Вводный курс для разработчиков и администраторов БД

Изучаем SQL

Символ®

O'REILLY®

Алан Бьюли

Learning SQL

Alan Beaulieu

O'REILLY®

Изучаем SQL

Алан Бьюли

Санкт-Петербург — Москва
2007

Алан Бьюли

Изучаем SQL

Перевод Н. Шатохиной

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*В. Коренев*
Редактор	*А. Кузнецов*
Корректор	*Н. Ткачева*
Верстка	*Д. Орлова*

Бьюли А.
Изучаем SQL. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 312 с., ил.
ISBN-13: 978-5-93286-051-9
ISBN-10: 5-93286-051-0

Книга Алана Бьюли, эксперта по языку SQL, – прекрасный учебник для тех, кто еще не знает, но хочет освоить этот язык. Книга не только позволит приобрести начальные знания, но и расскажет о наиболее часто употребляемых мощных средствах языка SQL, используемых опытными программистами.

Многие книги, посвященные SQL, грешат скучным изложением основ. Здесь же автор в стиле живого рассказа обсуждает SQL-выражения и блоки, различные типы условий, показывает, как посредством соединения таблиц создавать запросы к нескольким таблицам, рассматривает наборы данных и как они могут взаимодействовать в запросах, демонстрирует встроенные и агрегатные функции, показывает, как и где используются подзапросы. Подробно описаны различные типы соединений таблиц, применение условной логики, работа с транзакциями, индексы и ограничения.

Поскольку лучший способ изучения SQL – это практика, автор создает учебную базу данных MySQL и приводит множество вариантов реальных запросов, охватывающих весь теоретический материал. При таком подходе не научиться просто невозможно. Примеры кода можно использовать в своих программах и документации. Книга предназначена разработчикам приложений БД, администраторам БД и тем, кто создает отчеты.

ISBN-13: 978-5-93286-051-9
ISBN-10: 5-93286-051-0
ISBN 0-596-00727-2 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 13.02.2007. Формат 70x100 1/16 . Печать офсетная.
Объем 19,5 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

Языки программирования постоянно появляются и исчезают, и очень немногие из современных языков имеют более чем 10-летнюю историю. Среди долгожителей можно назвать КОБОЛ, который до сих пор довольно широко используется в мэйнфреймовых средах, и C, по-прежнему весьма популярный при разработке операционных систем, серверов и встроенных систем. В области баз данных это SQL, корни которого уходят в далекие 1970-е.

SQL – язык для формирования, манипулирования и извлечения данных из реляционной БД. Одна из причин популярности реляционных БД в том, что, будучи правильно спроектированными, они могут оперировать гигантскими объемами данных. В работе с большими наборами данных SQL напоминает современную цифровую фотокамеру с мощным объективом: он позволяет просматривать большие объемы данных или перейти к «крупному плану», т. е. сфокусироваться на отдельных строках (подвластно и все, что между этими крайностями). Другие СУБД дают сбой при мощных нагрузках, потому что их фокус слишком узок (увеличительные линзы достигают своего максимума). Именно по этой причине все попытки низвергнуть реляционные БД и SQL оканчиваются неудачей. Поэтому, даже несмотря на то, что SQL – старый язык, похоже, его ждет еще очень долгая жизнь и блестящее будущее.

Зачем изучать SQL?

Если вы собираетесь работать с реляционными БД – писать приложения, или выполнять задачи по администрированию, или формировать отчеты, – вам понадобится знать, как взаимодействовать с данными БД. Даже при использовании инструмента, генерирующего SQL (например, инструмента создания отчетов), могут возникнуть ситуации, в которых понадобится обойти автоматические возможности и создавать собственные SQL-выражения.

Дополнительное преимущество изучения SQL в том, что вы быстрее рассмотрите и поймете структуры данных, применяемые для хранения информации о вашей организации. Почувствовав себя уверенно со своей БД, вы сможете вносить предложения по изменению или дополнению ее схемы.

Почему именно эта книга?

Язык SQL включает несколько категорий. Выражения, с помощью которых создаются объекты БД (таблицы, индексы, ограничения и т. д.), называют *SQL-выражениями управления схемой данных* (*schema statements*). Выражения, предназначенные для создания, манипулирования и извлечения данных, хранящихся в БД, называют *SQL-выражениями для работы с данными* (*data statements*). Если вы администратор, то будете использовать и те и другие SQL-выражения. Если вы программист или составитель отчетов, то сможете (или вам будет *позволено*) использовать только SQL-выражения для работы с данными. Хотя в этой книге встречается много SQL-выражений управления схемой, основное внимание в ней уделено возможностям программирования.

Поскольку команд немного, SQL-выражения для работы с данными кажутся простыми. По-моему, многие из имеющихся книг по SQL только усиливают это впечатление, давая лишь поверхностный обзор того, что можно делать с помощью этого языка. Однако если вы собираетесь работать с SQL, вам следует полностью понимать все его возможности и то, как сочетать их для получения мощных результатов. На мой взгляд, эта книга – единственная, где язык SQL описан подробно, и при этом она не является «кирпичом» (вам знакомы эти «полные руководства» по 1250 страниц, пылящиеся у народа на полках).

Хотя примеры из книги подходят для MySQL, Oracle Database и SQL Server, мне пришлось отобрать один из этих продуктов, чтобы разместить БД для выполнения примеров и форматировать результирующие наборы, возвращаемые примерами запросов. Из этих трех я выбрал MySQL, потому что он свободно доступен, его легко установить и просто администрировать. Читателей, использующих другой сервер, прошу скачать и установить MySQL и загрузить предлагаемую БД, чтобы иметь возможность выполнять примеры и экспериментировать с данными.

Структура книги

Книга содержит 13 глав и 4 приложения:

- В главе 1 «Немного истории» рассматривается история компьютерных БД, включая возникновение реляционной модели и языка SQL.
- В главе 2 «Создание и заполнение базы данных» показывается, как создавать БД MySQL и таблицы, используемые в примерах к книге, и как заполнять таблицы данными.
- Глава 3 «Азбука запросов» знакомит с выражением `select` и представляет наиболее распространенные блоки (clauses): `select`, `from`, `where`.

