продуктами, основанными на Программе) на одном носителе информации или распространение такого носителя не вносит другие продукты *в* область действия данной лицензии.

3. Вы можете копировать или распространять Программу (или продукт, основанный на Программе, при соблюдении требований п. 2) в объектном коде или исполняемой форме при соблюдении условий пп. 1 и 2, следуя одному из нижеперечисленных требовании.

   а) Приложите к Программе соответствующий исходный код, который должен распространяться при соблюдении условий пп. 1 и 2 на носителях, обычно используемых для обмена программными продуктами.

   б) Приложите к Программе записанное предложение (действительное в течение по крайней мере трех лет) передать любым независимым исполнителям за плату, не превышающую стоимость физического выполнения операции доставки, полной машинной копии соответствующего исходного кода, распространяемого при соблюдении условий пп. 1 и 2 на носителях, обычно используемых для обмена программными продуктами.

   в) Приложите к Программе информацию, полученную вами в качестве предложения распространять соответствующий исходный код. (Эта альтернатива разрешена только для некоммерческой поставки и только в том случае, если вы получили программу в объектном коде или в исполняемой форме с таким предложением в соответствии со вторым подпунктом.)

Исходный код программы является предпочтительной формой для внесения изменений. Для исполняемого продукта полный исходный код означает наличие исходного кода всех модулей, содержащихся в продукте, плюс соответст-

вующие интерфейсные файлы и сценарии, используемые для управления компиляцией и установкой исполняемого файла. Но в виде исключения поставляемый исходный код не требует включения никаких компонентов, которые обычно распространяются (в исходном либо двоичном коде) с основными компонентами (компилятор, ядро и т.д.) операционной системы, где должен работать продукт, если только какой-то компонент сам не является обязательным приложением к исполняемому продукту.

Если поставка исполняемого или объектного кода выполнена в виде предложения обратиться к копии в обозначенном месте, то предложение эквивалентного доступа к копии исходного кода из того же самого источника расценивается как поставка исходного кода, даже если получателя не заставляют копировать исходный код вместе с объектным кодом.

4. Вы не можете копировать, модифицировать, лицензировать или распространять Программу в обход этой лицензии. Любые попытки поступить иначе с целью копирования, модификации, лицензирования или распространения Программы пресекаются с автоматическим аннулированием ваших прав, предусматриваемых лицензией. Но лица, получившие от вас копии или права в соответствии с лицензией, не лишаются своихлицензий до тех пор, пока они полностью соблюдают оговоренные лицензией условия.
5. От вас не требуется принимать условия лицензии, пока вы не подпишете ее. Но имейте в виду, что ничто иное не даст вам разрешение на модификацию или распространение Программы или ее производных продуктов. Эти действия запрещены законом, если вы не примете данную лицензию. Следовательно, модифицируя или распространяя Программу (или любой продукт, основанный на Программе), вы гарантируете свое принятие этой лицензии и всех ее условий и требований, которые необходимо соблюдать при копировании, распространении и модификации Программы или продуктов, основанных на ней.
6. Каждый раз, когда вы передаете Программу (или любой продукт, основанный на Программе) третьему лицу, оно автоматически получает от вас лицензию на право копировать, распространять или модифицировать Программу на указанных условиях. Вы не можете налагать какие бы то ни было дальнейшие ограничения на реализацию прав, данных получателю. Вы не несете ответственность за соблюдение другими лицами условий данной лицензии.
7. Если вследствие решения суда, или заявления о нарушении прав, или по какой-то другой причине (не связанной с вопросами авторских прав) поставленные перед вами условия (по постановлению суда, соглашению и т.п.) противоречат условиям данной лицензии, это не освобождает вас от ответственности за несоблюдение правил, предусматриваемых лицензией. Если вы не можете распространять Программу так, чтобы одновременно удовлетворять условиям, предусмотренным этой лицензией, и соблюдать другие обязательства, значит, вы вообще не можете заниматься распространением Программы. Например, если согласно некоторой лицензии не разрешается безгонорарное распространение Программы всеми, кто прямо или косвенно получает от вас копии, то единственный способ удовлетворить условия обеих лицензий заключается в полном отказе от распространения Программы.

Если какая-нибудь часть этого раздела не согласуется или не соблюдается при определенных обстоятельствах, то предполагае тся, что оставшаяся часть раздела или весь раздел в целом применяется при других обстоятельствах.

Целью этого раздела отнюдь не является склонять вас к нарушению другихправовых обязательств или оспаривать их обоснованность. Единственная цель этого раздела состоит в защите целостности системы распространения бесплатных программных продуктов, которая реализуется с помощью общих лицензий. Многие люди внесли огромный вклад в программное обеспечение, распространяемое через эту систему, будучи уверенными в ее постоянном применении. И только от решения самого автора зависит, будет ли он (или она) распространять программный продукт через **какую-либо** другую систему, и лицензия не может повлиять на этот выбор.

Предполагается, что в этом разделе внесена ясность в то, что считается **следствием** остальной части лицензии.

8. Если распространение и/или использование Программы ограничивается в определенных странах либо патентами, либо интерфейсами, защищенными **авторскими** правами, исходный владелец авторских прав, который вводит свою **Программу** под защиту этой лицензии, может добавить в явном виде ограничение на географическое распространение, перечислив страны, исключающиеся из **области** распространения (это значит, что распространение разрешено только **среди** тех стран, которые не входят в этот список). В этом случае лицензия включает ограничение, как если бы оно было записано в теле самой лицензии.
9. Организация FSF время от времени может публиковать модифицированную и/или новую версию общей лицензии GNU. Новые версии будут содержать ту же идею, что и настоящая версия, но могут отличаться в деталях, связанных с новыми проблемами или концепциями.

   Каждой версии присваивается отличительный номер. Если в Программе указан **номер** версии лицензии, который применяется к этой и "любым последующим версиям", у вас есть возможность следовать требованиям либо данной версии, либо **любой** из последующих версий, опубликованных организацией **FSF.** Если в Программе не указан номер версии Лицензии, вы можете выбрать любую версию, когда-либо опубликованную организацией FSF.
10. Если вы хотите объединить части Программы с другими свободно распространяемыми программами, условия распространения которых отличаются от описываемых, обратитесь к автору с просьбой о разрешении. Относительно **программных** продуктов, которые защищаются авторскими правами FSF, обращайтесь непосредственно в организацию FSF. Иногда мы делаем исключения. На наше решение влияет желание достичь двух целей: сохранить свободный статус всех продуктов, производных от наших свободно распространяемых **программ,** и продвинуть идеи совместного применения и многократного использования программ.

Гарантия отсутствует

11. Поскольку программа лицензируется бесплатно, для нее не существует никаких гарантий (до степени, разрешенной действующим законодательством). За исключением случаев, специально оговоренных в письменном виде, владельцы авторских прав и/или другие лица предоставляют программу "как есть" без какой-либо гарантии, явной или подразумеваемой, в том числе (но не только) подразумеваемой гарантии годности к продаже и пригодности к конкретному применению. Весь риск, связанный с качеством и выполнением программы, ложится на вас. В случае дефектности программы вам следует взять на себя стоимость всех необходимых доработок, поиска неисправностей и корректировки.
12. Ни при каких обстоятельствах, если того не требует закон или не указано в письменной форме, владелец авторских прав либо какое-то другое лицо, которое имело право модифицировать и/или распространять программу в соответствии с приведенными выше разрешениями, не несет перед вами ответственность за нанесенный программой ущерб, включающий любые повреждения общего, специального, случайного или косвенного характера, являющиеся следствием использования или невозможности использования программы (в том числе, но не только: потеря данных вами или другими лицами, неправильное представление данных или неспособность программы работать совместно с другими программами), даже если владелец или другое лицо было уведомлено о возможности подобного ущерба.

