

Крис Файли

SQL

QUICK START

VISUAL QUICKSTART GUIDE

SQL

Chris Fehily



Peachpit Press

QUICK START

SQL

Крис Файли

УДК 004.43
ББК 32.973.26-018.1
Ф48

Ф48 Фийли К.

SQL: Пер. с англ. – М.: ДМК Пресс, 2003. – 456 с.: ил. (Серия «Quick Start»).

ISBN 5-94074-233-5

Книга посвящена языку программирования SQL, применяемому для работы с реляционными базами данных. Обсуждается версия языка ANSI SQL-92 (SQL2).

В настоящем издании рассказывается об использовании запросов SQL для решения соответствующих классов задач по выборке данных, их модификации или по работе с объектами структуры базы данных. Все конструкции подробно описываются и иллюстрируются большим количеством примеров. Кроме того, для каждого типа запросов рассматриваются отклонения от стандарта в реализации наиболее распространенных СУБД: MS Access, MS SQL Server, Oracle, MySQL и PostgreSQL.

Книга предназначена всем тем, кто желает самостоятельно изучить язык SQL или усовершенствовать свои знания по этой теме.

Authorized translation from the English language edition, entitled SQL: VISUAL QUICKSTART GUIDE, 1st Edition, 0321118030 by FEHILY, CHRIS, published by Pearson Education, Inc, publishing as Peachpit Press, Copyright © 2003.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc. RUSSIAN language edition published by DMK PRESS, Copyright © 2003.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 0-201-11803-0 (англ.)

ISBN 5-94074-233-5 (рус.)

© Peachpit Press, 2003

© Перевод на русский язык, оформление
ДМК Пресс, 2003

СОДЕРЖАНИЕ

Введение	11
Глава 1. Основные характеристики СУБД	24
Выполнение программ SQL	25
Microsoft Access	27
Microsoft SQL Server	30
Oracle	33
MySQL	36
PostgreSQL	38
Глава 2. Реляционная модель	41
Таблицы, столбцы и строки	43
Первичные ключи	47
Внешние ключи	51
Связи	54
Нормализация	57
Наша типовая база данных	63
Глава 3. Основы SQL	69
Синтаксис SQL	69
Типы данных	75
Строковые типы данных	77
Битовые типы данных	79
Точные числовые типы данных	80
Действительные числовые типы данных	82
Календарные типы данных	84
Интервальные типы данных	88
Значение null	90

Глава 4. Выбор данных из произвольной таблицы	92
Выбор столбцов с помощью предложений SELECT и FROM	93
Создание псевдонимов столбцов с помощью предложения AS	96
Удаление повторяющихся строк с помощью ключевого слова DISTINCT	99
Сортировка строк с помощью предложения ORDER BY	101
Фильтрация строк с помощью предложения WHERE	109
Комбинирование условий с помощью операторов AND, OR и NOT	114
Сравнение по шаблону оператором LIKE	122
Сравнение с диапазоном с помощью оператора BETWEEN	128
Фильтрация с помощью оператора IN	131
Проверка на значение null с помощью оператора IS NULL	134
Глава 5. Операторы и функции	137
Создание производных столбцов	138
Арифметические операции	140
Определение последовательности вычисления	143
Объединение строк с помощью оператора	145
Выбор произвольной подстроки с помощью функции SUBSTRING()	149
Переключение регистра символов строки с использованием функций UPPER() и LOWER()	153
Удаление пробелов с помощью функции TRIM()	155
Определение длины произвольной строки с помощью функции CHARACTER_LENGTH()	159
Поиск подстроки с использованием функции POSITION()	161
Операции с данными даты и времени	165
Извлечение значений текущих даты и времени	169
Отображение информации о пользователе	171
Преобразование типов данных с помощью функции CAST()	173
Вычисление условных значений с помощью выражения CASE	179
Проверка на значения null с использованием функции COALESCE()	184
Сравнение выражений с помощью функции NULLIF()	185
Глава 6. Суммирование и группировка данных	188
Использование агрегатных функций	189
Поиск минимума посредством функции MIN()	192
Поиск максимума с использованием функции MAX()	194
Вычисление суммы с помощью функции SUM()	196
Порядок расчета среднего значения с помощью функции AVG()	198
Подсчет строк с помощью функции COUNT()	200

Исключение повторных значений с помощью предложения DISTINCT	202
Группирование строк с использованием предложения GROUP BY	206
Фильтрация групп с помощью предложения HAVING	215

Глава 7. Выбор данных из нескольких таблиц

219

Уточнение имен столбцов	220
Создание псевдонимов таблиц с помощью предложения AS	222
Использование объединений	224
Создание объединений с помощью синтаксиса JOIN и WHERE	228
Создание произвольного перекрестного объединения с использованием предложения CROSS JOIN	233
Создание произвольного естественного объединения с использованием предложения NATURAL JOIN	236
Создание внутреннего объединения с помощью команды INNER JOIN	241
Создание внешних объединений с помощью команды OUTER JOIN	265
Создание самообъединения	279
Комбинирование строк с помощью оператора UNION	286
Поиск общих строк с помощью команды INTERSECT	295
Поиск разных строк с помощью команды EXCEPT	297

Глава 8. Подзапросы

299

Принципы работы с подзапросами	300
Структура подзапросов	302
Подзапросы и объединения	303
Простые и сложные запросы	307
Определение названий столбцов в подзапросах	313
Значения null в подзапросах	314
Использование подзапросов в качестве выражений в списке заголовков столбцов	316
Сравнение значений, возвращаемых подзапросом, с использованием операторов сравнения	320
Проверка на входжение во множество с помощью оператора IN ...	325
Сравнение всех значений запроса с помощью ключевого слова ALL	333
Сравнение некоторых значений запроса с помощью ключевого слова ANY	337
Проверка наличия выборки с помощью оператора EXISTS	341
Сравнение эквивалентных запросов	347

Глава 9. Добавление, обновление и удаление строк	348
Отображение названий столбцов в таблице	349
Вставка строк с помощью команды INSERT	352
Изменение строк с помощью команды UPDATE	358
Удаление строк с помощью команды DELETE	363
Глава 10. Создание, изменение и удаление таблиц	367
Порядок создания таблиц	368
Основные принципы работы с ограничениями	369
Создание новой таблицы с помощью команды CREATE TABLE	371
Запрет значения null с помощью ограничения NOT NULL	373
Присвоение значения по умолчанию с помощью ограничения DEFAULT	376
Задание первичного ключа с помощью ограничения PRIMARY KEY	379
Задание внешнего ключа с помощью ограничения FOREIGN KEY	383
Присвоение уникальных значений с помощью ограничения UNIQUE	389
Проверка значений столбца с помощью ограничения CHECK	392
Создание временной таблицы с помощью команды CREATE TEMPORARY TABLE	395
Создание новой таблицы на основе существующей с помощью команды SELECT INTO	398
Изменение таблицы с помощью команды ALTER TABLE	401
Удаление таблицы с помощью команды DROP TABLE	404
Глава 11. Индексы	405
Создание индекса с помощью команды CREATE INDEX	405
Удаление индекса с помощью команды DROP INDEX	409
Глава 12. Представления	411
Создание представления с помощью команды CREATE VIEW	411
Считывание данных из представления	417
Изменение данных через представление	419
Удаление представления с помощью команды DROP VIEW	424
Глава 13. Транзакции	425
Выполнение транзакций	426
Приложение	431
Предметный указатель	445

Посвящается моему отцу

Благодарности

Беки Морган (Becky Morgan) – за критику первого варианта рукописи.

Марджори Баер (Marjorie Baer) – за то, что посоветовала прочитать книгу «Do you know SQL?».

Кэти Симпсон (Kathy Simpson) – за то, что иногда ради работы жертвовала сном.

Лизу Бразиил (Lisa Brazieal) – за постоянное «вырезать и вклеить».

Морин Форис (Maureen Forsy) – за отсутствие переносов в ключевых словах.

Брайена Штайнвега (Bryan Steinweg) – за переносы на следующую страницу.

Даррена Пеннингтона (Darren Pennington) – за то, что сажился на своего любимого «конька».

Нэнси Олдридж-Рюнцель (Nancy Aldrich-Ruenzel) – за свежие идеи.

Информация в примере с базой данных – вымышленная. Я позаимствовал пару названий из новелл Яна М. Бэнкса (Iain M. Bank's).

ВВЕДЕНИЕ

SQL – это стандартный язык программирования, применяемый для создания, модификации, поиска и извлечения информации, хранящейся в произвольной реляционной базе данных, управляемой соответствующей системой управления базами данных (СУБД).

С помощью SQL вы можете, в частности, превратить любой вопрос типа «А где живут наши клиенты?» в такую команду, которую программное обеспечение вашей базы данных сможет понять и выполнить (для приведенного вопроса это может быть команда `SELECT city, state FROM customers`). Если вы уже умеете извлекать информацию аналогичного типа с помощью графического инструментария построения запросов, то, скорее всего, заметили, что он становится весьма ограничивающим и громоздким по мере того, как сложность ваших запросов возрастает. Вот здесь и нужен SQL, хотя решением указанной проблемы его возможности не ограничиваются. Например, вы можете применять SQL для того, чтобы добавлять, модифицировать и удалять данные и объекты базы данных. И именно потому, что язык SQL такой мощный, его поддерживают наиболее популярные СУБД, в частности Microsoft Access, Oracle и MySQL, хотя уровень

этой поддержки существенно зависит от того, о какой именно СУБД идет речь.

Прежде чем перейти к собственно программированию, имеет смысл познакомиться с основными идеями реляционных баз данных и SQL. Так мы и поступим.

SQL

SQL – это:

- язык программирования;
- простой в изучении;
- непроцедурный;
- встроенный и/или интерактивный;
- стандартизованный;
- используемый для манипулирования данными и объектами баз данных;
- не аббревиатура.

Рассмотрим пункты этого списка подробнее.

Язык программирования

SQL – один из формальных языков, то есть средство, с помощью которого вы передаете компьютеру инструкции, называемые программой. Программное обеспечение вашей базы данных выполняет

эту программу, написанную на языке SQL. Это значит, что СУБД выполняет те запросы, которые вы ей передали, и отображает результаты их работы, в том числе какое-нибудь сообщение об ошибке. Надо сказать, что языки программирования, называемые также формальными языками, отличаются от языков общения, называемых неформальными или естественными языками, главным образом тем, что создаются под конкретную цель, полностью лишены двусмысленности, имеют весьма ограниченные словарный запас и гибкость. Таким образом, если вы не получили результата от работы своей программы, на который рассчитывали при ее написании, это произошло потому, что ваша программа содержит какую-либо ошибку (логическую или синтаксическую – в последнем случае, скорее всего, будет выведено соответствующее сообщение, описывающее ошибку), а не потому, что компьютер неправильно понял ваши инструкции, формализованные в виде программы (эта информация проявляется, почему отладка программ считается одной из основных задач программирования).

Будучи формальным языком, SQL, как и другой язык этого типа, имеет свои синтаксис и семантику. *Синтаксис* включает собственно слова и символы, которые вы можете применять, а также правила, по которым эти слова и символы можно использовать при создании команд и программ. *Семантика* помогает выяснить реальное значение, смысл любой синтаксически правильной команды. Вы вполне можете написать на SQL какую-нибудь команду, соответствующую синтаксису языка, которая, тем не менее, будет выражать неверный смысл (то есть будет правильной синтаксически, но неверной семантически). Подробнее о синтаксисе и семантике SQL рассказывается в главе 3.

Простой в изучении

Уточним: SQL прост в изучении по сравнению с другими языками программирования. Дело в том, что, если вы пока не написали ни одной программы, переход от неформального языка к любому формальному языку вас поначалу сильно разочарует. Тем не менее подчеркнем, что команды на SQL читаются как предложения неформального языка, что уже сильно облегчает понимание смысла. Так, любой новичок в программировании, скорее всего, поймет, что записанная на SQL команда `SELECT au_fname, au_lname FROM authors ORDER BY au_lname` совпадает по смыслу с предложением «Перечислить имена и фамилии всех авторов, сортируя их по фамилиям». Но этот человек почти наверняка сочтет эквивалентную по смыслу программу, написанную на С или на Perl, абсолютно непонятной.

Непроцедурный

Заметим сразу: если вы никогда не программировали, то можете пропустить этот раздел, не беспокоясь об утрате целостности восприятия остального материала; но если вы программировали на С или Perl, то имели дело с процедурным языком. А для любого процедурного языка программист должен явно прописывать шаги, которые выполняет компьютер, чтобы получить желаемый результат. SQL относится к непроцедурным языкам (такие языки еще называют «декларативными»). Программируя на таком языке, вам надо описать то, что именно вы хотите сделать, а не то, как вы собираетесь это делать. В случае с SQL оптимизатор, который входит в состав программного обеспечения вашей СУБД, самостоятельно рассчитает это самое «как». Именно поэтому язык SQL не содержит таких конструкций

Листинг 1. Этот код на Microsoft Access Visual Basic извлекает имена и фамилии из таблицы в базе данных, содержащей информацию об авторах, и помещает результат выборки в выходной массив

```

Листинг

Sub GetAuthorNames()
    Dim db As Database
    Dim rs As Recordset
    Dim i As Integer
    Dim au_names() As String
    Set db = CurrentDb()
    Set rs = db.OpenRecordset("authors")
    rs.MoveLast
    ReDim au_names(rs.RecordCount - 1, 1)
    With rs
        .MoveFirst
        i = 0
        Do Until .EOF
            au_names(i, 0) = ![au_fname]
            au_names(i, 1) = ![au_lname]
            i = i + 1
            .MoveNext
        Loop
    End With
    rs.Close
    db.Close
End Sub

```

Листинг 2. Эта команда SQL выполняет ту же самую задачу, что и вся процедура на VB, показанная в листинге 1. Внутренний оптимизатор СУБД сам определяет, как наилучшим образом извлечь соответствующие данные

```

Листинг

SELECT au_fname, au_lname
FROM authors;

```

управления, как IF-THEN-ELSE, WHILE и операторов перехода GOTO¹.

Чтобы наглядно продемонстрировать различие между процедурными и непроцедурными языками, мы написали две эквивалентные программы, выполняющие по сути одну и ту же задачу. Первая программа, текст которой приведен в листинге 1, написана на Microsoft Access Visual Basic, то есть на процедурном языке. Вторая программа, текст которой приведен в листинге 2, написана на SQL. Программа на VB извлекает имена авторов из таблицы, содержащей информацию о них. Конечно, нет необходимости вникать в детали первой программы, но обратите внимание, что она использует цикл Do Until для того, чтобы явно определить то, как именно извлекать нужные данные. Вторая программа делает то же самое, но одной-единственной командой SQL. В первом случае мы имеем примерно 20 строк кода, а во втором – всего одну команду. Но важнее всего другое: программируя на SQL, вам надо указать только то, что именно необходимо сделать, а дальше сама СУБД автоматически и незаметно для вас определяет и исполняет ту последовательность пошаговых операций, которую требуется выполнить для достижения желаемого результата. Более того, листинг 2 – это простейший из всех возможных запросов SQL. А если к этому запросу добавить обычные операции сортировки, фильтрации и объединения, то для выполнения всего, что будет делать усложненная, но единственная команда SELECT, записанная на SQL, может понадобиться до 100 строк процедурного кода.

¹ Однако управляющие конструкции обычно присутствуют в SQL и применяются при создании так называемых хранимых процедур – определенных наборов инструкций, предназначенных для выполнения сложных действий с данными. – *Прим. науч. ред.*

Встроенный и/или интерактивный

Если вы включаете команды SQL как элементы в какие-нибудь программы, написанные на процедурных языках, считается, что вы применяете так называемый встроенный SQL, а те процедурные языки, на которых написаны эти более крупные или, точнее, несущие программы, называют базовыми. Таким языком на практике чаще всего оказывается какой-нибудь язык программирования общего назначения, в частности C, Java или COBOL, или какой-либо язык сценариев, например Perl, PHP или Python. То есть любой сценарий PHP, запускаемый как CGI-программа, может применять вложенные команды SQL, чтобы запрашивать произвольную базу СУБД MySQL. СУБД в этом случае передаст результат этого запроса какой-нибудь переменной PHP для отображения и дальнейшей обработки. Например, как это показано в листинге 3, команда SQL встроена в программу, написанную на Access Visual Basic.