- Глава 4 «Фильтрация» представляет разные типы условий, которые могут использоваться в блоке `where` выражений `select`, `update` и `delete`.
- В главе 5 «Запрос к нескольким таблицам» показывается, как запросы могут работать с несколькими таблицами посредством соединений таблиц.
- Глава 6 «Работа с множествами» – все о множествах данных и о том, как они могут взаимодействовать внутри запросов.
- Глава 7 «Создание, преобразование и работа с данными» представляет несколько встроенных функций, используемых для манипулирования или преобразования данных.
- В главе 8 «Группировка и агрегаты» показывается, как можно агрегировать данные.
- Глава 9 «Подзапросы» представляет подзапрос (мой любимый прием) и показывает, как применяются подзапросы.
- Глава 10 «И снова соединения» продолжает рассматривать различные типы соединений таблиц.
- В главе 11 «Условная логика» рассматривается использование условной логики (т. е. `if-then-else`) в выражениях `select`, `insert`, `update` и `delete`.
- Глава 12 «Транзакции» знакомит с транзакциями и их использованием.
- В главе 13 «Индексы и ограничения» исследуются индексы и ограничения.
- Приложение A «ER-диаграмма примера базы данных» содержит схему базы данных, используемой для всех примеров книги.
- Приложение B «MySQL-расширения языка SQL» представляет некоторые интересные возможности реализации SQL – MySQL, не входящие в стандарт ANSI.
- Приложение C «Решения к упражнениям» содержит решения упражнений, приводимых в главах.
- Приложение D «Дополнительные источники» подсказывает, куда можно обратиться, чтобы получить более глубокие навыки.

Условные обозначения, используемые в книге

В книге используются следующие типографские обозначения:

Курсив

Используется для имен файлов, имен каталогов и URL-адресов. Также используется для выделения и при первом упоминании технического термина.

Моноширинный шрифт

Используется для примеров кода и обозначения ключевых слов SQL в тексте.

Моноширинный курсив

Используется для обозначения пользовательских терминов.

ВЕРХНИЙ РЕГИСТР

Используется для обозначения ключевых слов SQL в примерах кода.

Моноширинный полужирный шрифт

Выделяет ввод пользователя в примерах с интерактивным взаимодействием. Также выделяет элементы кода, на которые следует обратить особое внимание.

Так выделяются советы, рекомендации или общие примечания. Например, с помощью примечаний я обращаю ваше внимание на полезные новые возможности Oracle9*i*.

Обозначает предупреждение или предостережение. Так я предупреждаю, например, о том, что неаккуратное применение некоего блока SQL может иметь неожиданные последствия.

Контакты

Пожалуйста, присылайте комментарии и вопросы по данной книге издателю:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (в Соединенных Штатах или Канаде)
(707) 829-0515 (международный или местный)
(707) 829-0104 (факс)

Для этой книги издательство O'Reilly поддерживает веб-страницу, на которой приведены список опечаток, примеры и вся дополнительная информация. Эту страницу можно найти по адресу:

http://www.oreilly.com/catalog/learningsql

Чтобы прокомментировать или задать технические вопросы по этой книге, присылайте электронные сообщения по адресу:

bookquestions@oreilly.com

Более подробная информация о книгах издательства O'Reilly, конференциях, центрах ресурсов и портале O'Reilly Network представлена на веб-сайте:

http://www.oreilly.com

Использование примеров кода

Цель этой книги – помочь вам выполнить работу. Код, представленный в книге, в общем случае можно использовать в программах и документации. На воспроизведение небольших фрагментов кода в вашей программе разрешение не требуется. Для продажи или распространения CD-ROM с примерами из книг O'Reilly разрешение *необходимо*. Если, отвечая на вопросы, вы ссылаетесь на книгу и цитируете пример кода, разрешение не требуется. Для включения существенного объема кода примеров из этой книги в документацию собственного продукта разрешение *необходимо*.

Мы признательны за указание авторства, но не требуем этого. Обычно указание источника включает название, автора, издателя и ISBN. Например: «Learning SQL by Alan Beaulieu. Copyright 2005 O'Reilly Media, Inc., 0-596-00727-2.»

Если вы сомневаетесь в корректности использования вами примеров кода, обратитесь за разъяснениями по адресу *permissions@oreilly.com*.

Safari Enabled

Если на обложке книги есть пиктограмма «Safari® Enabled», это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

Safari предлагает намного лучшее решение, чем электронные книги. Это виртуальная библиотека, позволяющая без труда находить тысячи лучших технических книг, вырезать и вставлять примеры кода, загружать главы и находить быстрые ответы, когда требуется наиболее верная и свежая информация. Она свободно доступна по адресу *http://safari.oreilly.com*.

Благодарности

Книга – живой организм, и то, что сейчас перед вами, далеко от моих первоначальных набросков. Эта метаморфоза произошла во многом благодаря моему редактору Джонатану Геннику (Jonathan Gennick). Спасибо тебе за помощь на каждом этапе проекта – как за твою редакторскую доблесть, так и за экспертную поддержку в вопросах, связанных с языком SQL. Еще я хотел бы поблагодарить трех моих технических рецензентов: Питера Гулутзана (Peter Gulutzan), Джозефа Молинаро (Joseph Molinaro) и Джеффа Кокса (Jeff Cox), побудивших меня сделать эту книгу и технически насыщенной, и подходящей для читателей, не знакомых с SQL. Также огромная благодарность многим сотрудникам O'Reilly Media, помогавшим воплотить эту книгу в реальность, в том числе корректору Мэтт Хатчинсон (Matt Hutchinson), дизайнеру обложки Элли Волкхаузен (Ellie Volckhausen) и художнику-оформителю Робу Романо (Rob Romano).

1

Немного истории

Прежде чем засучить рукава и приступить к работе, полезно представить некоторые базовые концепции и заглянуть в историю компьютеризированного хранения и извлечения данных.

Введение в базы данных

База данных – это всего лишь набор взаимосвязанных данных. Например, телефонный справочник – это база данных имен, телефонных номеров и адресов всех людей, проживающих в определенной местности. Будучи, несомненно, широко и часто используемой базой данных, телефонный справочник не лишен следующих недостатков:

- Поиск конкретного телефонного номера занимает много времени, особенно если в телефонном справочнике очень много записей.
- Телефонный справочник проиндексирован только по именам/фамилиям, поэтому для поиска абонента по определенному адресу такая база данных практически не используется, хотя теоретически это и возможно.
- С момента публикации телефонного справочника адекватность представленной в нем информации постепенно снижается, поскольку люди уезжают и приезжают из этой местности, меняют свои телефонные номера или переезжают по другому адресу в той же местности.

Недостатками, присущими телефонным справочникам, обладает любая другая некомпьютеризированная система хранения данных, например регистратура поликлиники, хранящая медицинские карты пациентов. Из-за громоздкости бумажных баз данных одними из первых компьютерных приложений были разработаны *системы баз данных* (*database systems*) – средства компьютеризированного хранения и извлечения данных. Поскольку система БД хранит информацию в элек-

тронном виде, а не на бумаге, она может быстрее извлекать данные, индексировать их разными способами и поставлять своим пользователям самую свежую информацию.