Конец условий

Приложение: как применить эти требования к новым программным продуктам

Если вы написали новую программу и хотите, чтобы любой человек смог ею свободно воспользоваться, лучше всего присвоить ей статус бесплатно распространяемого программного продукта. Тогда любой желающий в рамках приведенной выше лицензии сможет ее свободно распространять и модифицировать.

Для этого включите приведенные ниже строки в дистрибутивный пакет программы. Лучше всего поместить их в качестве комментария в начало каждого исходного файла, чтобы были хорошо заметны фразы об отсутствии гарантий. В каждый исходный файл должна быть, как минимум, включена строка "copyright", а также ссылка на то, где можно прочитать полный текст лицензионного соглашения.

В первой строке необходимо указать фамилию автора программы и идею ее создания.

Copyright © *год создания, фамилия автора*

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or

> FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
>
> You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA

Кроме того, обязательно поместите в дистрибутивный пакет свои координаты: почтовый адрес и/или адрес электронной почты.

Если программа работает в интерактивном режиме, сделайте так, чтобы при запуске выводилось короткое сообщение наподобие приведенного ниже.

> Gnomovision version 69, Copyright © *год создания, фамилия автора*
>
> Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'. This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

Выше были указаны две гипотетические команды 'show w' и 'show c', с помощью которых пользователь может просмотреть соответствующие разделы общей лицензии GNU. Конечно, имена команд могут быть другими. Более того, эти команды могут вызываться из меню или в результате щелчка мышью — это зависит от типа вашей программы.

При необходимости укажите также координаты работодателя (если вы работаете программистом) или учебного заведения (если вы студент), которые смогут подтвердить полный отказ от всех авторских прав на программу. Ниже приведен пример, где можно вставить в текст реальные имена.

> Yoyodyne, Inc., hereby disclaims all copyright interest in the program 'Gnomovision' (which makes passes at compilers) written by James Hacker.
>
> *Личная подпись, дата*
>
> *Должность*

Данное лицензионное соглашение не предусматривает использование программы или ее частей в коммерческих проектах. Если ваше программное обеспечение представляет собой библиотеку подпрограмм, вы можете разрешить подключать ее компоненты к коммерческим программам. При этом вместо общей лицензии GNU используйте библиотечную лицензию.

Стабильность

Программа MySQL состоит из множества модулей. В табл. Б.1 приведены сведения об их статусе. Версия 3.23 программы считается стабильной, и компания MySQL AB рекомендует использовать ее в промышленных условиях.

Свойство	***Стабильность***
Инструкция ALTER TABLE	Стабильно
Автоматическое восстановление таблиц MyISAM	Бета

Свойство	*Стабильность*
Базовые средства языка SQL	Стабильно
Таблицы BDB	Бета
Библиотека языка C	Стабильно
DBD	Стабильно
Полнотекстовый поиск	Бета
Инструкция GRANT	Стабильно
Таблицы **InnoDB**	Альфа
Обработчик таблиц ISAM	Стабильно
Модуль оптимизации объединений	Стабильно
Потоки Linux	Стабильно
Инструкции `LOAD` DATA. . ., `INSERT. . .SELECT`	Стабильно
Блокировка	Гамма
Таблицы MERGE	Бета/гамма
Библиотека **MIT-pthreads**	Стабильно
Обработчик таблиц **MyISAM**	Стабильно
MyODBC (ODBC SDK 2.5)	Гамма
Утилита `mysqlaccess`	Стабильно
Другие реализации потоков	Бета/гамма
Модуль синтаксического и лексического анализа	Стабильно
Модуль оптимизации запросов	Стабильно
Модуль оптимизации диапазонов	Стабильно
Репликация	Бета/гамма
Библиотека `pthreads` для Solaris 2.5+	Стабильно
Стандартные клиентские программы	Стабильно

Поддержка

Компания MySQL AB оказывает платные профессиональные услуги по вопросам поддержки пользователей MySQL. Диапазон услуг самый разный: от консультаций по телефону до регистрации инженеров компании на сервере клиента для решения проблем. Приведенная ниже информация взята с Web-узла компании (*www.mysql.com/support/index.html*).

Базовая поддержка по электронной почте

Поддержка данного типа очень недорога и представляет собой наилучший способ спонсирования разработчиков MySQL. Разработчики бесплатно консультируют пользователей во множестве списков рассылки, что становится возможным благодаря плате за базовую поддержку.

Расширенная поддержка по электронной почте

Поддержка данного типа дополнительно подразумевает, что запросы, поступающие от вашего адреса электронной почты, будут обрабатываться раньше, чем запросы незарегистрированных пользователей и пользователей, оплативших поддержку первого типа. Кроме того, будут рассматриваться ваши замечания, касающиеся дальнейшего развития программы.

Поддержка регистрации

Поддержка данного типа дополнительно подразумевает, что в случае возникновения проблем инженеры компании зарегистрируются в вашей системе для решения проблемы "на месте". Кроме того, они подскажут, как можно оптимизировать систему.

Расширенная поддержка регистрации

Поддержка данного типа дополнительно подразумевает, что инженеры проведут детальный анализ системы и помогут ее оптимизировать. Они могут также оптимизировать саму программу, чтобы она лучше соответствовала потребностям системы.

Телефонная поддержка

Поддержка данного типа дополнительно подразумевает, что в вашем распоряжении будет динамически обновляемая Web-страница с текущим списком разработчиков MySQL, которым можно позвонить в критической ситуации. Если проблема не критична, можно попросить одного из разработчиков перезвонить в течение 48 часов.

ЗАРЕЗЕРВИРОВАННЫЕ СЛОВА

Приложение B

ACTION
ADD
AFTER
AGGREGATE
ALL
ALTER
AND
AS
ASC
AUTO_INCREMENT
AVG
AVG_ROW_LENGTH
BETWEEN
BIGINT
BINARY
BIT
BLOB
CHARACTER
CHECK
CHECKSUM
COLUMN
COLUMNS
COMMENT
CONSTRAINT
CREATE
CROSS
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
DATA
DATABASE
DATABASES
DATE
DATETIME

BOOL
BOTH
BY
CASCADE
CASE
CHANGE
CHAR
DEC
DECIMAL
DEFAULT
DELAYED
DELAY_KEY_WRITE
DELETE
DESC
DESCRIBE
DISTINCT
DISTINCTROW
DOUBLE
DROP
ELSE
ENCLOSED
END
ENUM
ESCAPE
ESCAPED
EXISTS
EXPLAIN
FIELDS
FILE
FIRST
DAY
DAYOFMONTH
DAYOFWEEK
DAYOFYEAR
DAY_HOUR
DAY_MINUTE
DAY_SECOND
IN
INDEX
INFILE
INNER
INSERT
INSERT_ID
INT
INT1
INT2
INT3
INT4
INT8
INTEGER
INTERVAL
INTO
IS
ISAM
JOIN
KEY
KEYS
KILL
LAST_INSERT_ID
LEADING

FLOAT
FLOAT4
FLOAT8
FLUSH
FOR
FOREIGN
FROM
FULL
FUNCTION
GLOBAL
GRANT
GRANTS
GROUP
HAVING
HEAP
HIGH_PRIORITY
HOSTS
HOUR
HOUR_MINUTE
HOUR_SECOND
IDENTIFIED
IF
IGNORE
MODIFY
MONTH
MONTHNAME
MYISAM
NATURAL
NO
NOT
LEFT
LENGTH
LIKE
LIMIT
LINES
LOAD
LOCAL
LOCK
LOGS
LONG
LONGBLOB
LONGTEXT
LOW_PRIORITY
MATCH
MAX
MAX_ROWS
MEDIUMBLOB
MEDIUMINT
MEDIUMTEXT
MIDDLEINT
MINUTE
MINUTE_SECOND
MIN_ROWS
SQL_BIG_RESULT
SQL_BIG_SELECTS
SQL_BIG_TABLES
SQL_LOG_OFF
SQL_LOG_UPDATE
SQL_LOW_PRIORITY_UPDATES
SQL_SELECT LIMIT