Если в режиме реального времени адресовать команды на языке SQL непосредственно вашей СУБД, а она отобразит соответствующие результаты сразу же, как только получит, значит, вы применяете интерактивный или динамический SQL. Отметим, что все современные серверы СУБД поставляются в комплекте с такими графическими приложениями или утилитами командной строки, которые воспринимают интерактивные команды SQL или/и текстовые файлы, содержащие SQL-программы, то есть сценарии.

В настоящем издании рассматриваются только интерактивные команды SQL, поскольку их можно встроить в любую прикладную программу или сценарий. Однако имейте в виду, что между интерактивными и встроенными командами могут существовать небольшие синтаксические различия.

Листинг 3. Здесь Visual Basic выполняет функцию базового языка для встроенного SQL. Однако во всех примерах данной книги используется синтаксис только интерактивного SQL, который может слегка отличаться от синтаксиса встроенного SQL в зависимости от конкретной СУБД

```

Листинг
Sub GetAuthorNames2()
    Dim db As Database
    Dim rs As Recordset
    Set db = CurrentDb()
    Set rs = db.OpenRecordset("SELECT
→ au_fname, au_lname FROM authors;")
    ' Далее программа обрабатывает результаты
→ запроса, находящиеся в переменной rs.
    rs.Close
    db.Close
End Sub

```

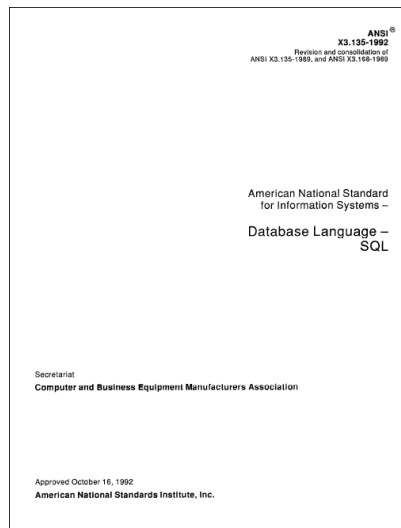


Рис. 1. Титульный лист 626-страничного документа ANSI, который называется X3.135-1992 Database Language – SQL и, собственно, официально определяет стандарт SQL-92.

При желании вы можете купить электронную версию этого документа на сайте www.ansi.org, но он рассчитан главным образом на тех программистов, кто самостоятельно разрабатывает, пишет и внедряет СУБД

Стандартизованный

SQL никому не принадлежит. Это значит, что на права собственности никто не претендует. Как легко понять из рис. 1, SQL – это открытый стандарт, разрабатываемый одним из комитетов в Американском национальном институте стандартизации (ANSI). В свою очередь, ANSI – это не частная фирма, а орган правительства США, призванный способствовать развитию национальных стандартов. Более того, SQL является не только американским, но и международным стандартом ISO/IEC 9075:1992, так как в 1992 году был включен в число стандартов Международной организации по стандартизации (ISO). В этой связи хотелось бы заметить, что настоящее издание основано на стандарте ANSI SQL 1992. Таким образом, когда вы встретите обозначения ANSI SQL, SQL-92 или просто SQL, их следует рассматривать как синонимы, если специально не указано ничего другого. Положение несколько усложняется тем, что появился стандарт ANSI SQL-99. Однако на него пока можно не обращать внимания, так как до его внедрения пройдет еще несколько лет и он будет обратно совместим со стандартом SQL-92. Значит, программа, отвечающая стандарту SQL-92, будет работать под любой СУБД, отвечающей стандарту SQL-99.

Следует заметить, что все поставщики СУБД в разной степени привносят свои дополнения в канонический ANSI SQL, чтобы сделать его более мощным. Чаще всего такими расширениями являются дополнительные команды, ключевые слова, функции, типы данных и конструкции управляющей логики, например IF, WHILE и FOR. Иногда, правда, этих расширений оказывается слишком много. В частности, Oracle и Microsoft уже внесли так много добавлений, что появились PL/SQL и Transact-SQL – самостоятельные

полноправные языки, а не надмножества SQL. Однако обычно расширения одного поставщика несовместимы с расширениями другого. Имея это в виду, мы указываем читателю на те элементы «расширенного» SQL, которые не соответствуют стандарту ANSI (см. раздел «Выполнение команд SQL на коммерческих СУБД»).

Используемый для манипулирования данными и объектами базы данных

Все команды SQL принято делить на две основные группы – язык манипулирования данными (DML) и язык определения данных (DDL).

Для любой базы данных команды группы DML отбирают, обсчитывают, вставляют, удаляют и редактируют те данные, которые хранятся в этой базе. В настоящем издании рассматриваются команды SELECT, INSERT, UPDATE и DELETE (см. главы 4–9).

Команды группы DDL создают, модифицируют и уничтожают такие объекты базы данных, как таблицы, индексы и представления. В нашей книге речь идет о командах CREATE, ALTER и DROP (см. главы 10–13).

Несколько слов о произношении

SQL нельзя произносить как «СИКВЭЛ» (английское слово *sequel* означает «последствие» или «продолжение»). Избегайте этой ошибки, а лучше четко проговаривайте каждую букву из трех: S-Q-L. Мы не согласны с теми людьми, которые произносят «СИКВЭЛ» только потому, что так говорят многие. Дело в том, что смешение *sequel* и SQL является историческим курьезом, образованные люди знают об этом, и для них вариант «СИКВЭЛ» будет свидетельствовать о таком же низком уровне владения ремеслом, как если бы вы оказались писателем и при написании

своих романов стали бы разбивать инфинитивы, вставляя наречия после частицы *to* (что тоже в английском языке вроде бы допускается, но в серьезной литературе почти никогда не встречается). Короче говоря, помните, что «СИКВЭЛ» неприятно звучит для уха профессионала. А еще MySQL следует произносить как «МАЙ-ЭС-КЬЮ-ЭЛЬ», а PostgreSQL – как «ПОСТ-ГРЕС-КЬЮ-ЭЛЬ».

Немного о названии

К сожалению, довольно распространено еще одно заблуждение относительно SQL. Многие считают, что сочетание трех букв S-Q-L есть аббревиатура, которая расшифровывается как Structured Query Language (Структурированный язык запросов). Так вот, SQL означает SQL, и более ничего. Почему? Да потому, что так заявлено в ANSI. Официальное название SQL – Database Language SQL (Язык баз данных SQL). Обращаем особое внимание на данное заблуждение только потому, что оно вовсе не так безобидно. Если начинающие программисты будут считать, что SQL и вправду означает «язык структурированных запросов», это сослужит им плохую службу, так как именно этот язык является наихудшим из всех возможных описаний канонического SQL, хотя живучесть данного описания кажется академикам и профессионалам забавной по следующим причинам:

- SQL – не структурированный язык, поскольку его нельзя разбить на блоки или процедуры;
- SQL не ограничивается только запросами, поскольку в нем есть много других команд помимо SELECT;
- SQL не является полным языком по определению Тюринга.

Несколько общих замечаний о книге

Эта книга рассказывает об использовании SQL для сохранения в базе данных и запрашивания из нее нужной информации. В главах с 1 по 3 приводится общий описательный материал по СУБД, по реляционной модели и по синтаксису SQL. Начиная с главы 4 изложение становится более наглядным и основывается на конкретных примерах.

Автор, безусловно, рассчитывал на то, что читатель хорошо разбирается в своей ОС (будь то Windows, Unix, Mac OS или др.), в частности понимает, как работает файловая система, умеет создавать, редактировать, удалять и организовывать файлы, папки и каталоги. В некоторых задачах требуется умение вводить команды по приглашению операционной системы или командной оболочки (в старых версиях Windows его еще называют приглашением DOS).

Изложение материала книги ограничено наиболее ходовыми командами ANSI SQL, поэтому ее нельзя назвать исчерпывающим руководством. Для изучения команд, которые здесь не рассматриваются, рекомендуем обзавестись подробными справочными пособиями.

Сайт поддержки

На сайте www.peachpit.com/vqs/sql вы найдете все материалы книги (с исправленными ошибками). Свои вопросы, предложения и замечания высылайте по электронной почте: feh1@pacbell.net.

Для кого предназначена эта книга

Эта книга — для вас, если вы:

- не обладаете большим опытом в программировании, но умеете работать на компьютере;
- изучаете SQL самостоятельно или с инструктором;
- не интересуетесь базами данных, но по роду своей деятельности вынуждены обрабатывать большие объемы структурированной информации (как это часто имеет место в работе бухгалтера, научного сотрудника, Web-программиста, аналитика рынка, торгового представителя, финансового советника, руководителя офиса и даже секретаря);
- хотите выйти за ограничения, налагаемые дружественным, но крайне мало-мощным графическим инструментарием построения запросов по образцу (QBE – Query by Example);
- намерены перейти от программного обеспечения настольной СУБД к серверному программному обеспечению, поддерживающему SQL;
- уже знакомы с SQL и хотите узнать о нем как можно больше;
- должны по роду деятельности создавать, модифицировать или/и удалять такие объекты баз данных, как таблицы, индексы и представления;
- вынуждены встраивать SQL в программы, написанные на языках Java, C или Perl;
- являетесь Web-программистом и работаете с MySQL или PostgreSQL;
- просто хотите иметь настольный справочник по SQL.

Эта книга вам совершенно не подходит, если вы планируете изучить следующие вопросы:

- как проектировать базы данных (хотя обзоры толковых идей, которыми следует руководствоваться, приводится в главе 2);
- частные расширения, которые поставщики СУБД добавили к ANSI SQL;
- как профессионально и с применением современных методов программировать и администрировать базы данных (мы не касаемся, например, инсталляции, триггеров, хранимых процедур, тиражирования, резервирования, восстановления, оптимизации, курсоров, схем сортировки, сравнений, алфавитов и трансляции).

Дополнительные соглашения

Иногда части текста выделяются разными шрифтами:

- моноширинный шрифт служит для написания кода как в листингах, так и в тексте книги и для обозначения экранного текста в командной строке;
- *курсив* применяется при вводе новых терминов и, совместно с моноширинным шрифтом, при обозначении заменяемых имен идентификаторов;
- участки рисунков, на которые следует обратить внимание, обведены;
- полужирным моноширинным шрифтом отмечены результаты, о которых говорится в тексте;
- полужирным шрифтом выделены элементы интерфейса.

При рассмотрении схем синтаксиса SQL и листингов будем придерживаться следующих правил:

- для улучшения читабельности и сохранности текста кода служит плотная печать. А для экономии места на страницах книги ширина отступа (величина отступа слева при начале абзаца) выбрана равной двум пробелам, хотя обычно она равна четырем пробелам;

Настольные и серверные СУБД, поддерживающие SQL

Любая серверная СУБД, поддерживающая SQL, работает в качестве серверной части архитектуры «клиент-сервер». Это значит, что она хранит и поддерживает базы данных и отвечает на запросы, которые клиенты оформляют на языке SQL. Здесь под клиентом понимается любое приложение или любой компьютер, которые могут посылать запросы на языке SQL (в дальнейшем мы будем называть такие запросы SQL-запросами) на сервер и получать ответы с этого сервера. Например, если в вашей локальной сети применяется архитектура «клиент-сервер», то клиентом является компьютер на вашем рабочем столе, а сервером – мощная специальная машина, расположенная в соседней комнате, в соседнем здании или в другой стране. При этом правила, описывающие порядок передачи запросов и соответствующих ответов в архитектуре «клиент-сервер», определяются протоколом связи с СУБД, например ODBC и JDBC.

Любая настольная СУБД – это СУБД, установленная и работающая локально. Она представляет собой независимое приложение, которое хранит свои собственные базы данных и производит всю обработку данных, связанную с SQL. Никакая настольная СУБД не может принимать запросы от других клиентов, что, конечно, означает, что никакая настольная СУБД не может работать так, как это делает любой SQL-сервер.

В качестве примеров SQL-сервера можно привести Microsoft SQL Server, Oracle, MySQL и PostgreSQL (заметим, что понятие SQL-сервера включает в себя понятие серверной СУБД, поддерживающей SQL, но не совпадает с ним). А в качестве настольных СУБД, поддерживающих SQL, можно назвать Microsoft Access и FileMaker Pro. В отношении программных продуктов, англоязычные названия которых содержат словосочетание SQL server, следует знать об одном нюансе. Дело в том, что SQL server (слово server написано с маленькой буквы) относится к коммерческому серверному продукту, поддерживающему SQL, произвольного поставщика, а SQL Server (слово Server написано с большой буквы) – только к конкретному продукту SQL Server компании Microsoft.

В соответствии с устоявшимися требованиями мы будем использовать слова «клиент» («клиентский») и «сервер» («серверный») по отношению не только к клиентскому и, соответственно, серверному программному обеспечению, но и к той машине, на которой работает соответствующее программное обеспечение. В тех случаях, когда надо подчеркнуть различие между компьютером и программным обеспечением, мы будем делать специальные пояснения.

- каждая команда SQL начинается с новой строки;
- как вы узнаете из главы 3, SQL является языком свободной формы без ограничений на разрыв строки или на число слов в одной строке. У нас же каждое предложение любой команды начинается с новой строки и с отступом от первой строки, например:

```
SELECT au_fname, au_lname
FROM authors
ORDER BY au_lname;
```

- в SQL регистр не учитывается, следовательно, идентификаторы `myname`, `MyName` и `MYNAME` считаются идентичными с точки зрения канонического SQL. Однако верхний регистр используется нами для выделения ключевых слов, таких как `SELECT`, `NULL` и `CHARACTER` (см. раздел «Синтаксис SQL» в главе 3), а нижний регистр – для выделения элементов, конкретные значения которых должны быть определены пользователем, как это имеет место, например, для имен таблиц, столбцов и для псевдонимов. Стоит особо подчеркнуть, что в некоторых СУБД имена идентификаторов, определяемые пользователем, являются, в отличие от канонического SQL, чувствительными к регистру. Поэтому безопаснее все-таки учитывать регистр при написании программ;
- в табл. 1 поясняется смысл специальных символов, используемых в схемах и комментариях синтаксиса SQL;
- все кавычки в кодах SQL только прямые – или одинарные (`'`), или двойные (`"`). Никакие другие кавычки (имеются в виду фигурные) недопустимы. Неправильные кавычки не воспринимаются транслятором, и код, содержащий их, работать не будет;
- когда колонка, отведенная для строки кода, окажется для нее слишком узкой, то строка будет разбита на две или более

частей. Стрелка (\rightarrow) в начале строки означает, что эта строка является продолжением предыдущей.

С

Этим значком помечены советы, которые помогут вам в практической работе.

П

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

Выполнение команд SQL на коммерческих СУБД



В тексте книги иногда встречается пиктограмма, которая означает, что речь идет об отступлении производителями СУБД от канонов стандартного ANSI SQL-92. Поэтому, увидев такой значок, вам придется определенным образом модифицировать код, приведенный в каком-нибудь листинге, под авторскую версию SQL конкретного поставщика СУБД, чтобы вы смогли запустить этот код на своей СУБД. Например, если в тексте говорится, что оператор объединения (конкатенации) двух строк в ANSI SQL является `||` (двойной конвейер или труба), но продукты Microsoft применяют для этой цели знак «+» (плюс), а MySQL применяет вместо оператора конкатенации функцию `CONCAT()`, вам, если вы работаете на соответствующей СУБД, придется во всех выражениях типа *allb* в запросе поменять знак «`||`» или на знак «+», или на функцию `CONCAT()`.

В подавляющем большинстве случаев наши программы SQL будут работать или совсем без изменений, или с незначительными синтаксическими поправками на всех основных СУБД. К сожалению, хотя и редко, некоторые программы не будут работать совсем. Скажем, MySQL 4.0 и более ранних версий не поддерживает

подзапросы (хотя их поддержка планировалась в версии 4.0).

Перечислим СУБД, учитываемые в настоящем издании:

- Microsoft Access 2002;
- Microsoft SQL Server 2000;
- Oracle Release 9i;
- MySQL 4.0;
- PostgreSQL 7.1.

Мы остановились на этих СУБД, потому что они самые ходовые. Все программы SQL, приведенные в данной книге, были протестированы указанными коммерческими версиями СУБД. Поскольку соответствие базовому ANSI SQL в последующих версиях никогда не ухудшается, наши программы будут работать и на всех будущих версиях этих СУБД.