В первых системах БД информация хранилась на магнитных лентах. Количество лент во много раз превышало количество устройств считывания, поэтому техническим специалистам приходилось постоянно менять ленты, чтобы предоставить ту или иную запрашиваемую информацию. Поскольку объем памяти у компьютеров той эпохи был невелик, при многократных запросах одних и тех же данных обычно требовалось многократное считывание этих данных с магнитной ленты. Хотя эти системы БД и были существенным усовершенствованием по сравнению с бумажными БД, им было еще далеко до возможностей современных технологий. (Современные системы БД могут работать с терабайтами данных, распределенными по многим дискам с быстрым доступом, и держать в быстродействующей памяти десятки гигабайт этих данных; впрочем, я немного забегаю вперед.)

Нереляционные системы баз данных

Первые несколько десятилетий данные в компьютеризированных системах БД хранились и представлялись по-разному. Например, в *иерархической системе баз данных* (*hierarchical database system*) данные были представлены в виде одной или нескольких древовидных структур. На рис. 1.1 показано, как с помощью древовидных структур можно организовать данные банковских счетов Джорджа Блейка (George Blake) и Сью Смит (Sue Smith).

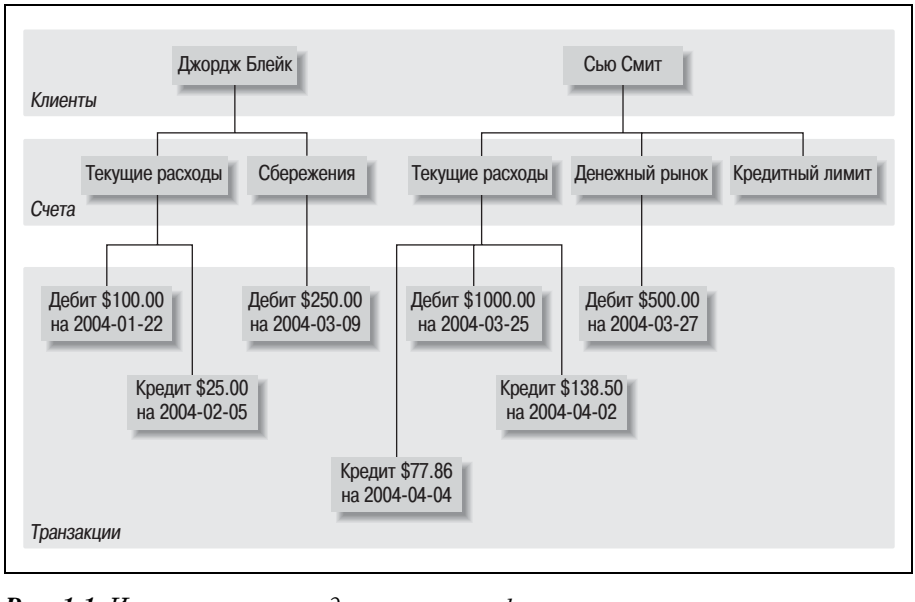


Рис. 1.1. *Иерархическое представление информации по счетам*

И у Джорджа, и у Сью есть собственное дерево, включающее их счета и транзакции, производимые по этим счетам. Иерархическая система базы данных предоставляет средства для нахождения дерева конкретного клиента и последующего обхода этого дерева в поисках нужных счетов и/или транзакций. У каждого узла дерева может быть ни одного или один родитель и ни одного, один или много дочерних узлов. Такую конфигурацию называют *иерархией с одним родителем* (*single-parent hierarchy*).

Другой распространенный подход, называемый *сетевой базой данных* (*network database system*), представляет собой наборы записей и наборы связей (links), определяющих отношения (relationships) между разными записями. На рис. 1.2 показано, как выглядели бы те же счета Джорджа и Сью в такой системе.

Чтобы найти транзакции, производимые по депозитному счету денежного рынка Сью, понадобилось бы сделать следующее:

1. Найти клиентскую запись Сью Смит.
2. Перейти по связи от клиентской записи Сью Смит к списку ее счетов.
3. Просматривать цепочку счетов до тех пор, пока не будет найден счет денежного рынка.

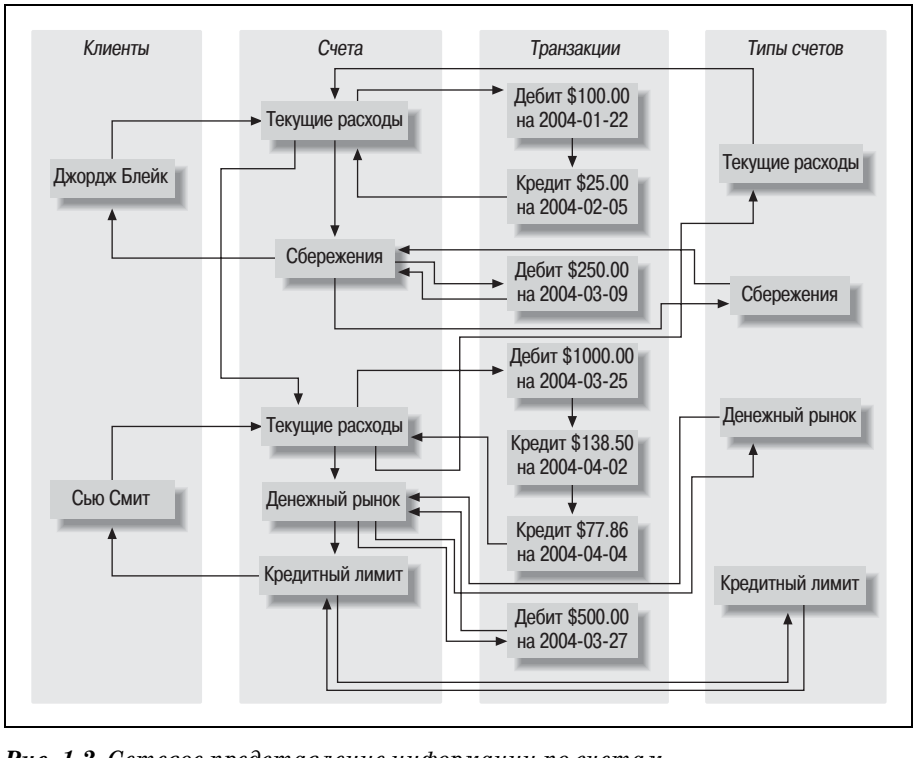


Рис. 1.2. *Сетевое представление информации по счетам*

4. Перейти по связи от записи денежного рынка к списку его транзакций.