NULL
NUMERIC
ON
OPTIMIZE
OPTION
OPTIONALLY
OR
ORDER
OUTER
OUTFILE
PACK_KEYS
PARTIAL
PASSWORD
PRECISION
PRIMARY
PRIVILEGES
PROCEDURE
PROCESS
PROCESSLIST
READ
REAL
REFERENCES
REGEXP
RELOAD
RENAME
REPLACE
RESTRICT
RETURNS
REVOKE
RLIKE
SQL_SMALL_RESULT
SQL_WARNINGS
STARTING
STATUS
STRAIGHT_JOIN
STRING
TABLE
TABLES
TEMPORARY
TERMINATED
TEXT
THEN
TIME
TIMESTAMP
TINYBLOB
TINYINT
TINYTEXT
TO
TRAILING
TYPE
UNIQUE
UNLOCK
UNSIGNED
UPDATE
USAGE
USE
USING
VALUES
VARBINARY
VARCHAR

ROW	VARIABLES
ROWS	VARYING
SECOND	WHEN
SELECT	WHERE
SET	WITH
SHOW	WRITE
SHUTDOWN	YEAR
SMALLINT	YEAR_MONTH
SONAME	ZEROFILL

КОДЫ ОШИБОК MYSQL

Приложение Г

В табл. Г.1 перечислены коды ошибок, возвращаемые утилитой `perror`.

Код	*Описание*
0	Успешное выполнение
1	Операция запрещена
2	Нет такого файла или каталога
3	Нет такого процесса
4	Прерванный системный вызов
5	Ошибка ввода/вывода
6	Устройство не сконфигурировано
7	Список аргументов слишком длинный
8	Ошибка формата функции `exec` О
9	Неправильный дескриптор файла
10	Нет дочерних процессов
11	Ресурс временно недоступен
12	Невозможно выделить память
18	Доступ запрещен
14	Неправильный адрес

Код	*Описание*
15	Необходимо устройство блочного доступа
16	Устройство или ресурс заняты
17	Файл существует
18	Неправильная ссылка на другое устройство
19	Нет такого устройства
20	Объект не является каталогом
21	Объект является каталогом
22	Неправильный аргумент
23	Слишком много открытых файлов в системе
24	Слишком много открытых файлов
25	Неправильный вызов функции `ioctl()` для устройства
26	Текстовый файл занят
27	Файл слишком велик
28	На устройстве не осталось свободного места
29	Неправильная операция перемещения по файлу
30	Файловая система доступна только для чтения
31	Слишком много ссылок
32	Разорванный канал
33	Недопустимое значение числового аргумента
34	Недопустимое значение числового результата
35	Удалось избежать взаимоблокировки ресурсов
36	Имя файла слишком велико
37	Блокировки недоступны
38	Функция не реализована
39	Каталог не является пустым
40	Слишком много уровней символических ссылок
41	Неизвестная ошибка
42	Отсутствует сообщение требуемого типа

Код	*Описание*
43	Идентификатор удален
44	Недопустимый номер канала
45	Уровень 2 не синхронизирован
46	Зависание уровня 3
47	Сброс уровня 3
48	Недопустимый номер ссылки
49	Драйвер протокола не подключен
50	Отсутствует доступная структура CSI
51	Зависание уровня 2
52	Неправильная операция обмена
53	Неправильный дескриптор запроса
54	Буфер обмена переполнен
55	Нет структуры anode
56	Неверный код запроса
57	Неправильный слот
58	Неизвестная ошибка
59	Неправильный формат файла шрифтов
60	Устройство не является символьным
61	Данные недоступны
62	Время истекло
63	Закончились ресурсы потоков ввода/вывода
64	Компьютер не подключен к сети
65	Пакет не инсталлирован
66	Объект находится в удаленной системе
67	Ссылка была повреждена
68	Ошибка анонсирования
69	Ошибка функции `srmount()` .
70	Коммуникационная ошибка при отправке данных

Код	*Описание*
71	Ошибка протокола
72	Попытка выполнить несколько переходов
73	Ошибка RFS
74	Неправильное сообщение
75	Значение слишком велико для соответствующего типаданных
76	Имя не уникально в сети
77	Неправильное состояние файлового дескриптора
78	Адрес в удаленной системе изменился
79	Доступ к совместно используемой библиотеке невозможен
80	Попытка доступа к поврежденной совместно используемой библиотеке
81	Секция .lib файла a.out повреждена
82	Попытка подключить слишком много совместно используемых библиотек
83	Невозможно напрямую обратиться к совместно используемой библиотеке
84	Неправильный или неполный многобайтовый символ
85	Прерванный системный вызов должен быть запущен повторно
86	Ошибка канала потоков ввода/вывода
87	Слишком много пользователей
88	Попытка выполнить операцию, разрешенную только для сокетов
89	Необходим адрес получателя
90	Сообщение слишком велико
91	Неправильный тип протокола для сокета
92	Протокол недоступен
93	Протокол не поддерживается
94	Тип сокета не поддерживается
95	Операция не поддерживается
96	Семейство протоколов не поддерживается

Код	*Описание*
97	Семейство адресов не поддерживается для данного протокола
98	Адрес уже используется
99	Невозможно назначить запрашиваемый адрес
100	Сеть не функционирует
101	Сеть недоступна
102	Разрыв сетевого соединения при сбросе системы
103	Программа разорвала соединение
104	Соединение разорвано противоположной стороной
105	Буфер переполнен
106	Точка доставки пакета уже подключена
107	Точка доставки пакета не подключена
108	Доставка пакета невозможна, так как принимающая сторона не функционирует
109	Слишком много ссылок; слияние невозможно
110	Соединение разорвано по истечении тайм-аута
111	В соединении отказано
112	Узел не функционирует
113	Нет маршрута к узлу
114	Операция уже выполняется
115	Операция начала выполняться
116	Устаревший дескриптор файла NFS
117	Структура требует очистки
118	Файл не соответствует заданному типу
119	Отсутствуют доступные семафоры XENIX
120	Файл соответствует заданному типу, но не найден ключ для операции чтения или обновления
121	Дублирующийся ключ в операции записи или обновления
122	Превышена дисковая квота

Код	*Описание*
123	Запись была изменена с момента последнего чтения; изменение является обратимым
124	Получен неверный индекс
126	Индексный файл поврежден / неверный формат файла
127	Файл записей поврежден
131	Команда не поддерживается базой данных
132	Старый файл базы данных
133	Перед обновлением не была прочитана ни одна строка
134	Запись уже была удалена (или файл записей поврежден)
135	Файл записей переполнен
136	Индексный файл переполнен
137	Записей больше нет (попытка чтения за пределами файла)
138	Табличное расширение не поддерживается
139	Слишком большая запись (>= 16 Мбайт)
140	Неправильные опции создания таблицы
141	Дублирующийся уникальный ключ либо дублирующееся ограничение уникальности в операции записи или обновления
142	Использован неизвестный набор символов
143	Конфликт определений таблицы типа MERGE и исходных таблиц
144	Таблица повреждена либо последняя операция восстановления завершилась неудачей
145	Таблица была помечена как поврежденная и должна быть восстановлена

В табл. Г.2 перечислены коды ошибок, возвращаемые функциями библиотеки языка C. Соответствующие сообщения определены в английской версии файла `errmsg.txt`.