Сразу заметим, программы из этой книги вы можете запустить и на DB2, и на Sybase Adaptive Server, и на Informix. Если какая-то программа откажется работать, прочитайте документацию по соответствующей СУБД и определите, в чем ее реализация SQL отличается от стандартного ANSI SQL.

П

В список избранных СУБД версия Oracle 8i тоже включена. Просто имейте в виду: если номер версии этой СУБД не указан, значит, примечания или советы имеют отношение как к Oracle 9i, так и к Oracle 8i.

Таблица 1. Специальные символы, применяемые для объяснений правил синтаксиса

Знаки	Назначение
	Символ «вертикальная черта» или «труба» разделяет альтернативные элементы, выбор одного из которых может быть, судя по контексту, как обязательным, так и необязательным. В любом случае можно выбрать не более одного из этих элементов. Никогда не печатайте этот символ в тексте программ! Выражение A B C будет означать или «Или А, или В, или С», или «Или А, или В, или С, или ничего». Не путайте знак трубы со знаком двойной трубы . Дело в том, что двойная труба является оператором SQL и применяется для конкатенации (объединения) строк
[]	В квадратные скобки заключаются элементы, выбор хотя бы одного из которых необязателен. Никогда не печатайте этот символ в тексте программ! Выражение [A B C] будет означать: «Печатай или А, или В, или С или не печатай ничего»
{}	В фигурные скобки заключаются элементы, выбор одного из которых обязателен. Никогда не печатайте этот символ в тексте программ! Выражение {A B C} будет означать: «Печатай или А, или В, или С»
...	Многоточие означает, что предыдущий элемент/элементы можно повторить любое количество раз
()	В отличие от всех предыдущих знаков, круглые скобки являются элементом языка SQL. Их надо печатать там, где указывает соответствующая грамматическая схема

Разница между базой данных и СУБД

Любая база данных – это совсем не то программное обеспечение, на котором вы работаете. Поэтому говорить, что Oracle – это база данных, неправильно. Полное название программного обеспечения, обслуживающего базы данных, звучит так: система управления базами данных (СУБД). Важно понять, что любая база данных является не более чем компонентом некой СУБД, что она является элементом данных, будучи не более чем контейнером (состоящим из одного или нескольких файлов), содержащим структурированную информацию. Помимо задач контроля, организации, выборки, поддержания целостности и достоверности данных любая СУБД решает еще задачи обеспечения физического хранения, безопасности, тиражирования данных и исправления ошибок при их обработке.

Иногда вместо аббревиатуры СУБД используют аббревиатуру РСУБД, имея в виду именно реляционную СУБД. Такая СУБД организует данные в соответствии с реляционной моделью (о реляционной модели мы подробнее поговорим в главе 2). Хотим отметить, что в настоящем издании рассматриваются только реляционные базы данных, поэтому при использовании аббревиатуры СУБД речь идет о РСУБД.

Что такое FileMaker Pro

Если говорить кратко, FileMaker Pro – это распространенное настольное приложение со встроенной базой данных, которое поддерживает команду SQL SELECT с предложениями FROM, WHERE и ORDER BY (подробнее об этом см. в главе 4). Этих возможностей вполне достаточно для выполнения самых распространенных типов запросов. Чтобы запустить любой допустимый SQL-запрос на FileMaker Pro, можно воспользоваться инструментарием SQL Query Builder или оператором Execute SQL script. Основательно разобраться в этих вопросах поможет справочная система FileMaker Pro или книга Цинтии Л. Барон и Даниеля Пека (Cynthia L. Baron, Daniel Peck) «FileMaker Pro 5/5.5 Advanced: Visual QuickPro Guide». Наконец, вы можете многое узнать о FileMaker Pro на сайте www.filemaker.com.

Программное обеспечение, необходимое для полноценной работы с этой книгой

Чтобы воспроизвести примеры настоящей книги на компьютере, вам понадобится:

- текстовый редактор;
- типовая база данных;
- СУБД.

Текстовый редактор

Хотя печатать короткие интерактивные команды SQL в ответ на приглашение командной строки может показаться даже удобным, обычно вы все-таки стараетесь сохранять нетривиальные запросы в текстовых файлах. В соответствии с общепринятыми правилами имена файлов SQL имеют расширение .sql, хотя вы можете выбрать любое другое расширение, например .txt.

Любой текстовый редактор — это некая служебная программа для создания и модификации текстовых файлов. Как известно, такие файлы содержат только печатные буквы, числа и символы, но никаких шрифтов, никакого форматирования, никаких «невидимых» знаков, цветов и графики, то есть ничего из той системы, которую принято связывать с понятием текстового процессора, в них нет. Хотя каждая операционная система обычно предоставляет вместе с неким текстовым редактором (см. табл. 2), вы вполне можете поэкспериментировать с разными коммерческими и условно бесплатными текстовыми редакторами (см. сайт www.download.com).

Таблица 2. Распространенные текстовые редакторы

Среда	Текстовый редактор
Windows	Notepad
Приглашение командной строки Windows	edit
Unix, Linux	vi, emacs, pico
Mac OS	Teach Text, Simple Text
Mac OS X Terminal	vi, emacs, pico

П Все программы SQL, содержащиеся в настоящей книге, можно скачать с сайта поддержки (см. раздел «Несколько общих замечаний о книге»).

П Вы вполне можете обойтись и без текстового редактора, используя какой-нибудь текстовый процессор, скажем Microsoft Word, и сохраняя соответствующие файлы как «text only». Но такой подход весьма проблематичен, и профессионалы рассматривают его как плохой стиль.

Типовая база данных

Большинство примеров, рассматриваемых в данной книге, рассчитаны на одну и ту же базу данных (см. раздел «Наша типовая база данных» в главе 2). Чтобы ее построить, запустите SQL-скрипт, приведенный в приложении, или, если вы работаете с Microsoft Access, откройте файл books.mdb (все эти файлы вы можете скачать с сайта поддержки). Если вы работаете с действующей серверной СУБД организации, то на создание базы данных и на запуск команд SQL вам может понадобиться официальное разрешение системного администратора.

Система управления базами данных

Для выполнения SQL-запросов просто передайте их какой-нибудь СУБД, которая «понимает» SQL. И уже она выполнит ваш запрос и отобразит соответствующие результаты. В первой главе мы расскажем о том, как запускать SQL-запросы на самых распространенных СУБД.

ОСНОВНЫЕ ХАРАКТЕРИСТИКИ СУБД

1

Итак, для выполнения программ SQL вам понадобится СУБД. Возможны две ситуации: или у вас на компьютере есть личная работающая копия СУБД, или вы имеете доступ к некой СУБД по сети. Во втором случае вам придется подключиться к какому-то серверу СУБД, размещенному на какой-то другой машине. Компьютер, на котором работает этот сервер, называется хостом. Поскольку наша книга не о СУБД, а об SQL, мы не станем пересказывать инструкции по установке и конфигурированию программного обеспечения баз данных. Отметим лишь, что процесс установки любой СУБД сильно зависит от компании-поставщика, самого продукта, его версии и конкретной операционной системы. Вот почему все СУБД выпускаются с весьма обширной документацией по установке, администрированию и эксплуатации, включая руководства пользователя, учебный и справочный материал.

Листинг 1.1. Файл listing0101.sql содержит простую SQL-команду SELECT, которую мы будем использовать для построения запросов к типовой базе в примерах, относящихся ко всем СУБД

```
ЛИСТИНГ
SELECT au_fname, au_lname
FROM authors
ORDER BY au_lname;
```

Выполнение программ SQL

В настоящей главе объясняется, как выполнять SQL-запросы и скрипты на следующих СУБД:

- Microsoft Access 2002;
- Microsoft SQL Server 2000;
- Oracle 9i;
- MySQL 4.0;
- PostgreSQL 7.1.

Графический интерфейс Microsoft Access позволит запускать только по одному SQL-запросу за раз, а для того, чтобы выполнить SQL-скрипт (то есть последовательность команд), вам придется встраивать команды SQL в какую-нибудь программу на Visual Basic.

Все остальные СУБД позволяют все то, что позволяет Microsoft Access, а также исполнять запросы и в интерактивном режиме, и в виде скрипта. В *интерактивном режиме* вы будете печатать отдельные команды SQL в командной строке и просматривать результаты выполнения каждой команды так, что ввод будет чередоваться с выводом результатов выполнения запросов. В *режиме скрипта*, или *сценария* (который еще иногда называют *пакетным режимом*), вы будете записывать все требуемые для выполнения запросы в текстовый файл (который в этом случае будет называться *файлом скрипта* (*сценария*), или *пакетным файлом*), а утилита командной строки выполнит записанные в этом файле запросы и вернет результаты.

Во всех примерах настоящей главы мы будем использовать одну и ту же типовую базу данных (о которой упоминалось во введении), команды SQL, записанные так же, как в листинге 1.1, и приведем минимально необходимый синтаксис утилит командной строки. Если вы хотите глубже

изучить этот синтаксис, обратитесь к справочному разделу документации по вашей СУБД.

П При указании имени любого файла SQL-скрипта в режиме сценария вы можете обозначить его и как абсолютное полное, и как относительное имя (см. врезку «Полный путь, или полное имя»).

П Чтобы запустить любую утилиту командной строки из заданного каталога, ваш *маршрут* (сокращенное название для понятия маршрутной переменной среды; не путать с полным именем) должен включать фактический каталог, в котором содержится эта утилита. Имейте в виду, что любой маршрут представляет собой список каталогов, в которых операционная система «ищет» программы. Особенность заключается в том, что инсталляторы одних СУБД детально формируют маршруты, а для других СУБД (как, например, MySQL и PostgreSQL) вам придется самостоятельно добавить каталог соответствующей утилиты в маршрут.

Для того чтобы увидеть свой маршрут, в командной строке напечатайте `path` (для Windows) или `echo $PATH` (для Unix или Mac OS X Terminal) и нажмите **Enter**.

Чтобы изменить маршрут, добавьте в маршрутную переменную среды каталог, в котором постоянно находится нужная утилита. Более подробную информацию по этому вопросу можно получить в справочной системе, если вы работаете под Windows (поиск по ключу «переменная среды»), а если вы работаете под Unix или Mac OS X Terminal, то можете модифицировать команду `path` в файле инициализации сеанса, который обычно называется `.bash_login`, или `.bashrc`, или `.cshrc`, или `.login`, или `.profile`, или `shrc`.

Полный путь, или полное имя

Любой *полный путь* указывает местоположение файла или каталога в иерархической файловой системе. При этом *абсолютный полный путь* определяет положение файла полностью посредством явного указания всего маршрута, начиная с самого верхнего узла, называемого *корнем*, дерева каталогов. В свою очередь, *относительный путь* прописывает местоположение файла или каталога не абсолютно, а по отношению к текущему месту. В операционной системе Windows любой абсолютный полный путь обязательно начинается или с обратного слэша (\), или с буквы, обозначающей дисковод, за которой следует двоеточие и слэш. В операционных системах Unix/Linux и Mac OS X любое абсолютное имя всегда начинается со слэша, наклоненного вправо (/).

Например, `C:\Program Files\Microsoft SQL Server` (для Windows) и `/usr/local/bin/mysql` (для Unix/Linux) – это абсолютные полные пути, а `scripts\listing0101.sql` (для Windows) и `doc/readme.txt` (для Unix) – это относительные имена.

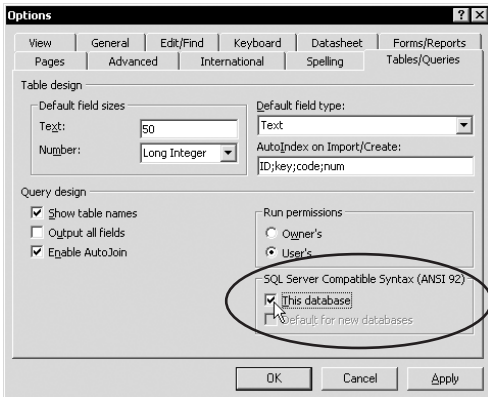


Рис. 1.1. Режим запросов SQL ANSI-92 даст вам возможность исполнять команды SQL, которые используют синтаксис ANSI-92 SQL

Microsoft Access

Microsoft Access – это коммерческая СУБД, которая служит для управления базами малого и среднего размера. Дополнительную информацию об Access вы можете узнать на сайте www.microsoft.com/office/access/.

Настоящее издание написано с расчетом на Microsoft Access 2002. Если вы не знаете точно, с какой версией СУБД работаете, выберите пункт меню: **Help [About Microsoft Access]** (Помощь [О Microsoft Access]).

Имейте в виду, что в Access 2002 по умолчанию установлен режим запросов ANSI-89. Чтобы иметь возможность исполнить многие из тех примеров, что приведены в настоящей книге, установите режим запросов ANSI-92.

Установка режима запросов SQL

Порядок действий:

1. Откройте диалоговое окно **Options** (Настройки), выполнив команды **Tools [Options]** (Инструменты [Опции]).
2. Откройте вкладку **Tables/Queries** (Таблицы/Запросы).
3. Установите флажок **This Database** (Эта база данных) – см. рис. 1.1.



Не торопитесь вводить этот режим для остальных ваших баз данных. Дело в том, что режимы запросов ANSI SQL-89 и ANSI SQL-92 несовместимы и их области типов данных, универсальных символов и зарезервированных слов существенно различаются. То есть в другом режиме SQL-запросов ваши команды могут или совсем не работать, или давать неожиданные результаты.



Подробнее о режимах запросов посмотрите в справочной системе Access (поиск по ключу «ANSI SQL query mode»).

Если вы работаете с Access лишь время от времени, то, скорее всего, создавая свои запросы, постоянно пользуетесь графическим конструктором запросов. Так вот, когда вы составляете любой запрос в режиме конструктора, Access автоматически и незаметно для вас генерирует эквивалентную вашему запросу команду SQL. Вы можете исполнять, просматривать и редактировать эту команду в режиме **SQL view**.

Чтобы исполнять команды языков DML и DDL (языки модификации и определения данных; см. раздел «SQL» во введении), Access применяет два почти одинаковых интерфейса. При этом команды языка DML – это SELECT, INSERT, UPDATE и DELETE, а команды языка DDL – это CREATE, ALTER и DROP.

Исполнение произвольной команды SQL

Необходимо выполнить следующие действия:

1. В окне базы данных щелкните по пиктограмме **Queries** (Запросы), которая расположена слева на панели **Objects** (Объекты), а потом по пиктограмме **New** (Создать), которая расположена сверху на панели инструментов (см. рис. 1.2).
2. В диалоговом окне **New Query** (Новый запрос) щелкните по опции **Design View** (Конструктор), а потом по кнопке **OK** (см. рис. 1.3).
3. В графическом бланке конструктора запроса, не добавляя ни таблиц, ни запросов, щелкните по кнопке **Close** (Закреть).

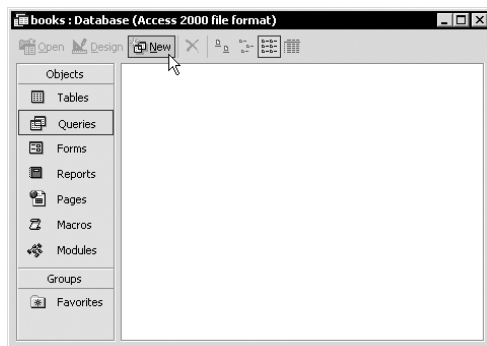


Рис. 1.2. Чтобы создать новый запрос, щелкните по кнопке **New** (Создать) на панели инструментов

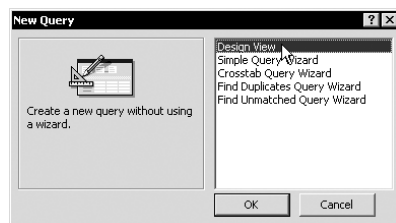


Рис. 1.3. В диалоговом окне **New Query** (Новый запрос) выберите опцию **Design View** (Конструктор)

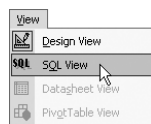


Рис. 1.4. Чтобы запустить запрос на языке DML (включающий в данном случае только команды SELECT, INSERT, UPDATE, DELETE), выберите опцию **SQL**

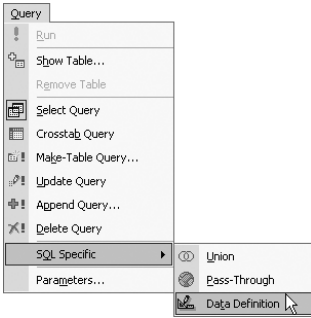


Рис. 1.5. Чтобы запустить запрос на языке DDL (включающий в данном случае только команды CREATE, ALTER, ROP), выберите опцию **Data Definition**



Рис. 1.6. Введите какую-нибудь команду SQL...