Одну интересную особенность сетевых баз данных демонстрирует набор записей `product` (тип счета), на рис. 1.2 крайний справа. Обратите внимание, что каждая запись `product` (`Checking` (текущие расходы), `Savings` (сбережения) и т. д.) указывает на список записей `account` (счет), соответствующих этому типу счета. Поэтому доступ к записям `account` может быть осуществлен из нескольких мест (и через записи `customer`, и через записи `product`), что делает сетевую базу данных *иерархией с несколькими родителями* (*multiparent hierarchy*).

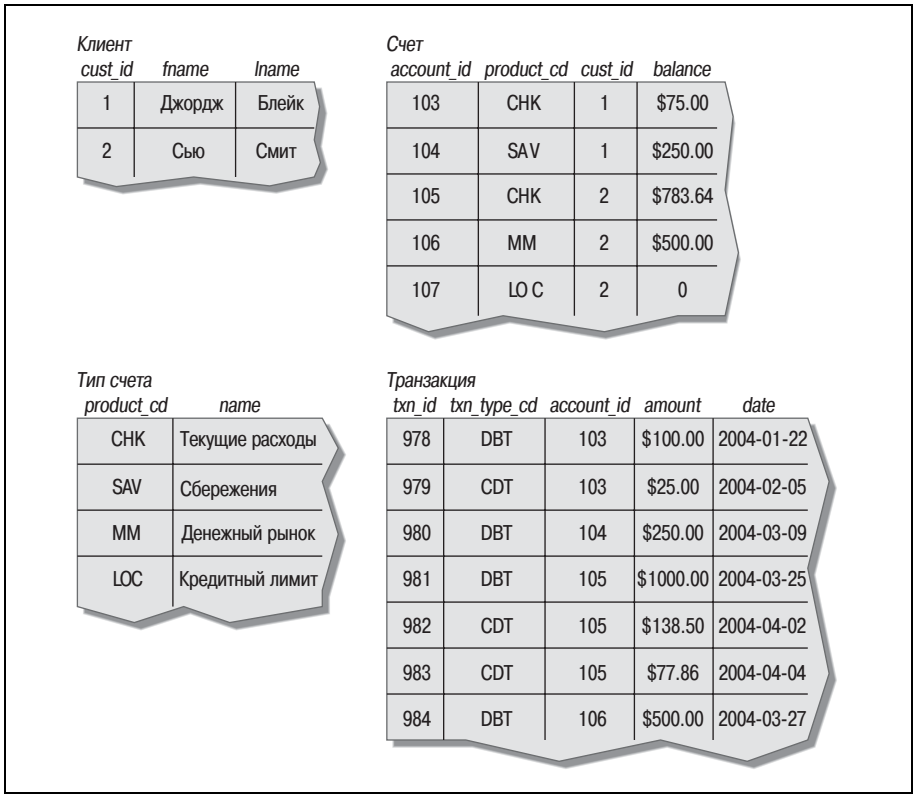
И иерархические, и сетевые системы баз данных ныне живы и здоровы, хотя преимущественно в мире мэйнфреймов. Кроме того, иерархические системы БД возродились в службах каталогов, таких как Active Directory компании Microsoft и Directory Server компании Netscape, а также с появлением XML (Extensible Markup Language, расширяемый язык разметки). Однако начиная с 1970-х годов все большую популярность приобретает новый способ представления данных, более строгий, но при этом более понятный и удобный.

Реляционная модель

В 1970 году сотрудник исследовательской лаборатории IBM доктор Е. Ф. Кодд (E. F. Codd) опубликовал статью под названием «A Relational Model of Data for Large Shared Data Banks» (Реляционная модель данных для больших банков данных коллективного пользования), в которой предложил представлять данные как наборы *таблиц*. Вместо указателей для навигации по взаимосвязанным сущностям используются избыточные данные, связывающие записи разных таблиц. На рис. 1.3 представлена информация счетов Джорджа и Сью в таком контексте.

На рис. 1.3 есть четыре таблицы, представляющие четыре обсуждаемые сущности: `customer`, `product`, `account` и `transaction` (транзакция). Посмотрев на таблицу `customer`, можно увидеть три *столбца*: `cust_id` (идентификационный номер клиента), `fname` (имя клиента) и `lname` (фамилия клиента). Ниже в таблице `customer` видим две *строки*: первая содержит данные Джорджа Блейка, вторая – данные Сью Смит. Максимально возможное количество столбцов в таблице отличается для разных серверов, но обычно это достаточно большое число, и с ним нет проблем (Microsoft SQL Server, например, допускает до 1024 столбцов в таблице). Число строк в таблице – это больше вопрос физических возможностей (т. е. определяется доступным дисковым пространством), чем ограничений серверов БД.

Каждая таблица реляционной базы данных включает информацию, уникально идентифицирующую строку этой таблицы (*первичный ключ* (*primary key*)), а также дополнительные данные, необходимые для полного описания сущности. Возвращаясь к таблице `customer`: в столбце `cust_id` каждому клиенту соответствует определенный номер. Напри-



Клиент

cust_id	fname	lname
1	Джордж	Блейк
2	Сью	Смит

Счет

account_id	product_cd	cust_id	balance
103	CHK	1	$75.00
104	SAV	1	$250.00
105	CHK	2	$783.64
106	MM	2	$500.00
107	LO C	2	0

Тип счета

product_cd	name
CHK	Текущие расходы
SAV	Сбережения
MM	Денежный рынок
LOC	Кредитный лимит

Транзакция

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	$100.00	2004-01-22
979	CDT	103	$25.00	2004-02-05
980	DBT	104	$250.00	2004-03-09
981	DBT	105	$1000.00	2004-03-25
982	CDT	105	$138.50	2004-04-02
983	CDT	105	$77.86	2004-04-04
984	DBT	106	$500.00	2004-03-27

Рис. 1.3. *Реляционное представление информации по счетам*

мер, Джорджа Блейка можно уникально идентифицировать с помощью клиентского идентификатора (ID №). Никогда никакому другому клиенту не будет присвоен такой же идентификатор, и этой информации достаточно, чтобы обнаружить данные Джорджа Блейка в таблице `customer`. Хотя в качестве первичного ключа можно было бы выбрать сочетание столбцов `fname` и `lname` (первичный ключ, состоящий из двух и более столбцов, называют *составным ключом* (*compound key*)), у двух и более человек, имеющих счета в банке, могут быть одинаковые имена и фамилии. Поэтому специально для первичных ключей в таблицу `customer` был включен столбец `cust_id`.

Некоторые из таблиц также содержат информацию, используемую для навигации к другой таблице. Например, в таблице `account` есть столбец `cust_id`, содержащий уникальный идентификатор клиента, открывшего счет, и столбец `product_cd`, содержащий уникальный идентификатор типа счета, которому будет соответствовать счет. Эти столбцы называют *внешними ключами* (*foreign keys*). Они служат той же цели, что и линии, соединяющие сущности в иерархической и сетевой версиях пред-

ставления информации по счетам. Однако, в отличие от жесткой структуры иерархической/сетевой моделей, реляционные таблицы можно использовать по-разному (даже так, как разработчики этой базы данных и не представляли себе).