Код	*Сообщение*
1000	hashchk
1001	isamchk
1002	NO
1003	YES
1004	Can't create file '%-.64s' (errno: %d)
1005	Can't create table '%-.64s' (errno: %d)
1006	Can't create database '%-.64s'. (errno: %d)
1007	Can't create database '%-.64s'. Database exists
1008	Can't drop database '%-.64s'. Database doesn't exist
1009	Error dropping database (can't delete '%-.64s', errno: %d)
1010	Error dropping database (can't rmdir '%-.64s', errno: %d)
1011	Error on delete of '%-.64s' (errno: %d)
1012	Can't read record in system table
1013	Can't get status of %-.64s' (errno: %d)
1014	Can't getworking directory (errno: %d)
1015	Can't lock file (errno: %d)
1016	Can't openfile: '%-.64s'. (errno: %d)
1017	Can't find file: '%-.64s' (errno: %d)
1018	Can't read dir of '%-.64s' (errno: %d)
1019	Can't change dir to '%-.64s' (errno: %d)
1020	Record has changed since last read in table '%-.64s'
1021	Disk full (%s). Waiting for someone to free some space....
1022	Can't write, duplicate key in table '%-.64s'
1023	Error on close of '%-.64s' (errno: %d)
1024	Error reading file '%-.64s' (errno: %d)

Код	*Описание*
1025	Error on rename of '%-.64s' to '%-.64s' (errno: %d)
1026	Error writing file '%-.64s' (errno: %d)
1027	'%-.64s' is locked against change
1028	Sort aborted
1029	View '%-.64s' doesn't exist for '%-.64s'
1030	Got error %d from table handler
1031	Table handler for '%-.64s' doesn't have this option
1032	Can't find record in '%-.64s'
1033	Incorrect information in file:'%-.64s'
1034	Incorrect key file for table: '%-.64s'. Try to repair it
1035	Old key file for table '%-.64s'; Repair it!
1036	Table '%-.64s' is read only
1037	Out of memory. Restart daemon and try again (needed %d bytes)
1038	Out of sort memory. Increase daemon sort buffer size
1039	Unexpected eof found when reading file '%-.64s' (errno: %d)
1040	Too many connections
1041	Out of memory; check if mysqld or some other process uses all available memory. If not you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space
1042	Can't get hostname for your address
1043	Bad handshake
1044	Access denied for user: '%-.32s@%-.64s' to database '%-.64s'
1045	Access denied for user: '%-.32s@%-.64s' (Using password: %s)
1046	No database selected
1047	Unknown command
1048	Column '%-.64s' cannot be null

Код	*Описание*
1049	Unknown database '%-.64s'
1050	Table '%-.64s' already exists
1051	Unknown table '%-.64s'
1052	Column: '%-.64s' in %-.64s isambiguous
1053	Server shutdown in progress
1054	Unknown column '%-.64s' in '%-.64s'
1055	'%-.64s' isn't in GROUP BY
1056	Can't group on '%-.64s'
1057	Statement has sum functions and columns in same statement
1058	Column count doesn't match value count
1059	Identifier name '%-.100s' is too long
1060	Duplicate column name '%-.64s'
1061	Duplicate key name '%-.64s'
1062	Duplicate entry '%-.64s' for key %d
1063	Incorrect column specifier for column '%-.64s'
1064	%s near '%-.80s' at line %d
1065	Query was empty
1066	Not unique table/alias: '%-.64s'
1067	Invalid default value for '%-.64s'
1068	Multiple primary key defined
1069	Too many keys specified. Max %d keysallowed
1070	Too many key parts specified. Max %d parts allowed
1071	Specified key was too long. Max key length is %d
1072	Key column '%-.64s' doesn't exist in table
1073	BLOB column '%-.64s' can't beused in key specification with the used table type

Код	*Описание*
1074	Too big column length for column '%-.64s' (max = %d). Use BLOB instead
1075	Incorrect table definition; there can only be one auto column and it must be defined as a key
1076	%s: ready for connections\n
1077	%s: Normal shutdown\n
1078	%s: Got signal %d. Aborting!\n
1079	%s: Shutdown Complete\n
1080	%s: Forcing close ofthread %ld user: '%-.32s'\n
1081	Can't create IP socket
1082	Table '%-.64s' has no index like the one used in CREATE INDEX. Recreate the table
1083	Field separator argument is not what is expected. Check the manual
1084	You can't use fixed rowlength with BLOBs. Please use 'fields terminated by'.
1085	The file '%-.64s' must be in the database directory or be readable by all
1086	File '%-.80s' already exists
1087	Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld
1088	Records: %ld Duplicates: %ld
1089	Incorrect sub part key. The used key part isn't a string or the used length is longer than the key part
1090	You can't delete all columns with ALTER TABLE. Use DROP TABLE instead
1091	Can't DROP '%-.64s'. Check that column/key exists
1092	Records: %ld Duplicates: %ld Warnings: %ld
1093	INSERT TABLE '%-.64s' isn't allowed in FROM table list
1094	Unknown thread id: %lu
1095	You are not owner of thread %lu
1096	No tables used
1097	Too many strings for column %-.64s and SET

Код	*Описание*
1098	Can't generate a unique log-filename %-.64s.(1-999)\n
1099	Table '%-.64s' was locked with a READ lock and can't be updated
1100	Table '%-.64s' was not locked with LOCK TABLES
1101	BLOB column '%-.64s' can't have a default value
1102	Incorrect database name '%-.100s'
1103	Incorrect table name '%-.100s'
1104	The SELECT would examine too many records and probably take a very long time. Check your WHERE and use SET OPTION SQL_BIG_SELECTS=1 if the SELECT is ok
1105	Unknown error
1106	Unknown procedure '%-.64s'
1107	Incorrect parameter count to procedure '%-.64s'
1108	Incorrect parameters to procedure '%-.64s'
1109	Unknown table '%-.64s' in %-.32s
1110	Column '%-.64s' specified twice
1111	Invalid use of group function
1112	Table '%-.64s' uses an extension that doesn't exist in this MySQL version
1113	A table must have at least 1 column
1114	The table '%-.64s' is full
1115	Unknown character set: '%-.64s'
1116	Too many tables. MySQL can only use %d tables in a join
1117	Too many columns
1118	Too big row size. The maximum row size, not counting BLOBs, is %d. You have to change some fields to BLOBs
1119	Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld -O thread_stack=#' to specify a bigger stack if needed
1120	Cross dependency found in OUTER JOIN. Examine your ON conditions

Код	*Описание*
1121	Column '%-.64s' is used with UNIQUE or INDEX but is not defined as NOT NULL
1122	Can't load function '%-.64s'
1123	Can't initialize function '%-.64s'; %-.80s
1124	No paths allowed for shared library
1125	Function '%-.64s' already exists
1126	Can't open shared library '%-.64s' (errno: %d %-.64s)
1127	Can't find function '%-.64s' in library
1128	Function '%-.64s' is not defined
1129	Host '%-.64s' is blocked because of many connection errors. Unblock with 'mysqladmin flush-hosts'
1130	Host '%-.64s' is not allowed to connect to this MySQL server
1131	You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords
1132	You must have privileges to update tables in the mysql database to be able to change passwords for others
1133	Can't find any matching row in the user table
1134	Rows matched: %ld Changed: %ld Warnings: %ld
1135	Can't create a new thread (errno %d). If you are not out of available memory, you can consult the manual for a possible OS-dependent bug
1136	Column count doesn't match value count at row %ld
1137	Can't reopen table: '%-.64s'
1138	Invalid use of NULL value
1139	Got error '%-.64s' from regexp
1140	Mixing of GROUP columns (MIN(),MAX(),COUNT()...) with no GROUP columns is illegal if there is no GROUP BY clause
1141	There is no such grant defined for user '%-.32s' on host '%-.64s'
1142	%-.16s command denied to user: '%-.32s@%-.64s' for table '%-.64s'