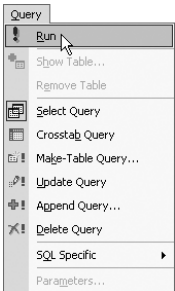


Рис. 1.7. ...и запустите ее

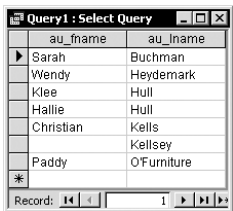


Рис. 1.8. Результаты выполнения команды SELECT

4. Чтобы запустить произвольную команду:

- языка DML – выберите команды **View [SQL View** (Режим [Режим SQL), см. рис. 1.4;
- языка DDL – выберите команды **Query [SQL Specific [Data Definition** (Запрос [Запрос SQL [Управление), см. рис. 1.5.

5. Наберите или вставьте через буфер обмена какую-нибудь команду SQL (например, из листинга 1.1; см. рис. 1.6).

6. Теперь, чтобы запустить команду SQL, выберите последовательно пункты **Query [Run** (Запрос [Выполнить) – см. рис. 1.7.

Access выведет результаты команды SELECT (см. рис. 1.8), но результаты выполнения любой другой команды SQL в зависимости от персональных настроек компьютера или вообще не будут показаны, или будут показаны только в виде каких-то окон, предупреждающих о возможных нештатных ситуациях.



Через один объект **Query** вы можете выполнить только одну команду SQL. Чтобы запустить несколько команд, используйте серию объектов **Query** или напишите программу на встроенном языке Visual Basic for Applications, которая выполнит ряд запросов последовательно.

Microsoft SQL Server

Microsoft SQL Server, в отличие от настольной СУБД Microsoft Access, является серверным программным обеспечением и поддерживает очень большие базы данных и огромное число транзакций. Microsoft SQL Server работает только под операционными системами компании Microsoft. Подробнее о Microsoft SQL Server можно узнать на сайте www.microsoft.com/sql/. Еще вы можете скачать бесплатную оценочную версию этого продукта, которая будет «жить» 120 дней, с сайта www.microsoft.com/sql/evaluation/.

Настоящая книга описывает Microsoft SQL Server 2000. Чтобы определить версию Microsoft SQL Server, с которой вы работаете, введите `SELECT @@VERSION`, нажмите **Enter**, в командной строке `osql` напечатайте `go` и снова нажмите **Enter**.

Чтобы исполнять программы SQL, можно воспользоваться графическим инструментарием SQL Query Analyzer или утилитой командной строки `osql`. Рассмотрим оба варианта.

Работа с SQL Query Analyzer

Порядок действий:

1. Щелкните последовательно по пунктам меню **Start** [**Programs** [**Microsoft SQL Server** [**Query Analyzer**.
2. Укажите сервер, имя пользователя и пароль и нажмите **OK**.
3. Выберите в соответствующем раскрывающемся списке панели инструментов какую-нибудь базу данных, например `books` (см. рис. 1.9).

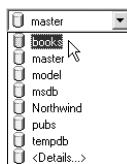


Рис. 1.9. Чтобы выполнить все ссылки (предложения), указанные в командах SQL, Query Analyzer использует выбранную вами базу данных

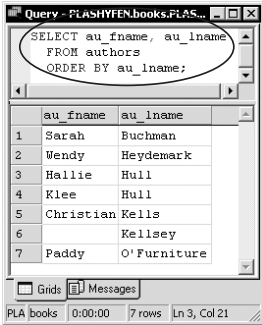


Рис. 1.10. Результаты исполнения команды `SELECT` в Query Analyzer

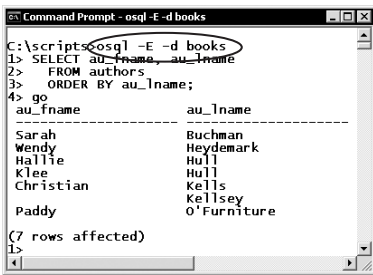


Рис. 1.11. Команда SQL в интерактивном режиме утилиты командной строки `osql`

4. Для работы с SQL есть два альтернативных способа:

- запустите SQL интерактивно, набрав какую-нибудь команду SQL, например приведенную в листинге 1.1, непосредственно в окне запроса;
- запустите SQL-сценарий, выбрав последовательно опции **File** [**Open** (Файл [Открыть)], потом какой-нибудь файл сценария и щелкнув по кнопке **Open** (Открыть).

5. Выберите опции **Query** [**Execute** (Запрос [Выполнить)]. Самое нижнее подокно покажет результат выполнения вашей команды (см. рис. 1.10).

П Чтобы запустить SQL Query Analyzer, можно воспользоваться утилитой командной строки `isqlw`.

Применение утилиты командной строки `osql` в интерактивном режиме

Порядок действий:

1. Наберите `osql -E -d dbname`, где:
 - `-E` указывает SQL Server не запрашивать пароль, а использовать доверительное соединение;
 - вместо `dbname` следует подставить имя базы данных, с которой вы собираетесь работать.
2. Введите любую команду SQL, например ту, что показана в листинге 1.1, и нажмите **Enter**. Имейте в виду, что команда не обязательно должна умещаться на одной строке.
3. Напечатайте `go` и посмотрите на результаты (см. рис. 1.11).

Использование утилиты командной строки `osql` в режиме сценария

В командной строке наберите

```
osql -E -d dbname -n -i sql_script,
```

где:

- `-E` инструктирует SQL Server не запрашивать пароль, а использовать доверительное соединение;
- вместо `dbname` вы подставите имя базы данных, с которой собираетесь работать;
- `-n` «гасит» при выдаче результатов нумерацию и символы приглашения «>»;
- вместо `sql_script` вы подставите или абсолютное, или относительное полное имя текстового файла, в котором записана некая программа SQL (см. листинг 1.1 и рис. 1.12).

Выход из утилиты командной строки `osql`

Напечатайте `exit` или `quit` и нажмите **Enter**.

Отображение списка опций командной строки `osql`

В командной строке напечатайте `osql -?` и нажмите **Enter**.

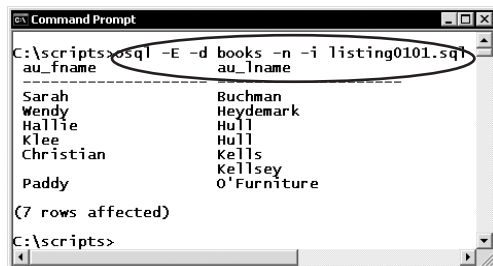


Рис. 1.12. Команда SQL в режиме сценария для утилиты командной строки `osql`



Настройки СУБД SQL Server могут потребовать указания имени пользователя и пароля, вместо того чтобы предоставить вам доверительное соединение. Для ввода пароля замените сначала опцию `-E` опцией `-U login_id`, где вместо `login_id` нужно подставить имя пользователя. Только после этого система запросит пароль.



Если вы выполняете команду `osql` с удаленного компьютера и по сети, то для указания системы SQL Server, к которой вы собираетесь подключиться, придется применить опцию `-S server`. Об особенностях использования этой опции узнайте у системного администратора.

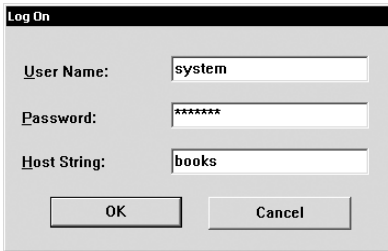


Рис. 1.13. Экран регистрации интерфейса SQL*Plus

Oracle

Oracle – это ведущая коммерческая СУБД, способная поддерживать гигантские базы данных и невероятные количества транзакций. Она работает под многими операционными системами и на разных платформах аппаратно-технического обеспечения. Наконец, эта СУБД настолько сложна, что для поддержания ее в рабочем состоянии нужен высококвалифицированный администратор баз данных.

На сайте www.oracle.com вы можете узнать об Oracle всю необходимую информацию. Вы даже можете скачать рассчитанную на единственного пользователя бесплатную версию «только для разработчика» этой СУБД с сайта сетевой поддержки технологии Oracle (otn.oracle.com). Неудобство заключается в том, что сам файл очень большой и ту же версию наверняка придется купить на CD-диске.

Для того чтобы исполнять под Oracle команды SQL, можно воспользоваться или графическим инструментарием SQL*Plus, или утилитой командной строки sqlplus. Имейте в виду, что в настоящем издании описывается главным образом версия Oracle 9i. Однако в книге вы найдете несколько советов и по Oracle 8i. Уточнить вашу версию Oracle можно при регистрации в системе для работы с SQL*Plus или sqlplus. В этом случае автоматически появляется сообщение **Connected to** (Подключен к...).

Использование графического интерфейса SQL*Plus

Порядок действий:

1. Запустите SQL*Plus. Эта процедура, однако, зависит от платформы. Например, под Windows вам следует выбрать опции **Programs** [**Oracle-OraHome91** [**Application Development** [**SQL Plus**.
2. Введите имя пользователя, пароль и имя базы данных, с которой вы хотите работать. Нажмите **OK** (см. рис. 1.13). Если вы

собираетесь работать с удаленной базой Oracle, узнайте у администратора идентификатор подключения, который надо ввести в текстовом окне **Host String**. По умолчанию учетная запись администратора базы данных содержит в качестве имени пользователя **SYSTEM**, а паролем по умолчанию является **manager**.

3. Теперь у вас есть две возможности:

- чтобы работать с SQL интерактивно, просто наберите какую-нибудь команду SQL и нажмите **Enter**; при этом не обязательно уместить эту команду в одну строку; строк может быть несколько (см. рис. 1.14);
- чтобы запустить произвольный сценарий SQL, напечатайте `@sql_script` и нажмите **Enter**. Вместо `sql_script` вы подставите имя текстового файла, в котором записана ваша программа SQL; оно может быть как абсолютным полным именем, так и относительным (см. рис. 1.15).

Применение утилиты командной строки `sqlplus` в интерактивном режиме

Порядок действий:

1. В командной строке напечатайте:
`sqlplus user/password@dbname.`
Здесь вместо *user* надо подставить имя пользователя, вместо *password* – пароль, а вместо *dbname* – имя той базы, с которой вы собираетесь работать.
2. Введите любую команду SQL (количество строк не имеет значения).
3. Нажмите **Enter** и посмотрите на результаты (рис. 1.16).

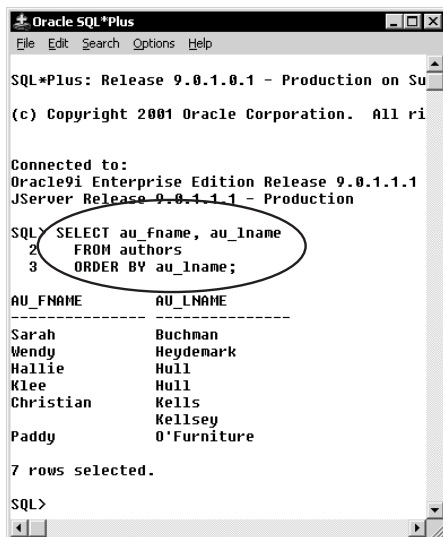


Рис. 1.14. Результаты интерактивного выполнения команды `SELECT` средствами `SQL*Plus`

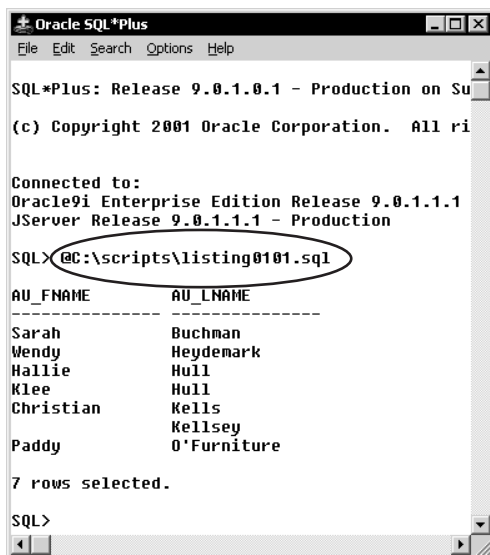


Рис. 1.15. Команда `SELECT`, выполненная с помощью `SQL*Plus`, но как сценарий

```

C:\scripts>sqlplus system/manager@books

SQL*Plus: Release 9.0.1.0.1 - Production on Su
(c) Copyright 2001 Oracle Corporation. All ri

Connected to:
Oracle9i Enterprise Edition Release 9.0.1.1.1
JServer Release 9.0.1.1.1 - Production

SQL> SELECT au_fname, au_lname
      2 FROM authors
      3 ORDER BY au_lname;

AU_FNAME      AU_LNAME
-----
Sarah          Buchman
Wendy          Heydemark
Hallie         Hull
Klee           Hull
Christian      Kells
Paddy          Kellsey
               O'Furniture
7 rows selected.

SQL>

```

Рис. 1.16. Результаты выполнения команды `SELECT` утилитой `sqlplus` в интерактивном режиме

Использование утилиты командной строки `sqlplus` в режиме сценария

В командной строке напечатайте:

```
sqlplus user/password@dbname @sql_script
```

Здесь вместо *user* надо подставить имя пользователя, вместо *password* – пароль, вместо *dbname* – имя той базы, с которой вы собираетесь работать, а вместо *sql_script* – имя текстового файла, в котором записана ваша программа SQL; оно может быть как абсолютным полным именем, так и относительным. Не забудьте вставить пробел между *dbname* и *@sql_script* (см. рис. 1.17).

Выход из утилиты командной строки `sqlplus`

Напечатайте `exit` или `quit` и нажмите **Enter**.

Отображение списка опций утилиты `sqlplus`

В командной строке наберите:

```
sqlplus -?
```

и нажмите **Enter**.

```

C:\scripts>sqlplus system/manager@books @listing0101.sql

SQL*Plus: Release 9.0.1.0.1 - Production on Sun May 12 20
(c) Copyright 2001 Oracle Corporation. All rights reserv

Connected to:
Oracle9i Enterprise Edition Release 9.0.1.1.1 - Productio
JServer Release 9.0.1.1.1 - Production

AU_FNAME      AU_LNAME
-----
Sarah          Buchman
Wendy          Heydemark
Hallie         Hull
Klee           Hull
Christian      Kells
Paddy          Kellsey
               O'Furniture
7 rows selected.

SQL>

```

Рис. 1.17. Результаты выполнения команды `SELECT` утилитой `sqlplus` в режиме сценария

C

Если вы хотите, чтобы система сама спросила у вас пароль, исключите из командной строки `sqlplus` опцию `/password`.

MySQL

MySQL – это одна из ведущих (то есть наиболее распространенных, качественных и покупаемых в настоящее время) СУБД с открытым исходным кодом. Она отличается быстродействием, устойчивостью и возможностью поддерживать базы данных больших объемов. Но, несмотря на все ее достоинства, среди которых основным и наиболее известным является быстродействие, MySQL не поддерживает нескольких очень важных функций SQL (планируется, что версия 4.1 будет поддерживать подзапросы и внешние ключи). Немаловажно и то, что MySQL работает под многими популярными операционными системами и на разных аппаратных платформах, будучи при этом бесплатной. Вы можете скачать эту СУБД с сайта www.mysql.com.

В настоящем издании рассматривается MySQL 4.0. Чтобы выяснить, с какой версией вы работаете, в командной строке `mysql` введите `SELECT VERSION();` и нажмите **Enter**.

Чтобы исполнять программы SQL, в среде рассматриваемой СУБД можете применять утилиту командной строки `mysql`.

Использование утилиты командной строки `mysql` в интерактивном режиме

Порядок действий:

1. Напечатайте в командной строке:

```
mysql dbname.
```

Предполагается, что вместо *dbname* вы подставите имя базы данных, с которой собираетесь работать.

2. Введите какую-нибудь команду SQL, например из листинга 1.1. При этом запись этой команды может занимать более одной строки.