Может показаться излишним хранить одни и те же данные в нескольких местах, но реляционная модель использует избыточность данных очень четко. Например, если таблица `account` включает столбец для уникального идентификатора клиента, открывшего счет, это правильно, а если включены также его имя и фамилия, то это неправильно. Например, если клиент изменяет имя, нужна уверенность, что его имя хранится только в одном месте базы данных. В противном случае данные могут быть изменены в одном месте, но не изменены в другом, что приведет к их недостоверности. Правильное решение – хранить эту информацию в таблице `customer`. В другие таблицы следует включить только `cust_id`. Также неправильно располагать в одном столбце несколько элементов данных, например в столбец `name` помещать имя и фамилию человека или в столбце `address` указывать улицу, город, страну и почтовый индекс. Процесс улучшения структуры базы данных с целью обеспечения хранения всех независимых элементов данных только в одном месте (за исключением внешних ключей) называют *нормализацией* (*normalization*).

Вернемся к четырем таблицам на рис. 1.3; на первый взгляд может быть непонятно, как использовать их для поиска транзакций Джорджа Блейка по его текущему счету. Во-первых, находим уникальный идентификатор Джорджа Блейка в таблице `customer`. Затем строку в таблице `account`, столбец `cust_id` которой содержит уникальный идентификатор Джорджа, а столбец `product_cd` соответствует строке таблицы `product`, столбец `name` которой содержит значение `"Checking"`. Наконец, в таблице `transaction` находим строки, столбец `account_id` которых соответствует уникальному идентификатору из таблицы `account`. Возможно, все это кажется сложным, но с помощью языка SQL может быть осуществлено одной-единственной командой, как вы вскоре увидите.

Немного терминологии

В предыдущих разделах были введены некоторые новые термины, поэтому приведем кое-какие формальные определения. В табл. 1.1 приведены термины, используемые в данной книге, и их определения.

Таблица 1.1. Термины и определения

Термин	Определение
Сущность (entity)	То, что представляет интерес для пользователей базы данных, например клиенты, запчасти, географическое положение и т. д.
Столбец (column)	Отдельный элемент данных, хранящийся в таблице.

Термин	Определение
Строка (row)	Набор столбцов, которые вместе полностью описывают сущность или некоторое действие, производимое над сущностью. Также называется *записью* (*record*).
Таблица (table)	Набор строк, хранящийся в памяти (непостоянная таблица) или на постоянном запоминающем устройстве (постоянная таблица).
Результирующий набор (result set)	Другое название непостоянной таблицы, обычно являющейся результатом SQL-запроса.
Первичный ключ (primary key)	Один или более столбцов, которые можно использовать как уникальный идентификатор для каждой строки таблицы.
Внешний ключ (foreign key)	Один или более столбцов, которые можно совместно использовать для идентификации одной строки другой таблицы.

Что такое SQL?

Помимо определения реляционной модели Кодд предложил язык для работы с данными в реляционных таблицах, названный DSL/Alpha. Вскоре после публикации статьи Кодда в IBM была организована группа для создания прототипа языка на базе его идей. Эта группа разработала упрощенную версию DSL/Alpha, которую назвали SQUARE. В результате усовершенствования SQUARE появился язык SEQUEL, который в конце концов получил имя SQL.

Сейчас SQL разменял четвертый десяток, претерпев за свой век множество изменений. В середине 1980-х Национальный институт стандартизации США (American National Standards Institute, ANSI) начал разрабатывать первый стандарт языка SQL, который был опубликован в 1986 г. Дальнейшие доработки были отражены в следующих версиях стандарта SQL (1989, 1992, 1999 и 2003 гг.). Наряду с усовершенствованием базового языка в SQL появились и новые возможности для обеспечения объектно-ориентированной функциональности.

SQL идет рука об руку с реляционной моделью, потому что результатом SQL-запроса является таблица (в данном контексте также называемая *результирующим набором*). Таким образом, в реляционной базе данных можно создать новую постоянную таблицу, просто сохранив результирующий набор запроса. Аналогично в качестве входных данных запрос может использовать как постоянные таблицы, так и результирующие наборы других запросов (подробно это будет рассмотрено в главе 9).

И последнее замечание: SQL не акроним (хотя многие настаивают, что это сокращение от Structured Query Language (Структурированный язык запросов)). Название этого языка произносится по буквам (т. е. «S», «Q», «L») или как «sequel» (сиквел).

Классы SQL-выражений

Язык SQL разбит на несколько отдельных частей. В данной книге будут рассмотрены: SQL-выражения управления схемой данных (SQL schema statements), предназначенные для определения структур данных, хранящихся в базе данных; SQL-выражения для работы с данными (SQL data statements), предназначенные для работы со структурами данных, ранее определенными с помощью SQL-выражений управления схемой; SQL-выражения управления транзакциями, предназначенные для начала, завершения и отката транзакций (рассматриваются в главе 12). Например, новая таблица базы данных создается с помощью SQL-выражения управления схемой `create table` (создать таблицу), а чтобы заполнить ее данными, потребуется SQL-выражение для работы с данными `insert` (вставить).

Чтобы дать представление об этих выражениях, приведем SQL-выражение управления схемой, создающее таблицу `corporation` (корпорация):

```
CREATE TABLE corporation
 (corp_id SMALLINT,
  name VARCHAR(30),
  CONSTRAINT pk_corporation PRIMARY KEY (corp_id)
 );
```

Это выражение создает таблицу с двумя столбцами, `corp_id` и `name`, где столбец `corp_id` определен как первичный ключ таблицы. Подробная информация о данном выражении, например доступные в MySQL типы данных, приводится в следующей главе. Теперь рассмотрим SQL-выражение для работы с данными, которое вставляет в таблицу `corporation` запись для корпорации Acme Paper Corporation:

```
INSERT INTO corporation (corp_id, name)
VALUES (27, 'Acme Paper Corporation');
```

Это выражение добавляет в таблицу `corporation` строку со значением `27` в столбце `corp_id` и значением `Acme Paper Corporation` в столбце `name`.