Код	*Описание*
1143	%-.16s command denied to user: '%-.32s@%-.64s' for column '%-.64s' in table '%-.64s'
1144	Illegal GRANT/REVOKE command. Please consult the manual for which privileges can be used
1145	The host or user argument to GRANT is too long
1146	Table '%-.64s.%-.64s' doesn't exist
1147	There is no such grant defined for user '%-.32s' on host '%-.64s' on table '%-.64s'
1148	The used command is not allowed with this MySQL version
1149	You have an error in your SQL syntax
1150	Delayed insert thread couldn't get requested lock for table %-.64s
1151	Too many delayed threads in use
1152	Aborted connection %ld to db: '%-.64s' user: '%-.32s' (%-.64s)
1153	Got a packet bigger than 'max_allowed_packet'
1154	Got a read error from the connection pipe
1155	Got an error from fcntl()
1156	Got packets out of order
1157	Couldn't uncompress communication packet
1158	Got an error reading communication packets
1159	Got timeout reading communicationpackets
1160	Got an error writing communication packets
1161	Got timeout writing communication packets
1162	Result string is longer than max_allowed_packet
1163	The used table type doesn't support BLOB/TEXT columns
1164	The used table type doesn't support AUTO_INCREMENT columns
1165	INSERT DELAYED can't be used with table '%-.64s', because it is locked with LOCK TABLES

Код	*Описание*
1166	Incorrect column name '%-.100s'
1167	The used table handler can't index column '%-.64s'
1168	All tables in the MERGE table are not identically defined
1169	Can't write, because of unique constraint, to table '%-.64s'
1170	BLOB column '%-.64s' used in key specification without a key length
1171	All parts of a PRIMARY KEY must be NOT NULL; If you need NULL in a key, use UNIQUE instead
1172	Result consisted of more than one row
1173	This table type requires a primary key
1174	This version of MySQL is not compiled with RAID support
1175	You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column
1176	Key '%-.64s' doesn't exist in table '%-.64s'
1177	Can't open table
1178	The handler for the table doesn't support check/repair
1179	You are not allowed to execute this command in a transaction
1180	Got error %d during COMMIT
1181	Got error %d during ROLLBACK
1182	Got error %d during FLUSH_LOGS
1183	Got error %d during CHECKPOINT
1184	Aborted connection %ld to db: '%-.64s' user: '%-.32s' host: `%-.64s' (%-.64s)
1185	The handler for the table does not support binary table dump
1186	Binlog closed, cannot RESET MASTER
1187	Failed rebuilding the index of dumped table '%-.64s'
1188	Error from master: '%-.64s'
1189	Net error reading from master

Код	*Описание*
1190	Net error writing to master
1191	Can't find FULLTEXT index matching the column list
1192	Can't execute the given command because you have active locked tables or an active transaction
1193	Unknown system variable '%-.64'
1194	Table '%-.64s' is marked as crashed and should be repaired
1195	Table '%-.64s' is marked as crashed and last (automatic?) repair failed
1196	Warning: Some nontransactional changed tables couldn't be rolled back
1197	Multistatement transaction required more than 'max_binlog_cache_size' bytes of storage. Increase this mysqld variable and try again
1198	This operation cannot be performed with a running slave, run SLAVE STOP first
1199	This operation requires a running slave, configure slave and do SLAVE START
1200	The server is not configured as slave, fix in config file or with CHANGE MASTER TO
1201	Could not initialize master info structure, check permissions on master.info
1202	Could not create slave thread, check system resources
1203	User %-.64s has already more than 'max_user_connections' active connections
1204	You may only use constant expressions with SET

РУКОВОДСТВО ПО ОФОРМЛЕНИЮ SQL-СЦЕНАРИЕВ

В этом приложении…

Общие правила
Идентификаторы
Таблицы
Инструкции

Приложение Д

В этом приложении приведено краткое руководство по стилю оформления SQL-листингов. Не рассматривайте изложенные здесь правила как единственно верный способ записи инструкций SQL. Никто не запрещает вам иметь свой стиль. Согласованность важнее, чем единообразие.

Данное руководство касается листингов, сохраняемых в текстовом файле или иным способом передаваемых другим пользователям. К текстам инструкций, набираемых в оболочке интерпретатора команд, предъявляется только одно требование: скорость ввода.

Общие правила

Начинайте каждую инструкцию с нулевой колонки и разбивайте ее на несколько строк, если длина инструкции превышает 79 символов. Все последующие строки должны начинаться с отступа. Лучше всего разбивать строки непосредственно перед идентификатором предложения. В случае необходимости вставляйте разрыв строки после запятой. Каждое условие отбора в предложении WHERE должно записываться в отдельной строке. Приведем пример:

```
SELECT
       i.ID,
       i.Name,
       i.Department,
       s.ID,
       s.Name,
       s.Price
  FROM item i  INNER JOIN sku s ON  (i.ID = s.Item)
  WHERE i.Department = 3
    AND s.Price > 5.00
  ORDER BY i.Name
```

Идентификаторы

Записывайте зарезервированные слова MySQL проп исными буквами. Имена баз данных и таблиц должны состоять только из строчных букв. Имена столбцов должны начинаться с прописной буквы, как и отдельные слова в имени столбца, например `PostalCode`.

Имя промежуточной таблицы, которая служит для представления отношения "многие ко многим", должно состоять из имен исходных таблиц, разделенных символом подчеркивания, например sku_variation. Не используйте символы подчеркивания в именах столбцов. Также не рекомендуется использовать идентификаторы с цифрами.

В инструкциях разрешается применять псевдонимы. Старайтесь выбирать однобуквенные псевдонимы. В случае необходимости можно воспользоваться второй буквой, как показано в следующем примере:

```
SELECT s.Name, v.Name
  FROM sku s INNER JOIN sku_variation sv
                ON (s.ID = sv.SKU)
             INNER JOIN variation v
                ON (sv.Variation = v.ID)
```

Таблицы

По возможности старайтесь использовать столбец-счетчик в качестве первичного ключа. Назовите его ID и сделайте первым столбцом.

Сначала должны быть перечислены все столбцы, а после них — индексы. Все определения могут сопровождаться комментариями. Внешние ключи игнорируются в MySQL версии3.23, но они послужат документацией к схеме базы данных.

Имя внешнего ключа должно соответствовать имени таблицы, на которую он ссылается. Например, в следующей инструкции столбец Attribute ссылается на таблицу attribute:

```
CREATE TABLE variation (
    /* столбцы */
    ID INT(11) NOT NULL AUTO_INCREMENT,
    Name CHAR(80) NOT NULL,
    Attribute INT(11) NOT NULL,
    Description CHAR(255),
    Graphic  CHAR(255),
    DisplayPrecedence INT(11) NOT NULL,

    /* индексы */
    PRIMARY KEY(ID),
    KEY Name(Name),
    KEY Attribute(Attribute),
    KEY DisplayPrecedence(DisplayPrecedence)
)
```

Не включайте имена таблиц в имена столбцов. Записи `variation.ID` вполне достаточно, тогда как запись `variation.variation ID` явно избыточна.