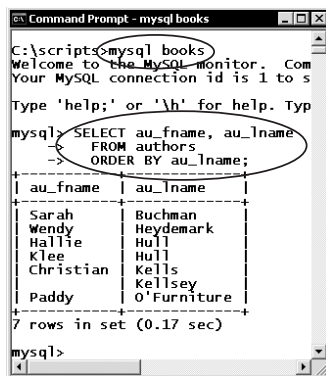


Рис. 1.18. Результат интерактивного выполнения команды `SELECT` утилитой командной строки `mysql`

3. Нажмите **Enter** и посмотрите на результаты (рис. 1.18).

Использование утилиты командной строки `mysql` в режиме сценария

В командной строке напечатайте `mysql -t dbname < sql_script`,

где:

- `-t` инструктирует систему поместить результаты в некую таблицу (если вы предпочитаете распечатать их с разделителями в виде знака табуляции, исключите эту опцию из командной строки);
- вместо *dbname* подставьте имя реальной базы данных, с которой планируете работать;
- вместо *sql_script* подставьте абсолютное полное или относительное полное имя реального текстового файла, в котором содержится программа SQL для запуска в режиме сценария (см. рис. 1.19).

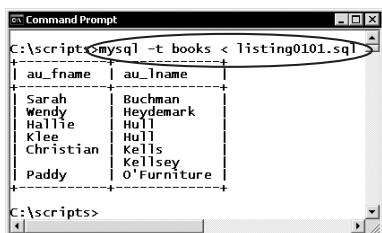


Рис. 1.19. Результат выполнения команды SELECT утилитой командной строки mysql в режиме сценария

Выход из утилиты командной строки mysql

Напечатайте `exit` или `quit` и нажмите **Enter**.

Вывод на экран списка опций утилиты командной строки mysql

Нужно выполнить следующие действия:

1. В командной строке наберите `mysql -?` и нажмите **Enter**.
2. Эта команда осуществляет выход в виде нескольких экранов, и для того, чтобы просматривать их, переключая один за другим по своему желанию, введите дополнительные опции в командной строке: `mysql -? | more` (см. табл. 1), нажмите **Enter** и переключайте экраны с распечаткой выхода нажатием клавиши пробела (см. рис. 1.20).

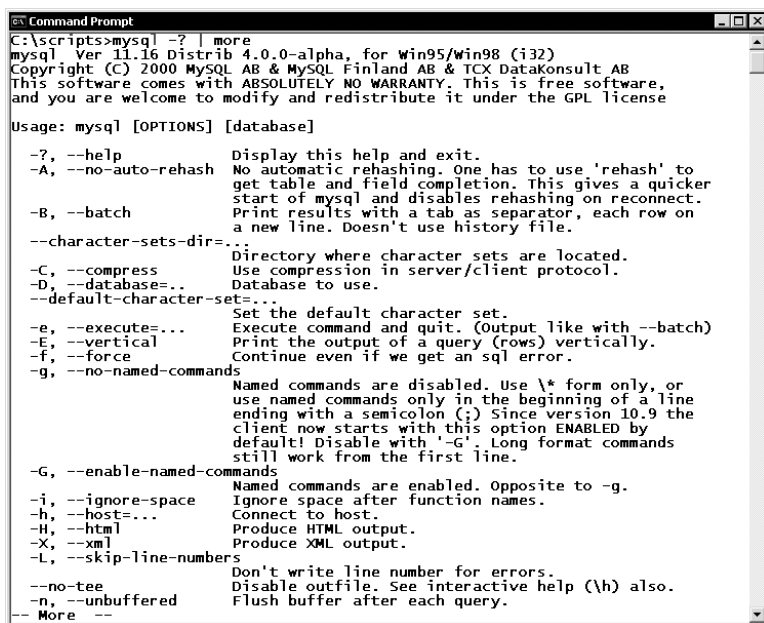


Рис. 1.20. Экран справочной системы mysql

П

Если вы работаете с MySQL по сети с удаленного компьютера, то для того, чтобы подключиться к серверу, понадобится указать имя хост-машины, имя пользователя и пароль. Поэтому наберите `mysql -h host -u user -p dbname`, где предполагается:

- что вместо *host* вы подставите имя реальной хост-машины;
- что вместо *user* вы подставите имя пользователя;
- что вместо *dbname* вы подставите имя базы данных, с которой хотите работать.

П

При таком наборе опций MySQL запросит у вас пароль. Выясните все параметры, необходимые для подключения, у администратора базы данных.

П

При запуске сервера MySQL, то есть `mysqld`, вы можете воспользоваться опцией `-ansi`, чтобы применять не синтаксис MySQL, а синтаксис ANSI SQL.

PostgreSQL

PostgreSQL – еще одна ведущая СУБД с открытым исходным кодом. Она отличается быстродействием, устойчивостью и возможностью поддерживать базы данных больших объемов с огромным числом транзакций, а также обширной функциональностью и высоким уровнем соответствия стандарту ANSI SQL. Немаловажно и то, что PostgreSQL работает под многими популярными операционными системами и на разных аппаратных платформах, будучи при этом бесплатной. Эту СУБД можно скачать с сайта www.postgresql.org.

В настоящем издании рассматривается версия PostgreSQL 7.1. Чтобы определить, с какой версией этой СУБД вы работаете, напечатайте в командной строке `psql -V` и нажмите **Enter**.

Чтобы исполнять программы SQL, можно использовать утилиту командной строки `psql`.

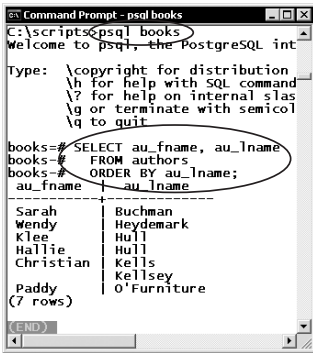


Рис. 1.21. Результат интерактивного выполнения команды `SELECT` утилитой командной строки `psql`

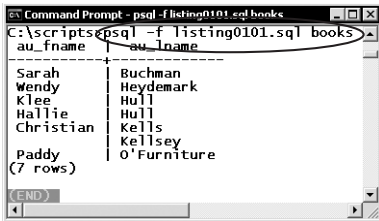


Рис. 1.22. Результат выполнения утилитой `psql` команды `SELECT` в режиме сценария

Использование утилиты командной строки `psql` в интерактивном режиме

Порядок действий:

1. В командной строке напечатайте `psql dbname`. Предполагается, что вместо `dbname` вы подставите имя базы данных, с которой собираетесь работать.
2. Введите какую-нибудь команду SQL, например ту, которая приведена в листинге 1.1. Эта команда может занять несколько строк.
3. Нажмите **Enter**. На экране появятся результаты (см. рис. 1.21).

Использование утилиты командной строки `psql` в режиме сценария

В командной строке напечатайте `psql -f sql_script db_name`.

Предполагается, что:

- вместо `sql_script` вы подставите абсолютное полное или относительное полное имя текстового файла, в котором содержится некая программа SQL;
- вместо `db_name` вы подставите имя базы данных, с которой собираетесь работать (см. рис. 1.22).

Выход из утилиты командной строки `psql`

Необходимо напечатать `\q` и нажать **Enter**.

Вывод на экран списка опций командной строки psql

В командной строке нужно напечатать `psql -?` и нажать **Enter** (см. рис. 1.23).

П

Если вы работаете в PostgreSQL по сети с удаленного компьютера, то для подключения к серверу понадобится указать имя хост-машины, имя пользователя и пароль. Поэтому напечатайте `psql -h host -U user -W dbname`, где предполагается, что:

- вместо `host` вы подставите имя реальной хост-машины;
- вместо `user` вы подставите имя пользователя;
- вместо `dbname` вы подставите имя базы данных, с которой хотите работать.

П

При таком наборе опций PostgreSQL запросит у вас пароль. Выясните все параметры, необходимые для подключения, у администратора базы данных.

```

C:\scripts>psql -?
This is psql, the PostgreSQL interactive terminal.

Usage:
psql [options] [dbname [username]]

Options:
-a                Echo all input from script
-A              Unaligned table output mode (-P format=unaligned)
-c <query>       Run only single query (or slash command) and exit
-d <dbname>      Specify database name to connect to (default: chris)
-e              Echo queries sent to backend
-E              Display queries that internal commands generate
-f <filename>    Execute queries from file, then exit
-F <string>      Set field separator (default: "|") (-P fieldsep=)
-h <host>        Specify database server host (default: domain socket)
-H              HTML table output mode (-P format=html)
-l              List available databases, then exit
-n              Disable readline
-o <filename>    Send query output to filename (or |pipe)
-p <port>        Specify database server port (default: hardwired)
-P var=[arg]     Set printing option 'var' to 'arg' (see \pset command)
-q              Run quietly (no messages, only query output)
-R <string>      Set record separator (default: newline) (-P recordsep=)
-s              Single step mode (confirm each query)
-S              Single line mode (newline terminates query)
-t              Print rows only (-P tuples_only)
-T text         Set HTML table tag options (width, border) (-P tableattr=)
-U <username>    Specify database username (default: chris)
-v name=val     Set psql variable 'name' to 'value'
-V              Show version information and exit
-W              Prompt for password (should happen automatically)
-x              Turn on expanded table output (-P expanded)
-X              Do not read startup file (~/.psqlrc)

For more information, type "??" (for internal commands) or "\help"
(for SQL commands) from within psql, or consult the psql section in
the PostgreSQL manual, which accompanies the distribution and is also
available at <http://www.postgresql.org>.
Report bugs to <pgsql-bugs@postgresql.org>.
  
```

Рис. 1.23. Экран справочной системы psql

РЕЛЯЦИОННАЯ МОДЕЛЬ

2

В последнее время появилось очень много хороших книг по разработке баз данных. Настоящее издание к ним не относится. Дело, конечно, не в том, что оно плохое, а в том, что не посвящено проектированию баз данных. Но эта книга о языке SQL, который без реляционных баз данных не имеет никакого смысла. Следовательно, чтобы стать настоящим программистом SQL, вам необходимо познакомиться

с реляционной моделью (см. рис. 2.1), которая оказалась настолько простой и хорошо приспособленной для решения задач организации данных и управления ими, что конкурирующие сетевая и иерархическая модели в конце концов потерпели фиаско. Основой реляционной модели является математическая теория множеств, которая требует от вас мыслить категориями множеств данных, а не категориями

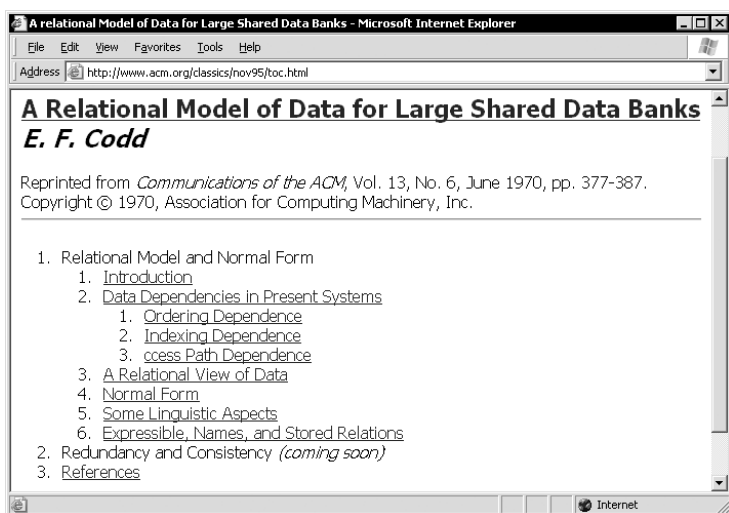


Рис. 2.1. Вы можете ознакомиться с описанием реляционной модели, созданной Е. Ф. Коддом, по адресу: <http://www.acm.org/classics/nov95/toc.html>. Все современные системы управления реляционными базами данных основаны на модели данных, определенной этим документом

отдельных элементов или строк данных. Итак, будучи основана на математической теории множеств, реляционная модель строго регламентирует, как выполнять алгебраические операции математической теории множеств, в частности объединение и пересечение, но на таблицах баз данных. И эти операции выполняются в принципе так же, как они определены для математической теории (см. рис. 2.2). При этом понятно, что таблицы являются аналогами множеств, то есть таблицы в реляционной модели играют ту же роль, какую в теории множеств играют множества. Таким образом, таблицы, подобно абстрактным множествам, являются некими совокупностями абсолютно различных элементов, но имеющих какие-то общие свойства. Отсюда следует, что, хотя некое математическое множество могло бы, например, иметь в качестве элементов положительные целые числа, а какая-нибудь таблица произвольной базы могла бы содержать информацию о студентах, между этим множеством и этой таблицей нет принципиальной разницы.

Звучит фундаментально, но, по правде говоря, вы вполне можете пропустить большую часть этой главы или всего лишь просмотреть ее. Дело в том, что научиться программировать на SQL можно и без серьезной теории, используя так называемый метод проб и ошибок. Это особенно верно в том случае, если ваш интерес не выходит за пределы простых команд SELECT. Но, как это имеет место и во всех других областях знаний, мощь теории состоит не в том, чтобы помочь в решении простых задач, а в том, что, если вы ее глубоко понимаете, у вас появляется возможность предсказывать результаты решения задач. Если же вы умеете предсказывать результаты, вам не придется бесконечно долго и нудно выяснять причины сбоев методом проб и ошибок. Вы будете сразу точно представлять, что именно работает не так.

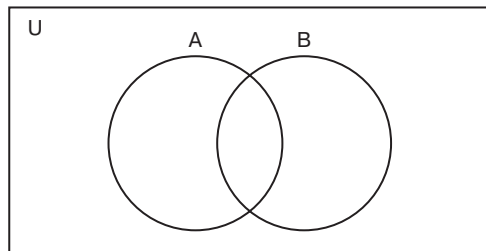


Рис. 2.2. Эта диаграмма описывает результат операций над множествами. Прямоугольник U – это универсум, а круги A и B – множества определенных элементов. Положение друг относительно друга и пересечение множеств A и B определяют отношения между ними. В реляционной модели круги представляют собой таблицы, а прямоугольник – всю информацию, находящуюся в базе данных

Таблица 2.1. Эквивалентность терминов

Модель	SQL	Файловая система
Отношение	Таблица	Файл
Атрибут	Столбец	Поле
Кортеж	Строка	Запись

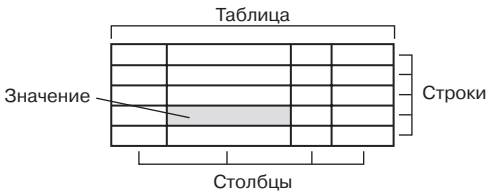


Рис. 2.3. Эта сетка является абстрактным представлением концепции таблицы, то есть того фундаментального элементарного модуля хранения данных, из которых построена любая реляционная база данных

au_id	au_fname	au_lname
-----	-----	-----
A01	Sarah	Bucman
A02	Wendy	Heydemark
A03	Hallie	Hull
A04	Klee	Hull

Рис. 2.4. Эта сетка является отображением некой реальной, а не абстрактной, как было выше, таблицы. Поэтому она приведена в том виде, в каком отображается любая таблица в СУБД и книгах. Мы видим, что эта таблица имеет 3 столбца, 4 строки и (3×4) 12 значений, а также то, что верхняя строка является заголовком, состоящим из имен столбцов

Таблицы, столбцы и строки

Начнем с терминологии. Если вы обладаете некоторым опытом работы с базами данных, то наверняка уже заметили, что в этой сфере деятельности одни и те же понятия часто называют разными терминами. Поэтому в табл. 2.1 показана взаимосвязь основных терминов и понятий. Здесь в первом столбце приведены термины, относящиеся к реляционной модели, во втором столбце – термины стандарта ANSI SQL и современной документации СУБД, а в третьем – термины файловой модели обработки данных. В настоящем издании используется терминология стандарта SQL, в том числе при обсуждении реляционной модели.

Таблицы

С точки зрения пользователя, любая база данных – это совокупность одной или нескольких таблиц, и ничего кроме таблиц. По определению любая *таблица*:

- является тем компонентом структуры базы данных, в котором содержатся собственно данные;
- содержит данные, соответствующие какому-то заданному *объектному типу*, где под ним понимается некий класс различных объектов, или/и событий, или/и концепций реального мира с общими свойствами (например, студенты, кинофильмы, гены, пациенты, погодные условия или бизнес-встречи – это разные объектные типы, поэтому вам следует хранить данные о, скажем, пациентах и бизнес-встречах в разных таблицах);
- является двумерной сеткой, образованной пересечением строк и столбцов (см. рис. 2.3 и 2.4);

- содержит на каждом пересечении строки и столбца некий элемент данных, называемый *значением* (см. рис. 2.3 и 2.4);
- всегда содержит не менее одного столбца, но может не содержать ни одной строки (всякая таблица, не имеющая строк, называется *пустой таблицей*);
- имеет имя, которое внутри любой базы данных, содержащей данную таблицу, является уникальным.