Наконец, приведем простое выражение `select` (выбрать) для извлечения только что созданных данных:

```
mysql> SELECT name
    -> FROM corporation
    -> WHERE corp_id = 27;
+------------------------+
| name                   |
+------------------------+
| Acme Paper Corporation |
+------------------------+
```

Все элементы БД, созданные посредством SQL-выражений управления схемой, хранятся в специальном наборе таблиц, который называется *словарем данных* (*data dictionary*). Все эти «данные о базе данных» на-

зывают *метаданными* (*metadata*). К таблицам словаря данных можно делать запросы с помощью оператора `select`, в точности как к созданным вами таблицам. Таким образом, текущие структуры данных, развернутые в БД во время выполнения, становятся доступными. Например, если требуется создать отчет о новых счетах, открытых за последний месяц, можно жестко закодировать известные на момент написания отчета имена столбцов таблицы `account` либо сделать запрос к словарю данных, получить текущий набор столбцов и динамически генерировать отчет при каждом выполнении.

Данная книга посвящена главным образом той части языка для работы с данными, к которой относятся команды `select`, `update` (обновить), `insert` и `delete` (удалить). SQL-выражения управления схемой рассмотрены в главе 2, где создается БД, используемая в примерах данной книги. Вообще говоря, SQL-выражения управления схемой не требуют особого внимания, за исключением их синтаксиса, тогда как у SQL-выражений для работы с данными (хотя их и немного) есть масса нюансов, нуждающихся в подробном изучении. Поэтому большинство глав данной книги посвящены SQL-выражениям для работы с данными.

SQL: непроцедурный язык

Если в прошлом вам приходилось работать с языками программирования, вы привыкли к описанию переменных и структур данных, использованию условной логики (if-then-else), циклическим конструкциям (do while ... end) и разделению кода на небольшие многократно используемые части (объекты, функции, процедуры). Код передается компилятору, и результирующий исполняемый код делает в точности (ну, не всегда *в точности*) то, что вы запрограммировали. С каким бы языком программирования ни работали, Java, C#, C, Visual Basic или любым другим *процедурным* языком, вы полностью управляете действиями программы. С SQL, однако, понадобится отказаться от привычного контроля над выполнением, потому что SQL-выражения определяют необходимые входные и выходные данные, а способ выполнения выражения зависит от компонента механизма СУБД (database engine), называемого *оптимизатором* (*optimizer*). Работа оптимизатора заключается в том, чтобы рассмотреть SQL-выражение и с учетом конфигурации таблиц и доступных индексов принять решение о самом эффективном пути выполнения запроса (ну, не всегда *самом* эффективном). Большинство СУБД позволяют программисту влиять на решения оптимизатора с помощью *подсказок оптимизатору* (*optimizer hints*), например предложений по использованию конкретного индекса. Однако большинство пользователей SQL никогда не доберется до этого уровня сложности и будет оставлять подобные тонкости администраторам БД или специалистам по вопросам производительности.

Следовательно, с SQL писать полные приложения не получится. Если требуется создать что-то сложнее простого сценария для работы с оп-

ределенными данными, понадобится интегрировать SQL со своим любимым языком программирования. Некоторые производители баз данных сделали это за вас, например Oracle с языком PL/SQL или Microsoft с TransactSQL. Благодаря этим языкам SQL-выражения для работы с данными являются частью грамматики языка программирования, что позволяет свободно интегрировать запросы к БД с процедурными командами. Однако при использовании не характерного для БД языка, такого как Java, для выполнения SQL-выражений понадобится специальное средство. Некоторые из этих программных средств предоставляются производителями БД, тогда как другие создаются сторонними производителями или разработчиками ПО с открытым исходным кодом. В табл. 1.2 показаны некоторые доступные варианты интегрирования SQL в конкретные языки программирования.

Таблица 1.2. Средства интегрирования SQL

Язык программирования	Программное средство
Java	JDBC (Java Database Connectivity) (JavaSoft)
C++	RogueWave SourcePro DB (инструмент сторонних производителей для соединения с БД Oracle, SQL Server, MySQL, Informix, DB2, Sybase и PostgreSQL)
C/C++	Pro*C (Oracle) MySQL C API (с открытым исходным кодом) DB2 Call Level Interface (IBM)
C#	ADO.NET (Microsoft)
VisualBasic	ADO.NET (Microsoft)

Если требуется только интерактивное выполнение SQL-команд, каждый производитель БД обеспечивает как минимум простой инструмент передачи SQL-команд механизму СУБД и просмотра результатов. Большинство производителей предлагает также графический инструмент, в одном окне которого вводятся SQL-команды, а в другом выводятся результаты их выполнения. Поскольку примеры данной книги работают с базой данных MySQL, для запуска примеров и форматирования результатов я буду использовать утилиту командной строки *mysql*.

Примеры SQL

Ранее в этой главе я обещал показать SQL-выражение, возвращающее все транзакции текущего счета Джорджа Блейка. Не будем тянуть, вот оно:

```
SELECT t.txn_id, t.txn_type_cd, t.date, t.amount
FROM customer c INNER JOIN account a ON c.cust_id = a.cust_id
  INNER JOIN product p ON p.product_cd = a.product_cd
  INNER JOIN transaction t ON t.account_id = a.account_id
```

```
WHERE c.fname = 'George' and c.lname = 'Blake'
  AND p.name = 'checking';
```

Без лишних на этом этапе подробностей: данный запрос идентифицирует в таблице account строку Джорджа Блейка, а в таблице product – строку с типом счета 'checking' (текущие расходы), в таблице account находит строку, соответствующую данной комбинации «клиент/тип счета», и возвращает четыре столбца таблицы transaction для всех транзакций по этому счету. Все концепции, присутствующие в данном запросе (и многие другие), будут рассмотрены в следующих главах; здесь мне просто хотелось показать, как выглядел бы запрос.

Предыдущий запрос содержит три разных *блока* (*clauses*): select, from и where. Практически каждый сформированный вами запрос будет включать, по крайней мере, эти три блока, хотя есть и другие блоки, применяемые для более сложных целей. Роль каждого из этих трех блоков можно продемонстрировать следующим образом:

```
SELECT /* одна или более сущностей */ ...
FROM   /* одно или более мест */ ...
WHERE  /* удовлетворяется одно или более условий */ ...
```

Большинство реализаций SQL воспринимают текст, расположенный между тегами /* и */, как комментарии.

Обычно первая задача при создании запроса – определить, какая таблица или таблицы понадобятся, а затем добавить их в блок from. Далее необходимо отсеять данные этих таблиц, которые не помогут ответить на запрос. Для этого в блок where вводятся условия. Наконец, принимается решение о том, какие столбцы разных таблиц требуется извлечь, и они добавляются в блок select. Вот простой пример поиска всех клиентов по фамилии Smith (Смит):

```
SELECT cust_id, fname
FROM customer
WHERE lname = 'Smith'
```

Этот запрос выполняет поиск в таблице customer всех строк, столбец lname которых соответствует строке 'Smith', и возвращает столбцы cust_id и fname этих строк.