Инструкции

При записи инструкций SELECT пользуйтесь оператором JOIN для создания **объединений**, а не операторами сравнения в предложении WHERE. Это позволяет легко определять, какие выражения задают правила объединения таблиц, а какие — ограничивают число записей в таблице результатов запроса.

В инструкциях INSERT указывайте только те столбцы, значения которых отличаются от заданных по умолчанию. **Поля-счетчики** не нужно указывать вообще. Если есть возможность, заменяйте группы одиночный инструкций INSERT многострочными инструкциями.

ПРИМЕР БАЗЫ ДАННЫХ

В этом приложении.

Диаграммы
Схема базы данных

Приложение E

В этом приложении приведена схема базы данных для проекта FreeTrade *(http://share.whichever.com/freetrade)*. Он представляет собой набор средств электронной коммерции, написанный на PHP. Я начал вести этот открытый проект в 1999 г. и применял его для создания нескольких Web-магазинов. Как и любой открытый проект, он постоянно находится в стадии доработки, поэтому в нем могут быть ошибки.

Диаграммы

На рис. E.1, E.2 и E3 изображены диаграммы базы данных. Единая диаграмма не смогла бы поместиться на одной странице, поэтому я разбил ее на несколько секций. Прямоугольники с пунктирным контуром представляют собой таблицы, которые определены в других секциях.