Столбцы

Столбцы любой заданной таблицы отличаются следующими свойствами:

- каждый столбец представляет какой-то конкретный атрибут (свойство) того объектного типа, который отображается этой таблицей (например, в таблице `employees` некий столбец по имени `hire_date` мог бы содержать даты найма сотрудников на работу);
- для каждого столбца определен свой домен, под которым понимается та система формальных ограничений на тип данных, их длину, формат, диапазон изменения, единственность и отсутность значения (`null`), которая задает множество допустимых значений для этого столбца (например, вы не можете поместить строковое значение `mimetic` в столбец `hire_date`, если для него допустимыми являются только значения данных с типом дата);
- данные, вводимые в любую позицию столбца, являются однозначными (атомарными, неделимыми; см. раздел «Нормализация» в этой главе);
- порядок следования столбцов (при просмотре таблицы слева направо) не имеет значения (см. рис. 2.5);

au_lname	au_id	au_fname
-----	-----	-----
Hull	A04	Klee
Buchman	A01	Sarah
Hull	A03	Hallie
Heydemark	A02	Wendy

Рис. 2.5. Считается, что строки и столбцы *не упорядочены*, если последовательность их прохождения при просмотре произвольной таблицы не имеет значения ни для смысла той информации, которую содержит эта таблица, ни для тех операций, которые могут быть проведены с этой таблицей в рамках реляционной модели. Значит, перестановки строк и столбцов не меняют смысла таблицы. Таким образом, таблица, представленная на этом рисунке, полностью эквивалентна таблице на рис. 2.4

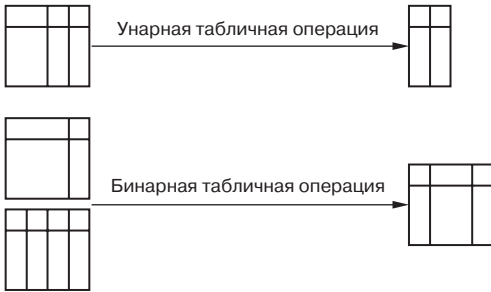


Рис. 2.6. Замкнутость таблиц гарантирует вам, что вы всегда будете получать определенную таблицу независимо от того, какую операцию над таблицами вы проводите. Это свойство позволяет вкладывать табличные операции друг в друга, причем глубина вложения не ограничена. При этом принято различать так называемые *унарные* и *бинарные* табличные операции, отличающиеся тем, что на вход и выход произвольной унарной операции поступает только по одной таблице, а на вход любой бинарной операции должны поступить две каких-нибудь таблицы, хотя на выходе этой бинарной операции тоже будет лишь одна какая-то таблица

- у каждого столбца есть имя, которое идентифицирует его в пределах своей таблицы (однако в других таблицах вы вполне можете применять то же самое имя столбца).

Строки

Перечислим характеристики строк в произвольной заданной таблице:

- каждая строка описывает какой-нибудь объект (реального мира), являющийся уникальной реализацией объектного типа (например, какой-нибудь студент или определенная бизнес-встреча);
- каждая строка в каждом столбце содержит или какое-нибудь значение (ненулевое), или элемент null;
- порядок следования строк при просмотре таблицы (скажем, сверху вниз) не имеет значения;
- в любой таблице в любой паре строк найдется хотя бы один столбец, значения в котором и в строках этой пары различны;
- в любой таблице каждая строка однозначно идентифицируется своим первичным ключом (см. раздел «Первичные ключи» в этой главе).

С

Для выборки строк и столбцов применяйте команду `SELECT` (см. главы 4–8). Чтобы удалять, добавлять и редактировать строки, применяйте команды `INSERT`, `UPDATE` и `DELETE` (см. главу 9), а чтобы добавлять, редактировать и удалять столбцы, используйте команды `CREATE TABLE` и `ALTER TABLE` (см. главу 10).

П

В рамках реляционной модели все таблицы обладают весьма привлекательным свойством замкнутости, которое состоит в том, что в результате произвольной операции реляционной модели, произведенной над любой таблицей, обязательно получится какая-нибудь таблица (см. рис. 2.6).

П

Любая СУБД применяет таблицы двух типов:

- таблицы пользователя;
- таблицы системы, или системные таблицы.

Разница между этими типами таблиц заключается в следующем. *Системные таблицы* содержат *метаданные*, то есть данные о самой базе данных (например, сведения о структуре базы, некоторые физические особенности, статистику работы базы, установки по обеспечению безопасности данных и системы). Совокупность системных таблиц во всей используемой версии СУБД называется *системным каталогом СУБД*. Любая СУБД динамически создает и обновляет системные таблицы, что, кстати сказать, полностью отвечает следующему требованию реляционной модели: *все данные* должны храниться в таблицах (см. рис. 2.7).

Таблицы пользователя, или пользовательские таблицы, содержат данные, которые пользователь определяет и заносит в них.

П

Число строк в любой работающей таблице меняется очень часто, а число столбцов — редко. Дело в том, что изменения столбцов могут коснуться ключей, повлиять на ссылочную целостность базы, на привилегии пользователей и т.д. В то же время ни добавление, ни удаление строк ни на что подобное не влияют.

П

Распределение данных по столбцам таблиц зависит от мастерства разработчика базы. Дело в том, что разработчики часто разбивают значения по столбцам на основе своего собственного понимания потребностей пользователя. Например, телефонные номера могли бы находиться в единственном столбце `phone_number` или в нескольких столбцах *наподобие* `country_code`, `area_code` и `subscriber_number`.


























Name	Owner	Type ▾
 authors	dbo	User
 publishers	dbo	User
 royalties	dbo	User
 title_authors	dbo	User
 titles	dbo	User
 dtproperties	dbo	System
 syscolumns	dbo	System
 syscomments	dbo	System
 sysdepends	dbo	System
 sysfilegroups	dbo	System
 sysfiles	dbo	System
 sysfiles1	dbo	System
 sysforeignkeys	dbo	System
 sysfulltextcatalogs	dbo	System
 sysfulltextnotify	dbo	System
 sysindexes	dbo	System
 sysindexkeys	dbo	System
 sysmembers	dbo	System
 sysobjects	dbo	System
 syspermissions	dbo	System
 sysproperties	dbo	System
 sysprotects	dbo	System
 sysreferences	dbo	System
 systypes	dbo	System
 sysusers	dbo	System

Рис. 2.7. Все СУБД хранят системную информацию в специальных таблицах, которые тоже называются системными. Здесь мы видим системные таблицы, создаваемые, сохраняемые и поддерживаемые для типовой базы данных СУБД Microsoft SQL Server, которая описана в настоящем издании. Несмотря на то что вы имеете такой же доступ и можете обращаться с системными таблицами абсолютно так же, как с пользовательскими, мы не советуем делать это без крайней необходимости. Системные таблицы можно трогать только тогда, когда вы предельно ясно представляете себе, чего добиваетесь

П

Имейте в виду, что похожесть таблиц баз данных и электронных таблиц – исключительно внешняя. А отличия таблицы базы от электронной таблицы, наоборот, фундаментальные потому, что никакая таблица любой реляционной базы:

- не зависит от порядка строк и столбцов;
- сама не производит никаких вычислений;
- не позволяет вводить данные в свободном формате (наоборот, соответствие вводимых данных допустимому формату всегда строго проверяется).

К тому же всякую таблицу легко можно связать с другими таблицами.

П

Имейте в виду, что прилагательное «реляционная» (relational), которое входит в название «реляционная база данных» (relational database), появилось благодаря математической «реляционной теории множеств» (relational set theory), а не из-за возможности устанавливать связь (to relate) между разными таблицами по их общим значениям.

П

Рекомендуем запомнить следующие термины:

- число строк любой таблицы называется *мощностью* этой таблицы;
- число столбцов любой таблицы называется *порядком* этой таблицы;
- говоря о базах данных, термин «схема» чаще всего применяют для того, чтобы обозначить структуру и организацию какой-нибудь базы данных, включая определение ее таблиц. Считается, что схема описывает все. Но по стандарту ANSI любая «схема» – это совокупность таких объектов базы данных, которые принадлежат какому-то одному пользователю;
- совокупность всех схем называется *каталогом* (имеется в виду стандарт ANSI).

Первичные ключи

Очевидно, что каждая база данных должна работать так, чтобы можно было обратиться к произвольному значению, хранящемуся в любой из ее таблиц. Но, поскольку все значения размещаются в таблицах, точнее – на пересечениях строк и столбцов этих таблиц, очевидно и то, что местоположение любого значения однозначно определяется, если известны конкретные таблица, столбец и строка. Не менее очевидно, что любые таблицу и столбец можно однозначно идентифицировать по их уникальным номерам. Однако строки уникальных имен не имеют. Тем не менее нам нужен механизм их однозначной идентификации. Такой механизм в реляционной модели существует и называется *первичный ключ* (Primary key). Поскольку, как мы видим, первичный ключ – это некая функция на строках произвольной таблицы, ее удобно связать со столбцами этой таблицы, которые тоже принимают свои значения на строках, то есть являются их функциями. Поэтому первичный ключ надо представлять себе как столбец или, в крайнем случае, набор столбцов особого рода.

Перечислим родовидовые признаки первичного ключа как механизма однозначной идентификации строк:

- *необходимость*. Каждая таблица должна иметь только один первичный ключ. Вспомните, что реляционная модель рассматривает любую таблицу как некое неупорядоченное множество строк. Поскольку для такого множества не существует понятий «следующая строка» или «предыдущая строка», вы не можете идентифицировать никакую строку по ее относительному расположению среди других строк. Если бы не было первичных ключей, некоторые данные стали бы недоступными;

- *уникальность*. Поскольку, как мы уже знаем, в любой таблице первичный ключ однозначно идентифицирует каждую строку, никакие две строки этой таблицы не могут иметь одно и то же значение первичного ключа (то есть первичный ключ не может на разных строках одной таблицы принимать одинаковые значения);
- *простой или составной*. Итак, в любой таблице ее единственный первичный ключ может включать один или более столбцов этой таблицы. Если первичному ключу соответствует ровно один столбец, то такой ключ называют *простым*. Любой первичный ключ, не являющийся простым и, следовательно, включающий два или более столбцов, называют *составным*;
- *никогда не принимает значение null*. Ни одно значение первичного ключа не может быть пустым. Отсюда для составного первичного ключа получаем, что ни одно значение столбцов, которые входят в составной первичный ключ, не может быть пустым (см. раздел «Значение null» в главе 3);
- *неизменность*. Будучи однажды создан, всякий первичный ключ крайне редко меняет значения, которые он принимает на строках;
- *одноразовость*. Если вы удалите какую-нибудь строку, то не сможете присвоить любой другой строке то значение, которое первичный ключ принимал на удаленной строке;
- *минимальность*. Для любой таблицы первичный ключ содержит только тот столбец (столбцы), который (которые) необходим (необходимы), чтобы выполнить свойство уникальности этого первичного ключа.

При создании любой базы данных ее разработчики назначают первичные ключи всем таблицам. Этот процесс очень важен. Дело в том, что, если первичные ключи будут созданы неудачно, добавлять и модифицировать данные (то есть новые строки) в какую-нибудь таблицу будет нельзя.

В данной главе мы приводим лишь краткий обзор основных принципов и подходов разработки баз данных. Поэтому для более подробного изучения этого вопроса вам следует обратиться к какому-нибудь специальному руководству.

Допустим, вы хотите выбрать первичный ключ для той таблицы, которая представлена на рис. 2.8. Ни столбец `au_fname`, ни столбец `au_lname`, взятые по отдельности, не подходят, потому что каждый из них, будучи назначен простым первичным ключом, нарушал бы постулат уникальности. Но и объединение этих столбцов в составной первичный ключ не дало бы решения по той же причине. Дело в том, что люди очень часто имеют одинаковые имена, что явилось бы потенциальным источником нарушения постулата уникальности, выбери мы имя, полное или нет, в качестве первичного ключа, простого или составного. Кроме того, люди могут менять свои имена как по собственному желанию (в частности, при разводе или вступлении в брак), так и по воле случая (написание имени может измениться). То же самое справедливо и по отношению к именам юридических лиц: компании могут иметь одинаковые имена, могут сливаться в один холдинг с новым именем, разделяться на новые компании, менять форму собственности и т.д. Именно потому, что имена так нестабильны (то есть несут в себе потенциальную угрозу для постулата неизменности), опыт проектирования баз данных говорит о том, что собственно имена – плохая ос-

au_id	au_fname	au_lname
-----	-----	-----
A01	Sarah	Buchman
A02	Wendy	Heydemark
A03	Hallie	Hull
A04	Klee	Hull

Рис. 2.8. Здесь первичным ключом является `au_id`

нова для построения первичного ключа. В рассматриваемом примере правильное решение – это столбец `au_id`, который я специально добавил для того, чтобы идентифицировать авторов (то есть не имена, а людей). Разработчики баз данных часто прибегают к этому приему. Если естественные или «очевидные» идентификаторы (вроде имен) не годятся, разработчик изобретает уникальный идентификатор.

После того как таблице назначен какой-нибудь первичный ключ, СУБД начинает жестко отслеживать целостность данных этой таблицы. Например, в нашем примере вы не смогли бы ввести в рассматриваемую таблицу строку

A02 Christian Kells

потому что в таблице уже есть строка, где значение столбца `au_id` равно A02. Ввести в нее строку

NULL Christian Kells

также не получится, потому что столбец `au_id`, будучи первичным ключом, не может принимать значение `null`. Допустимой является строка

A05 Christian Kells

С

Чтобы назначать первичные ключи, применяйте ограничение `PRIMARY KEY` (см. раздел «Задание первичного ключа с помощью ограничения `PRIMARY KEY`» в главе 10).

П

Опытные разработчики баз данных ставят первичный ключ на место первого столбца (столбцов, если ключ – составной) в таблице, хотя теоретически порядок следования столбцов не имеет, как мы теперь знаем, никакого значения. Поэтому имеет смысл всегда начинать поиск первичного ключа с первых столбцов. Кроме того, если имя какого-то столбца содержит сочетания «`id`», «`code`» или «`num`», это свидетельствует о том, что данный столбец может быть ключевым (то есть столбцом или первичного, или внешнего ключа; см. следующий раздел).

П

Разработчики баз данных часто игнорируют общепринятые уникальные идентификаторы, какими, например, являются номер полиса социального страхования гражданина США и ISBN книг, в качестве кандидатов на роль ключей. Вместо них обычно применяются самодельные ключи, которые каким-либо незатейливым образом кодируют информацию, имеющую смысл только для той организации, где работает пользователь конкретной проектируемой базы данных. Например, в качестве формата значений ключевого столбца, содержащего уникальные идентификаторы служащих организации, можно назначить произвольный ряд цифр, букв и знаков, в который по определенному правилу встроена, скажем, дата приема сотрудника на работу.

П

У разработчика базы данных при назначении первичного ключа может быть выбор из нескольких кандидатов, одинаково удовлетворяющих постулатам первичного ключа. В результате только один столбец из них становится первичным ключом, а все остальные – *альтернативными ключами*.

П

В рассмотренном выше примере вы могли бы совместно применить столбцы `au_id` и, скажем, `au_lname` в качестве составного первичного ключа, но это было бы нарушением постулата минимальности.



Коммерческие СУБД предоставляют в распоряжение разработчика специальный механизм автоматического построения однозначного идентификатора строки в виде специальных атрибутов или таких типов данных, которые автоматически присваивают уникальный идентификатор строки, будучи включенными в состав столбцов произвольной таблицы. Идея здесь в том, что, например, счетчик автоматически увеличивает себя на единицу при появлении каждой новой строки (то есть, строго говоря, в каждой строке есть столбец-счетчик, значение которого в этой строке равно ее очереди внесения в таблицу). В Microsoft Access таким специальным типом данных является `AutoNumber`, в Microsoft SQL Server – тип данных `uniqueidentifier` или свойство `IDENTITY`, в Oracle – тип данных `ROWID`, в MySQL – атрибут `AUTO_INCREMENT`, в PostgreSQL – тип данных `serial`.