Кроме создания запросов к БД вам, скорее всего, придется заполнять и изменять данные БД. Вот простой пример добавления новой строки в таблицу product:

```
INSERT INTO product (product_cd, name)
VALUES ('CD', 'Certificate of Depasit')
```

Ой, кажется, в слове «Deposit» ошибка! Никаких проблем. Это можно исправить с помощью выражения update:

```
UPDATE product
```

```
SET name = 'Certificate of Deposit'
WHERE product_cd = 'CD';
```

Обратите внимание, что в выражении `update` тоже есть блок `where`, как и в выражении `select`, потому что `update` должно отобрать строки, подлежащие изменению. В данном случае задано, что должны быть изменены только те строки, столбцы `product_cd` которых соответствуют строке `'CD'`. Поскольку столбец `product_cd` является первичным ключом таблицы `product`, следует ожидать, что выражение `update` изменит только одну строку (или ни одной, если такого значения в таблице нет). При выполнении любого SQL-выражения для работы с данными механизм СУБД выводит отчет с указанием того, сколько строк было подвержено его воздействию. Если используется интерактивный инструмент, например уже упомянутый инструмент командной строки *mysql*, будет получено сообщение о том, сколько строк было:

- возвращено выражением `select`;
- создано выражением `insert`;
- изменено выражением `update`;
- удалено выражением `delete`.

Если используется процедурный язык с одним из уже упомянутых программных средств, то после выполнения SQL-выражения для работы с данными это средство включит вызов функции запроса этой информации. В общем, не мешает проверять эти данные, чтобы убедиться, что выражение не сделало ничего непредвиденного (например, если забыть включить в выражение `delete` блок `where`, будут удалены все строки таблицы!).

Что такое MySQL?

Реляционные базы данных продаются уже более двух десятилетий. К самым зрелым и популярным продуктам относятся:

- Oracle Database от Oracle Corporation
- SQL Server от Microsoft
- DB2 Universal Database от IBM
- Sybase Adaptive Server от Sybase
- Informix Dynamic Server от IBM

Все эти серверы БД делают примерно одно и то же, хотя некоторые лучше оснащены для работы с очень большими или высокопроизводительными БД. Другие лучше ведут себя при работе с объектами, или очень большими файлами, или XML-документами и т. д. Кроме того, очень хорошо, что все эти серверы совместимы с последним стандартом ANSI SQL. Это положительный момент, и я обязательно покажу, как писать SQL-выражения, которые будут выполняться на любой из этих платформ (с небольшими изменениями или вообще без них).

Наряду с этим последние пять лет в сообществе сторонников открытого исходного кода наблюдалась активная деятельность по созданию жизнеспособной альтернативы коммерческим серверам БД. Два наиболее распространенных сервера БД с открытым исходным кодом – PostgreSQL и MySQL. Веб-сайт MySQL (*http://www.mysql.com*) в настоящее время заявляет о более чем 6 000 000 установок, их сервер доступен бесплатно, и я убедился, что скачать и установить его чрезвычайно просто. Поэтому я решил, что все примеры для данной книги будут выполняться на БД MySQL (версии 4.1.11). Для форматирования результатов запросов будет использоваться инструмент командной строки *mysql*. Даже если вы уже работаете с другим сервером и вообще не планируете использовать MySQL, я рекомендую установить последнюю версию сервера MySQL, загрузить схему и данные примера и экспериментировать с примерами этой книги.

Однако помните, что:

> Эта книга не о реализации SQL в MySQL.

Скорее, данная книга создана, чтобы обучить читателя создавать SQL-выражения, которые будут выполняться на MySQL и последних версиях Oracle Database, Sybase Adaptive Server и SQL Server с небольшими изменениями или вообще без них. Возможно, при использовании одного из упомянутых серверов IBM хлопот у вас будет чуть больше.

Чтобы по возможности сохранить код из данной книги платформонезависимым, я воздержусь от демонстрации некоторых интересных вещей, реализованных в языке SQL для MySQL и не осуществимых в других реализациях БД. Но для читателей, планирующих продолжать работу с MySQL, некоторые из этих возможностей рассмотрены в приложении B.

Дополнительные источники

Общая цель следующих четырех глав – представить SQL-выражения для работы с данными, уделив при этом особое внимание трем основным блокам выражения `select`. Кроме того, приводится множество примеров, использующих банковскую схему (она представлена в следующей главе и задействована во всех примерах данной книги). Надеюсь, что постоянное использование одной и той же БД позволит читателю вникать в суть примера, не тратя время на изучение применяемых таблиц.

Твердо усвоив основы, с помощью оставшихся глав вы изучите дополнительные концепции, по большей части не зависимые друг от друга. Поэтому, столкнувшись с какими-либо трудностями, всегда можно двинуться дальше, а позже перечитать главу. Прочитав книгу и проработав все примеры, вы уверенно пойдете к вершинам мастерства SQL.

Вот несколько заслуживающих внимания источников для читателей, желающих узнать больше о реляционных БД, истории компьютеризированных систем управления БД или языке SQL:

- К. Дж. Дейт (C. J. Date) «Database in Depth: Relational Theory for Practitioners», O'Reilly.
- К. Дж. Дейт «An Introduction to Database Systems, Eighth Edition», Addison Wesley.[1]
- К. Дж. Дейт «The Database Relational Model: A Retrospective Review and Analysis: A Historical Account and Assessment of E. F. Codd's Contribution to the Field of Database Technology», Addison Wesley.
- *http://en.wikipedia.org/wiki/Database_management_system*
- *http://www.mcjones.org/System_R/*

[1] К. Дейт «Введение в системы баз данных», 8-е издание, Вильямс, 2005.

2

Создание и заполнение базы данных

В этой главе представлена информация, необходимая для создания вашей первой БД, таблиц и ассоциированных данных, используемых в примерах книги. Также рассказывается о различных типах данных и об их применении при создании таблиц. Поскольку примеры книги выполняются на СУБД MySQL, здесь наблюдается небольшое смещение акцентов представляемого материала в сторону возможностей и синтаксиса MySQL, но большинство концепций применимы к любому серверу.

Создание базы данных MySQL

Если в вашем распоряжении уже есть СУБД MySQL, можно выполнять приведенные ниже инструкции, начиная с п. 8. Но не забывайте, что эта книга ориентирована на MySQL версии 4.1.11 или более поздних, поэтому если вы используете более раннюю версию, скорее всего, не помешает обновить ее или установить другой сервер.