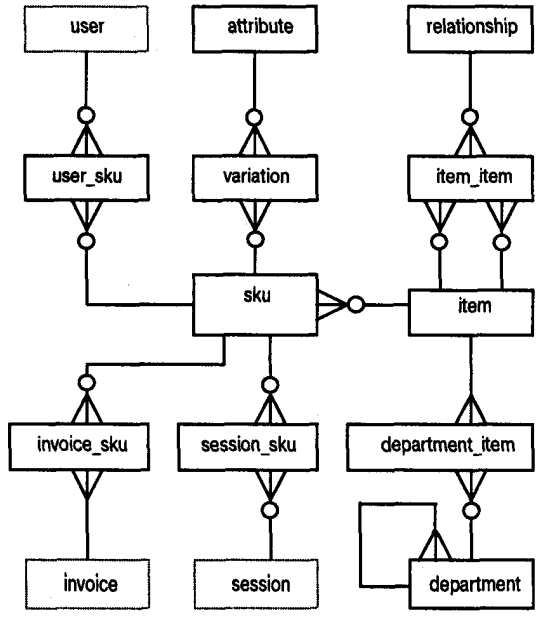


Рис. E. I. Таблицы каталога

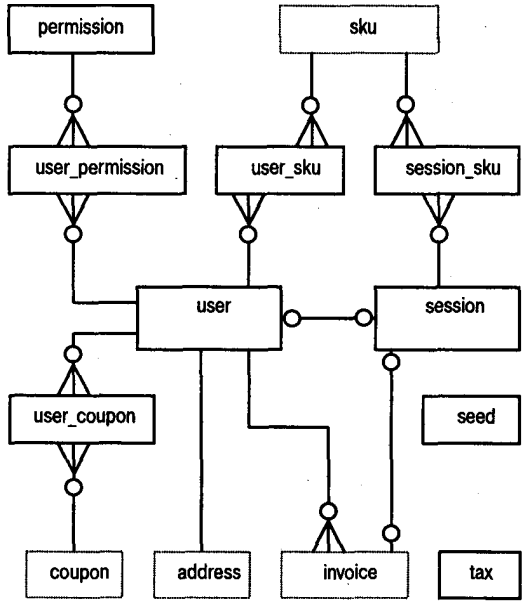


Рис. E.2. Таблицы пользователей

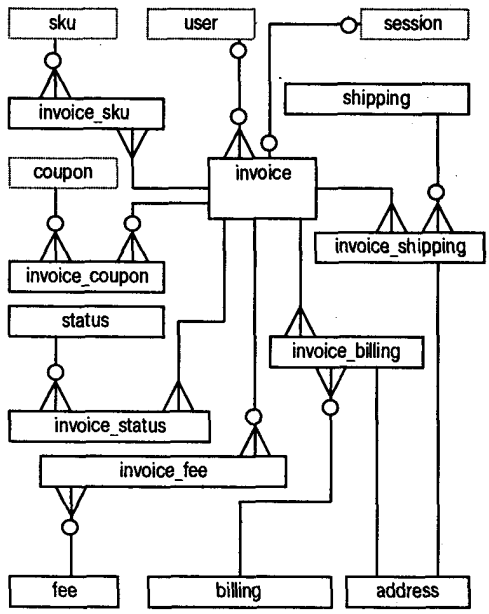


Рис. E.3. Таблицы заказов

Схема базы данных

Ниже приведены инструкции SQL, требуемые для реализации базы данных проекта **FreeTrade.**

```
/*
** Глобальные адреса для заказов и пользователей.
*/
DROP TABLE IF EXISTS address;
CREATE TABLE address (
    ID INT NOT NULL AUTO_INCREMENT,
    Name_Prefix CHAR(16),
    Name_First CHAR(255),
    Name_Middle CHAR(255),
    Name_Last CHAR(255) NOT NULL,
    Name_Suffix CHAR(16),
    Company CHAR(255),
    Street1 CHAR(255),
    Street2 CHAR(255),
    Street3 CHAR(255),
    City CHAR(255) NOT NULL,
    StateProv CHAR(255) NOT NULL,
    PostalCode CHAR(255) NOT NULL,
    CountryCode CHAR(2), /* ISO 3166 */
    Phone1 CHAR(32),
    Phone2 CHAR(32),
    Fax CHAR(32),
    Email CHAR(255) NOT NULL,

    PRIMARY KEY(ID)
);

/*
** Атрибуты товаров (цвет, размер).
*/
DROP TABLE IF EXISTS attribute;
CREATE TABLE attribute (
    ID INT NOT NULL AUTO_INCREMENT,
    Name CHAR(64) NOT NULL,
    Graphic CHAR(255),
    DisplayPrecedence INT NOT NULL,

    PRIMARY KEY(ID),
    INDEX (DisplayPrecedence, Name)
);

/*
** Варианты оплаты счетов (Visa, MasterCard).
*/
DROP TABLE IF EXISTS billing;
CREATE TABLE billing (
    ID INT NOT NULL AUTO_INCREMENT,
    ECML CHAR(4),
    Name CHAR(64) NOT NULL,
    DisplayPrecedence INT NOT NULL,
```

```
    PRIMARY KEY(ID),
    INDEX (DisplayPrecedence)
);


** Скидки.
*/
DROP TABLE IF EXISTS coupon;
CREATE TABLE coupon (
    ID INT DEFAULT '0' NOT NULL AUTO_INCREMENT,
    Name CHAR(32) NOT NULL,
    DollarOff DECIMAL(5,2),
    PercentageOff DECIMAL(5,2),
    MinAmountPurchased DECIMAL(5,2),
    StartDate TIMESTAMP(14),
    EndDate TIMESTAMP(14),
    NeverExpires ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Combineable ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    DisplayPrecedence INT,

    PRIMARY KEY (ID),
    INDEX (Name),
    INDEX (DisplayPrecedence)
);

/*
** Таблица контроля скидок, чтобы пользователь не мог
** дважды воспользоваться одной и той же скидкой.
*/
DROP TABLE IF EXISTS coupon_user;
CREATE TABLE coupon_user (
    Coupon INT NOT NULL,
    User INT NOT NULL,

    PRIMARY KEY (Coupon, User),

    FOREIGN KEY (Coupon) REFERENCES coupon (ID),
    FOREIGN KEY (User) REFERENCES user (ID)
);

/*
** Иерархия категорий товаров.
*/
DROP TABLE IF EXISTS department;
CREATE TABLE department (
    ID INT NOT NULL AUTO_INCREMENT,
    Name CHAR(64) NOT NULL,
    Graphic CHAR(255),
    Parent INT NOT NULL,
    Description blob,
    DisplayPrecedence INT NOT NULL,

    PRIMARY KEY(ID),
    INDEX (Parent),
    INDEX (DisplayPrecedence, Name),
```

```
    FOREIGN KEY (Parent) REFERENCES department (ID)
);

/*
** Таблица сопоставления элементов каталога
** различным категориям.
*/
DROP TABLE IF EXISTS department_item;
CREATE TABLE department_item (
    Department INT NOT NULL,
    Item INT NOT NULL,

    PRIMARY KEY(Department, Item),

    FOREIGN KEY (Department) REFERENCES department (ID),
    FOREIGN KEY (Item) REFERENCES item (ID)
);

/*
** Виды проплат, взимаемых по счетам.
*/
DROP TABLE IF EXISTS fee;
CREATE TABLE fee (
    ID INT NOT NULL AUTO_INCREMENT,
    Name CHAR(64) NOT NULL,

    PRIMARY KEY(ID),
    INDEX (Name)
);

/*
** Перечень сделанных заказов.
*/
DROP TABLE IF EXISTS invoice;
CREATE TABLE invoice (
    ID INT NOT NULL AUTO_INCREMENT,
    User INT NOT NULL,
    Active ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    Created DATETIME NOT NULL,

    PRIMARY KEY (ID),
    INDEX (User),

    FOREIGN KEY (User) REFERENCES user (ID)
);

/*
** Информация о способе оплаты заказа.
*/
DROP TABLE IF EXISTS invoice_billing;
CREATE TABLE invoice_billing (
    ID INT NOT NULL AUTO_INCREMENT,
    Invoice INT NOT NULL,
    Billing INT NOT NULL,
    Address INT NOT NULL,
```

```
    # Информация о кредитной карточке.
    CreditCardOwner CHAR(64) NOT NULL,
    CreditCardNumber CHAR(32) NOT NULL,
    CreditCardExpiration DATETIME NOT NULL,

    # Дополнительная информация о владельце кредитной
    # карточки, напечатанная на самой карточке, но не
    # закодированная в магнитной полосе.
    CreditCardVerification CHAR(4),

    PRIMARY KEY(ID),
    INDEX (Invoice),

    FOREIGN KEY (Invoice) REFERENCES invoice (ID),
    FOREIGN KEY (Address) REFERENCES address (ID),
    FOREIGN KEY (Billing) REFERENCES billing (ID)
);

/*
** Скидки, применимые к заказу.
*/
DROP TABLE IF EXISTS invoice_coupon;
CREATE TABLE invoice_coupon (
    Invoice INT NOT NULL,
    Coupon INT NOT NULL,

    PRIMARY KEY (Invoice, Coupon),

    FOREIGN KEY (Invoice) REFERENCES invoice (ID),
    FOREIGN KEY (Coupon) REFERENCES coupon (ID)
);

/*
** Счета, выставленные к оплате по сделанным заказам.
*/
DROP TABLE IF EXISTS invoice_fee;
CREATE TABLE invoice_fee (
    Invoice INT NOT NULL,
    Fee INT NOT NULL,
    Value DECIMAL(11,2) NOT NULL,

    INDEX (Invoice),
    INDEX (Fee),

    FOREIGN KEY (Invoice) REFERENCES invoice (ID),
    FOREIGN KEY (Fee) REFERENCES fee (ID)
);

/*
** Состояние заказа.
*/
DROP TABLE IF EXISTS invoice_status;
CREATE TABLE invoice_status (
    ID INT NOT NULL AUTO_INCREMENT,
    Invoice INT NOT NULL,
    Status INT NOT NULL,
```

```
   Created DATETIME NOT NULL,
   Description CHAR(255),

   PRIMARY KEY(ID),
   INDEX (Invoice),

   FOREIGN KEY (Invoice) REFERENCES invoice (ID),
   FOREIGN KEY (Status) REFERENCES status (ID)
);

/*
** Адрес доставки заказа.
*/
DROP TABLE IF EXISTS invoice_shipping;
CREATE TABLE invoice_shipping (
   ID INT NOT NULL AUTO_INCREMENT,
   Invoice INT NOT NULL,
   Address INT NOT NULL,
   Shipping INT NOT NULL,
   Message CHAR(255),

   PRIMARY KEY(ID),
   INDEX (Invoice),

   FOREIGN KEY (Invoice) REFERENCES invoice (ID),
   FOREIGN KEY (Address) REFERENCES address (ID),
   FOREIGN KEY (Shipping) REFERENCES shipping (ID)
);

/*
** Перечень товаров, входящих в заказ.
*/
DROP TABLE IF EXISTS invoice_sku;
CREATE TABLE invoice_sku (
   ID INT NOT NULL AUTO_INCREMENT,
   Invoice INT NOT NULL,
   SKU INT NOT NULL,
   Quantity INT NOT NULL,
   ExternalSKU CHAR(64) NOT NULL,
   Name CHAR(64) NOT NULL,
   ListPrice DECIMAL(11,2) NOT NULL,
   SalePrice DECIMAL(11,2) NOT NULL,
   Freight DECIMAL(11,2) NOT NULL,
   Shipping INT NOT NULL,
   GiftWrap char(1) NOT NULL DEFAULT 'N',

   PRIMARY KEY(ID),
   INDEX (Invoice),
   INDEX (SKU),
   INDEX (ExternalSKU),

   FOREIGN KEY (Invoice) REFERENCES invoice (ID),
   FOREIGN KEY (SKU) REFERENCES sku (ID)
);

/*
```

```
** Варианты исполнения товаров, входящих в заказ.
*/
DROP TABLE IF EXISTS invoice_sku_variation;
CREATE TABLE invoice_sku_variation (
    InvoiceSKU INT NOT NULL,
    Variation INT NOT NULL,