Внешние ключи

Поскольку, как мы договорились, разные таблицы содержат информацию о разных объектных типах, у нас должен быть способ осмысленного перемещения от таблицы к таблице. Так вот, рассматриваемая модель предоставляет нам такой механизм – *внешний ключ* (или *вторичный ключ* – Foreign key), который применяют для соединения одной произвольной таблицы с другой. Любой внешний ключ обладает следующими свойствами:

- является или одним столбцом, или группой столбцов в некоторой таблице, но таким столбцом или группой таких столбцов, что их значения или связаны со значениями столбцов другой какой-то таблицы, или содержат ссылки на значения столбцов этой таблицы;
- позволяет строкам из одной какой-нибудь таблицы иметь соответствующие им строки в другой таблице;
- таблица, которая содержит внешний ключ, называется *указывающей*, или *дочерней*, а таблица, на которую этот внешний ключ указывает, – *указываемой*, или *материнской*;
- устанавливает прямую взаимосвязь с первичным (или альтернативным) ключом материнской таблицы. Поэтому внешний ключ может принимать только те значения, которые уже принял первичный (или альтернативный) ключ в материнской таблице, или специальное значение NULL, если соответствующим столбцам разрешается его принимать. Это ограничение называется *ссылочной целостностью*. Например, любая строка в таблице appointments, содержащей, скажем, специальную информацию о визите к врачу, должна иметь связанную с ней строку в материнской

таблице patients, где находится, предположим, специальная информация о тех пациентах, которые, кроме всего прочего, совершают визиты к врачу. В противном случае появились бы визиты к врачу несуществующих пациентов или визиты неизвестно кого. Любая строка произвольной дочерней таблицы, не имеющая связанной строки в материнской таблице, называется *висячей*;

- все значения любого внешнего ключа ограничены тем же самым доменом, который наложен на соответствующий ключ родительской таблицы. В разделе «Таблицы, столбцы и строки» мы определили домен как множество допустимых значений столбца;
- в отличие от первичного ключа, значениями внешнего ключа могут быть значения NULL (то есть значения внешнего ключа могут быть пустыми или, иначе говоря, обнуляемыми; см. советы в конце этого раздела);
- может иметь имя, отличное от имени родительского ключа;
- значения внешнего ключа в общем случае не являются уникальными в собственной таблице;
- в самом первом пункте этого списка есть одно упрощение. Дело в том, что любой внешний ключ может ссылаться на первичный ключ своей собственной таблицы, а не только на первичный ключ какой-нибудь другой таблицы. Всякая таблица, где имеет место такая ситуация, называется *ссылающейся на себя*. Например, таблица employees, включающая информацию о сотрудниках организации, с первичным ключом employee_id, значениями которого являются уникальные идентификаторы сотрудников, вполне может

содержать некий внешний ключ `boss_id`, чтобы он:

- отражал информацию о начальниках тех сотрудников, которые идентифицируются значениями столбца `employee_id`;
- ссылался на столбец `employee_id` просто потому, что любой начальник обязательно является сотрудником организации (и чьим-то подчиненным, между прочим).

Разбираясь с понятием внешнего ключа, внимательно рассмотрите рис. 2.9, где наглядно показано, какую роль играют материнский первичный и дочерний внешние ключи в организации связи между материнской и дочерней таблицами.

Итак, вы определили некий внешний ключ. Теперь СУБД станет жестко отслеживать целостность на уровне ссылок (или ссылочную целостность). Например, вы не смогли бы вставить следующую строку в дочернюю таблицу `titles`, содержащую информацию о названиях книг, потому что в материнской таблице `publishers` с информацией об издательствах в столбце первичного ключа `pub_id` нет значения P05:

T05 Exchange of Platitudes P05

Разрешается вставить следующую строку, только если поле таблицы `titles`, по которому построен внешний ключ, может принимать значение `null`:

T05 Exchange of Platitudes NULL

Эта строчка полностью корректна:

T05 Exchange of Platitudes P04

С

Чтобы указать произвольный внешний ключ, применяйте ограничение `FOREIGN KEY` (см. раздел «Задание внешнего ключа с помощью ограничения `FOREIGN KEY`» в главе 10).

Первичный ключ

publishers

pub_id	pub name
P01	Abatis Publishers
P02	Core Dump Books
P03	Schadenfreude Press
P04	Tenterhooks Press

Первичный ключ Внешний ключ

titles

title_id	title_name	pub_id
T01	1977!	P01
T02	200 Years of Ger...	P03
T03	Ask Your System...	P02
T04	But I Did It Unco...	P04

Рис. 2.9. В таблице `titles` столбец `pub_id` является внешним ключом, который ссылается на столбец `pub_id`, являющийся первичным ключом таблицы `publishers`

П

SQL дает возможность программисту выполнить те действия по сохранению ссылочной целостности, которые СУБД предпринимает тогда, когда пользователь пытается удалить или модифицировать любое ключевое значение, на которое указывают значения внешних ключей (см. примечания в разделе «Задание внешнего ключа с помощью ограничения FOREIGN KEY» в главе 10).

П

Решение позволить полям внешнего ключа принимать значения null не следует из теории, а является чисто практическим. Дело в том, что отсутствие некоторых значений во внешних ключах затруднило бы поддержание ссылочной целостности. В то же время значения null во внешних ключах остаются таковыми (то есть значениями null) только временно в преддверии принятия какого-нибудь решения или/и выяснения каких-то фактов (см. раздел «Значение null» в главе 3)¹.

С

Во избежание недоразумений не называйте внешние ключи указателями или связями (link). Под указателями принято понимать то, что ссылается на какие-то места в физической памяти. В то же время внешние ключи связывают таблицы на основе значений данных. Разница в том, что внешние ключи – это логические, а не физические объекты.

¹ Зачастую значение NULL в полях внешнего ключа позволяет создавать таблицы, в которых различные строки связаны с разными таблицами. Например, таблица customers представляет собой обобщенный список покупателей и содержит два поля org_id и person_id, указывающие на таблицы organization и person соответственно, где находится информация о покупателях-организациях и покупателях – частных лицах. В этом случае в каждой строке таблицы customers одно из полей org_id или person_id всегда принимает значение NULL, а второе – указывает на соответствующую запись в родительской таблице. – *Прим. науч. ред.*

Связи

Любая связь – это соединение, устанавливаемое между общими столбцами двух разных таблиц. Различают следующие категории связей:

- один-к-одному;
- один-ко-многим;
- многие-ко-многим.

Один-к-одному

Считается, что между таблицами А и В существует связь «один-к-одному», если каждая строка таблицы А может иметь *не более одной* соответствующей ей строки таблицы В, и каждая строка таблицы В может иметь *не более одной* соответствующей ей строки таблицы А.

С теоретической точки зрения нет никакой разницы при хранении данных двух таблиц, между которыми существует связь «один-к-одному», в разных двух таблицах или в одной общей таблице. Следовательно, такая связь вроде бы не нужна. Именно поэтому связь «один-к-одному» применяют не из теоретических, а из практических соображений: или/и по соображениям безопасности, когда надо выделить и защитить конфиденциальную информацию, или/и по соображениям быстродействия системы, когда надо разбить слишком громоздкие монолитные таблицы, или/и для того, чтобы, например, не вставлять значения null в таблицы, отдельные столбцы которых содержат непустые (то есть известные) значения лишь для весьма небольшого подмножества.

Таким образом, способ установления связи «один-к-одному» заключается в том, что первичный ключ некой таблицы объявляется ее внешним ключом, указывающим на первичный ключ какой-нибудь другой таблицы (см. рис. 2.10 и 2.11).

titles	
title_id	title_name
T01	1977!
T02	200 Years of Ger...
T03	Ask Your System...
T04	But I Did It Unco...

royalties	
title_id	advance
T01	10000
T02	1000
T03	15000
T04	20000

Рис. 2.10. Пример связи «один-к-одному». Каждая строка таблицы titles имеет *не более одной* соответствующей ей строки таблицы royalties, и каждая строка таблицы royalties имеет *не более одной* соответствующей ей строки таблицы titles. Здесь первичный ключ таблицы royalties является ее внешним ключом, указывающим на первичный ключ таблицы titles



Рис. 2.11. Еще один способ изобразить связь, показанную на рис. 2.10. Связанные столбцы соединены линией, а значок ключа выделяет первичный ключ

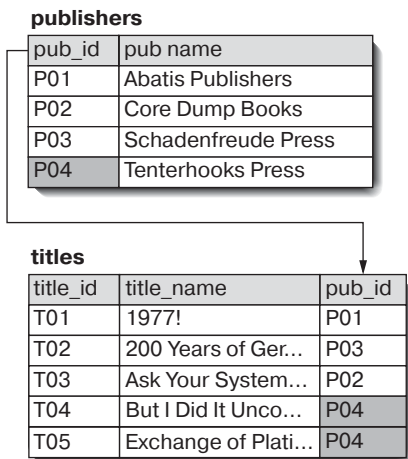


Рис. 2.12. Пример связи «один-ко-многим». Каждой строке в таблице `publishers` может соответствовать несколько строк в таблице `titles`, но каждая строка в таблице `titles` имеет только одну соответствующую строку в таблице `publishers`. Здесь мы видим, что первичный ключ таблицы `publishers`, строки которой могут участвовать в связи строго по одной, объявлен внешним ключом в таблице `titles`, строки которой могут участвовать в связи по нескольку штук

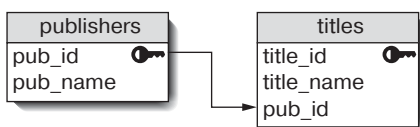


Рис. 2.13. Пример еще одного способа изобразить связь «один-ко-многим», показанную на рис. 2.12. Линия со стрелкой направлена от таблицы `publishers`, строки которой могут участвовать в связи строго по одной, к таблице `titles` (на нее указывает стрелка), строки которой могут участвовать в связи по нескольку штук. Пиктограмма ключа по-прежнему выделяет первичный ключ

Один-ко-многим

Считается, что между таблицами А и В установлена связь «один-ко-многим», если каждая строка в таблице А имеет *много* (то есть ноль, одну или много) соответствующих строк таблицы В, а каждая строка таблицы В – всего *одну* соответствующую строку таблицы А. Например, можно представить себе ситуацию, когда некий издатель может печатать разные книги, но каждая книга может быть выпущена только одним издателем.

Таким образом, способ образования связи «один-ко-многим» состоит в том, чтобы первичный ключ таблицы, строки которой могут участвовать в связи только по одной, объявить внешним ключом в той таблице, строки которой могут участвовать в связи по нескольку штук (см. рис. 2.12 и 2.13).

Многие-ко-многим

Считается, что между таблицами А и В установлена связь «многие-ко-многим», если каждая строка в таблице А может иметь много (то есть или ни одной, или несколько) соответствующих строк в таблице В, а каждая строка в таблице В может иметь много (то есть или ни одной, или несколько) соответствующих строк в таблице А.

Единственный способ установить связь «многие-ко-многим» между двумя произвольными таблицами заключается в том, чтобы создать третью таблицу (называемую стыковочной), в которой ее составной первичный ключ является комбинацией первичных ключей обеих таблиц со связью «один-ко-многим». При этом каждый из столбцов составного ключа по отдельности является внешним ключом. Этот прием построения составного ключа всегда дает

для каждой строки стыковочной таблицы какое-то уникальное значение и разбивает первоначальную связь «многие-ко-многим» на две независимые связи «один-ко-многим» (см. рис. 2.14 и 2.15).

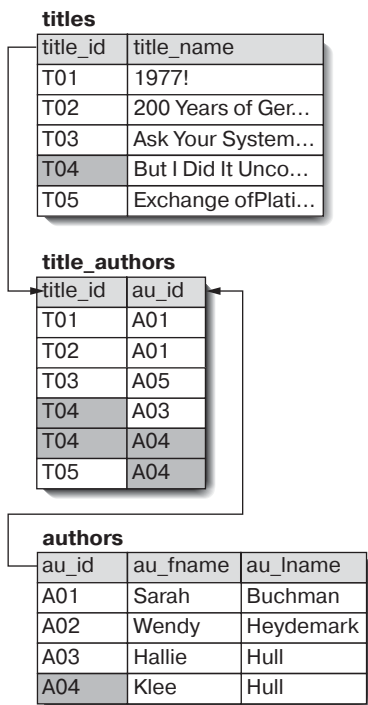


Рис. 2.14. Пример практической реализации связи «многие-ко-многим». Стыковочная таблица `title_authors` разбивает абстрактную связь «многие-ко-многим» между таблицами `titles` и `authors` на две реализуемых на практике связи «один-ко-многим». В результате такого преобразования каждая строка из таблицы `titles` может иметь много соответствующих строк в стыковочной таблице `title_authors` точно так же, как любая строка таблицы `authors` может иметь много соответствующих строк в таблице `title_authors`. Мы видим, что столбец `title_id` в таблице `title_authors` является внешним ключом, указывающим на первичный ключ таблицы `titles`. Но и столбец `au_id` в таблице `title_authors` тоже является внешним ключом, который указывает на первичный ключ таблицы `authors`.

П Предложения объединения (`join`), которые применяются для выполнения команд SQL на нескольких таблицах сразу, рассматриваются в главе 7.

П Если добавить повторяющиеся группы в соответствующие таблицы, можно построить связь «один-ко-многим» без третьей стыковочной таблицы, но такой подход был бы нарушением первой нормальной формы (см. следующий раздел).

П Стыковочную таблицу иногда называют *ассоциативной таблицей*, или *таблицей пересечений*.

П Связь «один-ко-многим» еще называют связью *предок-потомок*, или *ведущий-ведомый*.



Рис. 2.15. Еще один способ изобразить связь «многие-ко-многим», показанную на рис. 2.14

Нормализация

Возможно, у вас возник вопрос о том, зачем все эти сложности со связями, если можно занести всю информацию о книгах (или о любом другом объектном типе) в одну таблицу. Однако такая единая таблица содержала бы огромное количество дублируемых данных: каждая строка, соответствующая одной книге, включала бы избыточные данные об авторе, издателе и авторском гонораре. А избыточность – лютий враг администраторов баз данных. Дело в том, что именно избыточность является причиной неограниченного и быстрого роста баз данных. В слишком больших базах данных запросы исполняются очень-очень медленно, и обслуживание громоздких баз превращается в кошмар (когда какой-нибудь автор переедет на новое место жительства, вы, надо полагать, захотите внести изменения в одной строке, а не в тысяче строк).

Нормализация (Normalization) – это процесс последовательной пошаговой модификации таблиц произвольной базы данных, имеющий целью сокращение избыточности и несогласованности. Процесс нормализации устроен таким образом, что по завершении его очередного шага нормализуемая база данных переходит в определенную форму, называемую нормальной и связанную с этим шагом. Так, реляционная модель определяет три нормальных формы, каждая из которых названа в соответствии с порядковым числительным, показывающим номер шага, по завершении которого эта форма была достигнута:

- первая нормальная форма (1НФ);
- вторая нормальная форма (2НФ);
- третья нормальная форма (3НФ).

При этом каждая последующая нормальная форма сильнее предыдущей в том смысле, что удовлетворяет критериям предыдущей нормальной формы. Так, любая база данных, находящаяся в форме 3НФ, находится и в форме 2НФ (и, значит, в форме 1НФ тоже). При переходе любой базы данных на очередной уровень нормализации число таблиц в ней не уменьшается, поскольку нормализация на каждом шаге осуществляется с помощью декомпозиции (дробления) таблиц. Применяемая декомпозиция таблиц обладает двумя свойствами:

- *декомпозиция без потерь* означает, что никакое разбиение таблиц, осуществляемое в ходе нормализации, не приведет к потере информации;
- *декомпозиция с сохранением зависимости* означает, что никакое разбиение таблиц, осуществляемое в ходе нормализации, не приведет к потере ни одной связи.

Следует отметить, что не считаются избыточными вновь возникающие в процессе нормализации базы, а следовательно, разбиения таблиц, первичные и внешние ключи, являющиеся копией первичных ключей в исходных таблицах.