Следующие инструкции отражают минимальный набор действий, необходимых для установки сервера MySQL на компьютере, работающем под управлением Windows, создания базы данных и загрузки тестовых данных для этой книги:

1. Скачайте MySQL Database Server (версии 4.1.11 или более поздней) с *http://dev.mysql.com*. Если сервер планируется использовать только для обучения, скачайте Essentials Package (Основной пакет), включающий только широко используемые инструменты, а не Complete Package (Полный пакет).
2. Двойным щелчком по загруженному файлу запустите процесс установки.

3. Установите сервер, используя вариант «typical install» (обычная установка). Установка должна пройти быстро и безболезненно, но не стесняйтесь обращаться к онлайновому руководству по установке (*http://dev.mysql.com/doc/mysql/en/Installing.html*).
4. По завершении установки, перед тем как нажать кнопку завершения, убедитесь, что флажок Configure the MySQL Server now (Конфигурировать сервер MySQL сейчас) установлен. Это нужно, чтобы запустился Configuration Wizard (Мастер конфигурации).
5. При запуске Configuration Wizard выберите переключатель Standard Configuration (Стандартная конфигурация) и затем установите флажки Install as Windows Service (Установить как службу Windows) и Include Bin Directory in Windows Path (Включить каталог Bin в путь поиска Windows).
6. Во время конфигурирования вам будет предложено выбрать пароль для привилегированного пользователя root. Не забудьте записать пароль, он понадобится позже.
7. Откройте консоль (с помощью Start→Run→Command (Пуск→Выполнить→Command)) и из консоли зарегистрируйтесь как привилегированный пользователь с помощью команды `mysql -u root -p`. Вам будет предложено ввести пароль, после этого появится подсказка `mysql>`.
8. Создайте нового пользователя базы данных. Я создал пользователя lrngsql с помощью команды `grant all privileges on *.* to 'lrngsql'@'localhost' identified by 'xxxxx';` (замените *xxxxx* паролем, который выбрали для этого пользователя).
9. Завершите сеанс с помощью команды `quit;` (выйти) и зарегистрируйтесь из консоли как новый пользователь посредством команды `mysql -u lrngsql -p`.
10. Создайте базу данных. Я создал БД «bank» (банк) с помощью выражения `create database bank;`.
11. Выберите новую БД с помощью выражения `use bank;`.
12. Скачайте тестовые данные для этой книги. Файл можно найти на сайте *learningsql* в разделе *Examples* (примеры) для данной книги.
13. Из инструмента командной строки *mysql* с помощью команды `source` (источник) загрузите данные из закачанного файла, например `source c:\tmp\learning_sql.sql`. Вместо пути `c:\tmp\` укажите каталог, в котором находится сценарий с тестовыми данными.

Теперь у вас должна быть рабочая БД, заполненная всеми данными, необходимыми для примеров данной книги.

Инструмент командной строки mysql

При вызове инструмента командной строки *mysql* можно задать имя пользователя и используемую БД:

```
mysql -u lrngsql -p bank
```

Будет запрошен ваш пароль, и затем появится приглашение `mysql>`, с помощью которого вы сможете создавать SQL-выражения и просматривать результаты их выполнения. Например, чтобы узнать текущие дату и время, можно выполнить следующий запрос:

```
mysql> SELECT now( );
+---------------------+
| now( )              |
+---------------------+
| 2005-05-06 16:48:46 |
+---------------------+
1 row in set (0.01 sec)
```

Функция `now( )` – это встроенная функция MySQL, возвращающая текущие дату и время. Как видите, инструмент командной строки *mysql* форматирует результаты запросов, помещая их в прямоугольник, очерченный символами `+`, `-` и `|`. Выведя все результаты (в данном случае это всего одна строка), инструмент командной строки *mysql* покажет количество возвращенных строк и длительность выполнения выражения SQL.

Завершив работу с инструментом командной строки *mysql*, для возвращения в консоль просто введите `quit;` или `exit;`.

О пропущенном блоке from

При работе с некоторыми серверами БД нельзя создать запрос без блока `from` (из), в котором должна быть указана по крайней мере одна таблица. Oracle Database – именно такой сервер. Для тех случаев, когда требуется только вызвать функцию, Oracle предоставляет таблицу `dual` (двойственная), состоящую всего из одного столбца `dummy` (макет), который содержит всего одну строку данных. Для обеспечения совместимости с Oracle Database MySQL тоже предоставляет таблицу `dual`. Следовательно, предыдущий запрос текущих даты и времени можно было бы написать так:

```
mysql> SELECT now( )
       FROM dual;
+---------------------+
| now( )              |
+---------------------+
| 2005-05-06 16:48:46 |
+---------------------+
1 row in set (0.01 sec)
```

Если вы не работаете с Oracle и вам не нужна совместимость с этой СУБД, таблицу `dual` можно полностью игнорировать.

Типы данных MySQL

Вообще говоря, все популярные серверы БД обладают способностью хранить одни и те же типы данных, такие как строки, даты и числа. Обычно их различие заключается в возможности хранения специальных типов данных, например XML-документов, или очень больших текстов, или двоичных документов. Поскольку данная книга является введением в SQL и 98 % всех столбцов, которые вы когда-либо встретите, будут простыми типами данных, мы рассмотрим только символьные, числовые и временные типы данных.

Символьные данные

Символьные данные могут храниться как строки фиксированной или переменной длины. Разница заключается в том, что строки фиксированной длины справа дополняются пробелами, тогда как строки переменной длины – нет. При определении столбца символьного типа необходимо задать максимальный размер сохраняемой в нем строки. Например, если предполагается хранить строки длиной до 20 символов, можно использовать любое из этих описаний:

```
CHAR(20)     /* строка фиксированной длины */
VARCHAR(20)  /* строка переменной длины */
```

В настоящее время максимальная длина этого типа данных составляет 255 символов (хотя в будущих версиях будут допустимы более длинные строки). Для сохранения более длинных строк (таких как сообщения электронной почты, XML-документы и т. д.) используйте один из текстовых типов – `tinytext` (крошечный текст), `text` (текст), `mediumtext` (средний текст), `longtext` (длинный текст)), – рассматриваемых в данном разделе позже. В общем, тип `char` подходит для случая, когда в столбце предполагается хранить только строки одинаковой длины, например сокращенные названия государств, а тип `varchar` – для строк разной длины. Типы `char` и `varchar` одинаково применимы во всех основных серверах БД.

Когда речь идет о применении типа данных `varchar`, СУБД Oracle Database является исключением. Пользователи Oracle при описании символьных столбцов переменной длины должны применять тип `varchar2`.

Наборы символов

В языках, использующих латинский алфавит, например в английском, довольно мало символов, то есть каждый символ хранится как один байт. В других языках, таких как японский и корейский, много символов. Таким образом, в них для хранения одного символа требуется несколько байт. Поэтому такие наборы символов называют *многобайтовыми наборами символов* (*multibyte character sets*).