    Qualifier CHAR(255),

    PRIMARY KEY(InvoiceSKU, Variation)
);

/*
** Элементы каталога, которые могут быть представлены
** в одном или нескольких вариантах.
*/
DROP TABLE IF EXISTS item;
CREATE TABLE item (
    ID INT NOT NULL AUTO_INCREMENT,
    Name CHAR(64) NOT NULL,
    Description BLOB,
    Keywords VARCHAR(255),
    Thumbnail VARCHAR(255),
    Graphic VARCHAR(255),
    LargeGraphic VARCHAR(255),
    DisplayPrecedence INT,
    Active ENUM('N', 'Y') NOT NULL DEFAULT 'N',

    PRIMARY KEY(ID),
    INDEX (Name),
    INDEX (Active)
);

/*
** Произвольные отношения между элементами каталога.
*/
DROP TABLE IF EXISTS item_item;
CREATE TABLE item_item (
    Item INT NOT NULL,
    Related_Item INT NOT NULL,
    Relationship INT NOT NULL,

    PRIMARY KEY (Item, Related_Item, Relationship),

    FOREIGN KEY (Item) REFERENCES item (ID),
    FOREIGN KEY (Related_Item) REFERENCES item (ID),
    FOREIGN KEY (Relationship) REFERENCES relationship (ID)
);

/*
** Права пользователей системы.
*/
DROP TABLE IF EXISTS permission;
CREATE TABLE permission (
    ID INT NOT NULL AUTO_INCREMENT,
    Name CHAR(255) NOT NULL,
```

```
    PRIMARY KEY(ID),
    INDEX (Name)
);

/*
** Типы отношений между элементами каталога.
*/
DROP TABLE IF EXISTS relationship;
CREATE TABLE relationship (
    ID INT NOT NULL AUTO_INCREMENT,
    Name CHAR(255) NOT NULL,
    DisplayPrecedence INT NOT NULL,

    PRIMARY KEY(ID),
    INDEX (DisplayPrecedence, Name)
);

/*
** Таблица, в которой хранится инициализирующее значение
** генератора псевдослучайных чисел.
*/
DROP TABLE IF EXISTS seed;
CREATE TABLE seed (
    seed INT NOT NULL DEFAULT 314
);

/*
** Пользовательские сеансы.
*/
DROP TABLE IF EXISTS session;
CREATE TABLE session (
    ID CHAR(16) NOT NULL,
    User INT,
    LastAction DATETIME,
    Invoice INT,

    PRIMARY KEY(ID),
    INDEX (User),
    INDEX (Invoice),

    FOREIGN KEY (User) REFERENCES user (ID),
    FOREIGN KEY (Invoice) REFERENCES invoice (ID)
);

/*
** Покупательские корзины (списки товаров, заказанных
** в том или ином сеансе).
*/
DROP TABLE IF EXISTS session_sku;
CREATE TABLE session_sku (
    ID INT NOT NULL AUTO_INCREMENT,
    Session CHAR(24) NOT NULL,
    SKU INT NOT NULL,
    Quantity INT NOT NULL,
    Notes blob,
```

```
    PRIMARY KEY(ID),
    INDEX (Session),
    INDEX (SKU),

    FOREIGN KEY (Session) REFERENCES session (ID),
    FOREIGN KEY (SKU) REFERENCES sku (ID)
);

/*
** Варианты исполнения товаров, находящихся в корзине.
*/
DROP TABLE IF EXISTS session_sku_variation;
CREATE TABLE session_sku_variation (
    SessionSKU INT NOT NULL,
    Variation INT NOT NULL,
    Qualifier CHAR(255),

    PRIMARY KEY(SessionSKU, Variation)
);

/*
** Типы доставки (UPS, FedEx).
*/
DROP TABLE IF EXISTS shipping;
CREATE TABLE shipping (
    ID INT NOT NULL AUTO_INCREMENT,
    Name CHAR(255) NOT NULL,
    DisplayPrecedence INT NOT NULL,

    PRIMARY KEY(ID),
    INDEX (DisplayPrecedence, Name)
);

/*
** Товары, которые можно купить, сгруппированные по полю
** 'item' (элемент каталога).
*/
DROP TABLE IF EXISTS sku;
CREATE TABLE sku (
    ID INT NOT NULL AUTO_INCREMENT,
    Item INT NOT NULL,
    ExternalSKU CHAR(64) NOT NULL,
    Name CHAR(64) NOT NULL,
    ListPrice DECIMAL(11,2) NOT NULL,
    SalePrice DECIMAL(11,2) NOT NULL,
    AdditionalShipping DECIMAL(11,2) NOT NULL,
    DisplayPrecedence INT NOT NULL,
    Active ENUM('N', 'Y') NOT NULL DEFAULT 'N',
    InventoryAvailable INT NOT NULL,
    CanBackorder ENUM('N', 'Y') NOT NULL DEFAULT 'N',

    PRIMARY KEY(ID),
    INDEX (Item),
    INDEX (ExternalSKU),
    INDEX (DisplayPrecedence, Name),
```

```
    FOREIGN KEY (Item) REFERENCES item (ID)
);

/*
** Варианты исполнения товаров.
*/
DROP TABLE IF EXISTS sku_variation;
CREATE TABLE sku_variation (
    SKU INT NOT NULL,
    Variation INT NOT NULL,
    PRIMARY KEY (SKU, Variation),

    FOREIGN KEY (SKU) REFERENCES sku (ID),
    FOREIGN KEY (Variation) REFERENCES variation (ID)
);

/*
** Типы сообщений, описывающих состояние заказа.
*/
DROP TABLE IF EXISTS status;
CREATE TABLE status (
    ID INT NOT NULL AUTO_INCREMENT,
    Name CHAR(64) NOT NULL,

    PRIMARY KEY(ID),
    INDEX (Name)
);

/*
** Ставки налогов для США.
*/
DROP TABLE IF EXISTS tax;
CREATE TABLE tax (
    State CHAR(2) NOT NULL,
    Rate DECIMAL(4,5) NOT NULL,
    TaxShipping ENUM('N', 'Y') NOT NULL DEFAULT 'N',

    PRIMARY KEY(State)
);

/*
** Клиенты и системные администраторы.
*/
DROP TABLE IF EXISTS user;
CREATE TABLE user (
    ID INT NOT NULL AUTO_INCREMENT,
    Login CHAR(32) NOT NULL,
    Password CHAR(32) NOT NULL,
    Address INT NOT NULL,

    PRIMARY KEY(ID),
    INDEX (Login),
    INDEX (Address),

    FOREIGN KEY (Address) REFERENCES address (ID)
);
```

```
/*
** Права пользователей.
*/
DROP TABLE IF EXISTS user_permission;
CREATE TABLE user_permission  (
   User INT NOT NULL,
   Permission INT NOT NULL,

   PRIMARY KEY  (User,  Permission),
   FOREIGN KEY  (User)  REFERENCES user  (ID),
   FOREIGN KEY  (Permission)  REFERENCES permission  (ID)
);

/*
** Списки товаров, заказываемых пользователями.
*/
DROP TABLE IF EXISTS user_sku;
CREATE TABLE user_sku  (
   ID INT NOT NULL AUTO_INCREMENT,
   User INT NOT NULL,
   SKU INT NOT NULL,
   Quantity INT NOT NULL,

   PRIMARY KEY(ID),
   INDEX  (User),
   INDEX  (SKU),

   FOREIGN KEY  (User)  REFERENCES user  (ID),
   FOREIGN KEY  (SKU)  REFERENCES sku  (ID)
);

/*
** Варианты, выбранные пользователями.
*/
DROP TABLE IF EXISTS user_sku_variation;
CREATE TABLE user_sku_variation  (
   UserSKU INT NOT NULL,
   Variation INT NOT NULL,
   Qualifier CHAR(255),

   PRIMARY KEY(UserSKU, Variation)
);

/*
** Варианты исполнения для того или иного атрибута
**  (например, S, M, L для размера).
*/
DROP TABLE IF EXISTS variation;
CREATE TABLE variation  (
   ID INT NOT NULL AUTO_INCREMENT,
   Name CHAR(64)  NOT NULL,
   Attribute INT NOT NULL,
   Description CHAR(64),
   Graphic CHAR(255) ,
   DisplayPrecedence INT NOT NULL,
```

```
    PRIMARY KEY(ID),
    INDEX  (Attribute),
    INDEX  (DisplayPrecedence, Name),

    FOREIGN KEY  (Attribute)  REFERENCES attribute  (ID)
);
```

У

Ф

Научно-популярное издание

Леон Аткинсон

MySQL.
Библиотека профессионала

Литературный редактор *И.А. Попова*
Верстка *А.В. Плаксюк*
Художественный редактор *С.А. Чернокозинский*
Корректоры *Л.В. Коровкина, Т.А. Корзун*

Издательский дом "Вильяме".
101509, Москва, ул. Лесная, д. 43, стр. 1.
Изд.лиц. ЛР № 090230 от 23.06.99
Госкомитета РФ по печати.

Подписано в печать 5.06.2002. Формат 70x100/16.
Гарнитура NewBaskerville. Печать офсетная.
Усл. печ. л. 33,80. Уч.-изд. л. 26,20.
Тираж 3500 экз. Заказ № 577.

Отпечатано с диапозитивов в ФГУП "Печатный двор"
Министерства РФ по делам печати,
телерадиовещания и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.