Могут существовать и более высокие уровни нормализации (например, 4НФ и 5НФ), но рассматриваемая в этой главе реляционная модель эти уровни не включает. Кроме того, типы избыточности, которые устраняются формами 4НФ и 5НФ, встречаются настолько редко, что, если вы приведете какую-нибудь практическую базу данных к форме 3НФ, она с большой вероятностью будет удовлетворять и критериям формы 5НФ, а значит, и критериям формы 4НФ. Принимая во внимание все сказанное о 4НФ и 5НФ, мы не будем

рассматривать уровни нормализации в настоящей книге.

Необходимо понимать, что процесс нормализации, несмотря на формальное понятие нормальной формы, является не системным (то есть раз и навсегда обоснованным алгоритмом достижения строго формализованной цели), а итеративным, включающим в себя разбиение и, возможно, повторное объединение таблиц. Результат каждого шага такого итеративного алгоритма оценивает не какой-то формальный критерий, а разработчик базы данных. Он же решает, когда прекратить нормализацию, удовлетворившись результатом (скорее всего, временным), достигнутым на текущем шаге.

Первая нормальная форма

Считается, что произвольная таблица находится в *первой нормальной форме*, если одновременно выполняются следующие два условия:

- любое значение любого столбца этой таблицы является элементарным;
- таблица не включает в себя повторяющихся групп.

Для понимания этого определения необходимо дать еще два:

- произвольное значение любой таблицы называется *элементарным* (иногда эти значения называют скалярами), если оно представляет собой единое целое, которое нельзя разделить на части без потери смысла (см. рис. 2.16);
- любое множество из двух и более логически связанных столбцов одной таблицы называют *повторяющейся группой* (см. рис. 2.17).

Чтобы решить проблемы, отмеченные на рис. 2.16 и 2.17, следует распределить данные

одной таблицы по двум таблицам, как показано на рис. 2.18.

Вторая нормальная форма

Прежде чем привести критерии, определяющие понятие второй нормальной формы, напомним одно полезное правило, которое поможет вам понять, что некая таблица, которую вы привели к 1НФ, автоматически является 2НФ. Итак, любая таблица 1НФ является таблицей 2НФ, если выполнено хотя бы одно из следующих двух условий:

- первичный ключ этой таблицы не является составным (то есть состоит из одного столбца);
- любой столбец этой таблицы входит в состав ее простого или составного первичного ключа.

Приведем точное определение второй нормальной формы. Считается, что произвольная таблица находится во *второй нормальной форме*, если одновременно выполнены два следующих условия:

- она находится в первой нормальной форме;
- она не имеет частичных функциональных зависимостей.

Для понимания этого определения надо привести еще одно: произвольная таблица содержит по крайней мере одну *частичную функциональную зависимость*, если для этой таблицы одновременно выполнены два следующих условия:

- первичный ключ этой таблицы – составной;
- несколько (но не все) значений составного первичного ключа определяют значение какого-нибудь неключевого столбца.

title_id	title_name	authors
T01	1977!	A01
T04	But I Did It Unconsciously	A03, A04
T11	Perhaps It's aGlandular Problem	A03, A04, A06

Рис. 2.16. Данная таблица нарушает критерии первой нормальной формы потому, что столбец `authors` содержит многомерные значения, указывающие сразу на нескольких разных авторов одной и той же книги (то есть соавторов). Эти значения можно разделить без потери смысла, что и составляет противоречие с определением 1НФ

title_id	title_name	author1	author2	author3
T01	1977!	A01		
T04	But I Did It Unconsciously	A03	A04	
T11	Perhaps It's aGlandular Problem	A03	A04	A06

Рис. 2.17. Здесь мы пытаемся решить проблему, обозначенную на рис. 2.16. Предлагается просто согласиться с тем, что у всех книг по три автора, и добавить в таблицу еще два столбца, а если авторов окажется два или один – не ставить в соответствующих столбцах вообще ничего (но «ничего» не бывает; это будет `null`). В этом состоит логическая связь трех столбцов друг с другом. Значит, они являются повторяющейся группой, нарушая тем самым критерии 1НФ. Общее правило гласит: единый многомерный объект не следует представлять в одной таблице в виде нескольких столбцов

title_id	title_name
T01	1977!
T04	But I Did It Unco...
T11	Perhaps It's a Gla...

title_id	au_id
T01	A01
T04	A03
T04	A04
T11	A03
T11	A04
T11	A06

Рис. 2.18. Правильное решение заключается в том, чтобы оформить информацию об авторах в отдельную дочернюю таблицу, в которой для любой книги (то есть для любого названия книги), для каждого автора этой книги предоставлена только одна строка. В этом случае первичным ключом материнской таблицы будет `title_id`, а составной первичный ключ дочерней таблицы будет включать столбцы `title_id` и `au_id`

Смысл всех приведенных определений заключается в том, что любая таблица 2НФ является *полностью функционально зависимой*. А полная функциональная зависимость (которая определяется как отсутствие какой бы то ни было частичной функциональной зависимости) означает, что, если *любое* значение *любого* ключевого столбца изменилось, потребуется вносить изменения в значения неключевых столбцов.

Составной первичный ключ той таблицы, которая представлена на рис. 2.19, включает в себя столбцы `title_id` и `au_id`. А неключевые столбцы – это столбец `au_order`, который содержит информацию о том, в каком порядке перечислены авторы на обложке книги, и столбец `au_phone` с указанием телефонных номеров авторов. Чтобы для произвольной таблицы определить, является она полностью функционально зависимой или нет, спросите себя: «Могу ли я определить значение неключевого столбца, если знаю только *часть* значений первичного ключа?» Если ответом будет «Нет», значит, ваша таблица является полностью функционально зависимой (что хорошо). Если ответом будет «Да», следовательно, ваша таблица является частично функционально зависимой (что плохо).

Применим эту теорию на практике. Например, займемся таблицей `title_authors`, представленной на рис. 2.19. Для неключевого столбца `au_order` задайте себе два вопроса:

- могу ли я определить `au_order`, если известно только поле `title_id`? Ответ: «Нет, потому что у одной книги может быть несколько авторов, и, не зная всех авторов, нельзя найти нужную запись»;

- могу ли я определить `au_order`, если знаю только `au_id`? Ответ: «Нет, потому что один автор может написать несколько книг, и, не зная о какой книге идет речь, нельзя определить нужную строку».

Итак, пока все хорошо, столбец `au_order` является полностью функционально зависимым и может оставаться в таблице. Эту зависимость принято записывать так: $\{title_id, au_id\} \rightarrow \{au_order\}$ и читать: «`title_id` и `au_id` определяют `au_order`» или «`au_order` зависит от `title_id` и `au_id`». При этом выражение, стоящее слева от стрелки, называется *детерминантом*.

Теперь для неключевого столбца `au_phone` вам надо задать себе два вопроса:

- могу ли я определить `au_phone`, если знаю только `title_id`? Ответ: «Нет, потому что у одной книги может быть более одного автора»;
- могу ли я определить `au_phone`, если знаю только `au_id`? Ответ: «Да! Номер телефона автора от книги не зависит».

Таким образом, мы получили плохой результат. Столбец `au_phone` является частично функционально зависимым, и для того, чтобы таблица `title_authors` стала таблицей 2НФ, его следует переместить в другую таблицу, возможно, в таблицу `authors` или `phone_numbers`.

Третья нормальная форма

Считается, что произвольная таблица находится в третьей нормальной форме, если одновременно выполнены два следующих условия:

- она находится во второй нормальной форме;
- она не содержит транзитивных зависимостей.

title_authors	
title_id	🔑
au_id	🔑
au_order	
au_phone	

Рис. 2.19. Столбец `au_phone` зависит от `au_id`, а не от `title_id`. Следовательно, данная таблица содержит частичную функциональную зависимость и не является 2НФ

titles	
title_id	🔑
price	
pub_city	
pub_id	

Рис. 2.20. Значение в столбце `pub_city` определяется значением в столбце `pub_id`, что говорит о частичной зависимости.

Таблица `titles` не находится в третьей нормальной форме

Для понимания этого определения необходимо дать еще одно. Итак, некая таблица содержит по крайней мере одну *транзитивную зависимость*, если в этой таблице есть пара неключевых столбцов, в которых значение одного неключевого столбца определяет значение другого неключевого столбца.

Из определения третьей нормальной формы следует, что в произвольной таблице, находящейся в 3НФ, столбцы являются взаимно независимыми, то есть зависят только от столбцов первичного ключа, будь он простым или составным. Ясно, что 3НФ – это следующий логический шаг процесса нормализации после 2НФ.

Снова проверим теорию на практике. Первичный ключ таблицы, показанной на рис. 2.20, – это столбец `title_id`, а неключевые столбцы – это `price`, где хранится информация о цене книги, `pub_city`, где хранится информация о городе, в котором издана книга, и `pub_id`, где хранится информация об издателе книги.

Проверяя произвольную таблицу на 3НФ, вам следует спросить себя: «Могу я выяснить значение неключевого столбца, если буду знать только значение какого-нибудь другого неключевого столбца?» Ответ «Нет» будет означать, что проверяемый вами неключевой столбец не является транзитивно зависимым, и это хорошо. Если же ответ будет «Да», значит, проверяемый столбец является транзитивно зависимым от другого неключевого столбца, и это плохо. Применим это правило к таблице `titles` на рис. 2.20. Для неключевого столбца `price` в соответствии с этим правилом вам следует задать себе следующие вопросы:

- могу ли я определить `pub_id`, если знаю `price`? Ответ: «Нет»;

- могу ли я определить `pub_city`, если знаю `price`? Ответ: «Нет».

Для неключевого столбца `pub_city` вопросы будут следующими:

- могу ли я определить `price`, если знаю `pub_city`? Ответ: «Нет»;
- могу ли я определить `pub_id`, если знаю `pub_city`? Ответ: «Нет, потому что в одном городе может находиться много издателей».

Для неключевого столбца `pub_id` вопросы выглядят так:

- могу ли я определить `price`, если знаю `pub_id`? Ответ: «Нет»;
- могу ли я определить `pub_city`, если знаю `pub_id`? Ответ: «Да! Поскольку место выпуска книги зависит от того, кто является ее издателем».

Плохая новость: столбец `pub_city` является транзитивно зависимым от неключевого столбца `pub_id`, и для того, чтобы таблица `titles` была таблицей ЗНФ, его надо переместить в другую таблицу, возможно, в таблицу `publishers`.

Задавая вопросы наподобие тех, что приведены выше, недостаточно спросить: «Могу ли я определить А, если знаю В?» Чтобы осуществить полную проверку на транзитивную зависимость, надо еще спросить: «Могу ли я определить В, если знаю А?»

С

Если у вас нет опыта работы с базами данных, жесткость нормальных форм может показаться вам излишней и породить недоумение типа: «Почему я не могу, к примеру, поместить штат, город и почтовый индекс в один столбец, ведь это бы упростило процесс печати наклеек на почтовые конверты?» Ответ состоит в том, что вы не можете ручаться за то, как вы сами или другие пользователи захотят в будущем выбирать рассматриваемую информацию и манипулировать ею. Вот почему ЗНФ является наилучшей страховкой от такой неопределенности в будущем. Независимо от вашего отношения к идее нормализации и вышеизложенной теории, мы настоятельно рекомендуем вам нормализовать свои базы данных. Дело в том, что если этого не выполнить, то рано или поздно вам придется разбивать свои таблицы, чтобы сделать возможной сортировку, скажем, адресов по штату. Заметим, что профессионалы иногда денормализуют свои базы с целью ускорить выполнение запросов. Однако такие технические приемы в настоящем издании не рассматриваются.

Наша типовая база данных

Возьмите любую книгу по SQL или по разработке и проектированию баз данных, и, скорее всего, вы найдете в ней какую-нибудь базу данных, включающую информацию о, скажем, учебных курсах/студентах/преподавателях, или о клиентах/заказах/товарах, или об авторах/книгах/издателях. Поэтому мы не стали придумывать ничего нового, а выбрали в качестве типовой базы данных привычную базу авторов/книги/издатели. Нашу типовую базу мы назовем *books*. Перечислим для ознакомления несколько важных моментов:

- в разделе «Таблицы, столбцы и строки» уже говорилось, что для пользователя любая база данных представляется как набор таблиц, и ничего кроме таблиц. Так вот, *books* содержит пять таблиц, которые включают информацию об авторах книг, названиях книг, которые эти авторы опубликовали, об издателях, выпустивших эти книги, и о гонорарах, которые были получены авторами от издателей за опубликованные книги. На рис. 2.21 перечислены все названные таблицы и связи между ними с учетом графических условностей, которые мы ввели в этой главе;
- команды SQL, рассматриваемые в главе 9, будут модифицировать данные в базе *books*, а не просто извлекать их. В этой связи имейте в виду, что, если никаких специальных оговорок нет, каждый раздел любой главы будет начинаться с самой первой версии *books*, не смотря на все модификации. То есть вам имеет смысл считать, что все изменения нашей типовой базы, сделанные в любой главе, в следующую главу не переносятся;
- чтобы создать (или восстановить) базу данных *books* в своей СУБД, достаточно выполнить SQL-скрипт из приложения. Готовый скрипт можно «скачать» (как, впрочем, и готовую базу данных) с сайта издательства «ДМК Пресс» www.dmkpress.ru или с сайта поддержки книги (см. раздел «Несколько общих замечаний о книге» во введении);
- некоторые из упомянутых в данном разделе идей, в частности об использовании типов данных и значения *null*, рассматриваются в следующей главе;
- наша типовая база *books* – учебная. Ее сложность даже в первом приближении не дает представления о сложности реально работающих баз данных.

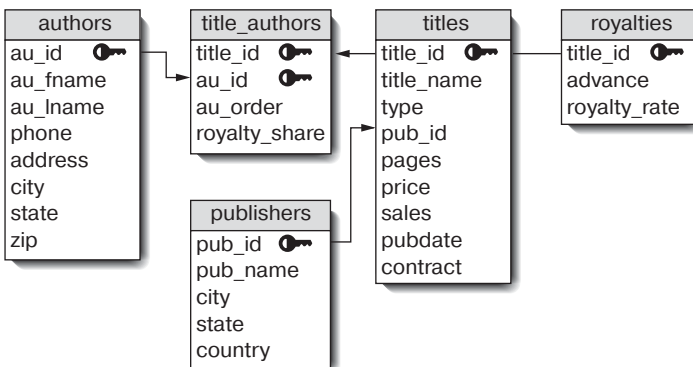


Рис. 2.21. Типовая база данных *books*

Таблица authors

Таблица authors описывает авторов книг, включенных в базу. Здесь каждый автор имеет уникальный идентификатор, являющийся первичным ключом. Табл. 2.22 отражает структуру таблицы authors, а рис. 2.22 раскрывает ее содержание.

Таблица 2.2. Структура таблицы authors

Имя столбца	Описание	Тип данных	null (Да/Нет)	Ключи
au_id	Уникальный идентификатор автора	CHAR(3)		PK
au_fname	Имя автора	VARCHAR(15)		
au_lname	Фамилия автора	VARCHAR(15)		
phone	Телефон автора	VARCHAR(12)	Да	
address	Адрес автора	VARCHAR(20)	Да	
city	Город автора	VARCHAR(15)	Да	
state	Штат автора	CHAR(2)	Да	
zip	Почтовый индекс автора	CHAR(5)	Да	

au_id	au_fname	au_lname	phone	address	city	state	zip
-----	-----	-----	-----	-----	-----	-----	-----
A01	Sarah	Buchman	718-496-7223	75 west 205 St	Bronx	NY	10468
A02	Wendy	Heydemark	303-986-7020	2922 Baseline Rd	Boulder	CO	80303
A03	Hallie	Hull	415-549-4278	3800 Waldo Ave, #14F	San Francisco	CA	94123
A04	Klee	Hull	415-549-4278	3800 Waldo Ave, #14F	San Francisco	CA	94123
A05	Christian	Kells	212-771-4680	114 Horatino St	New York	NY	10014
A06		Kellsey	650-836-7128	390 Serra Mall	Palo Alto	CA	94305
A07	Paddy	O'Furniture	941-925-0752	1442 Main St	Sarasota	FL	34236

Рис. 2.22. Содержимое таблицы authors

Таблица publishers

Таблица publishers описывает издателей книг. Здесь каждый издатель имеет некий уникальный идентификатор, являющийся первичным ключом. Табл. 2.3 отражает структуру таблицы publishers, а рис. 2.23 раскрывает ее содержание.

Таблица 2.3. Структура таблицы publishers

Имя столбца	Описание	Тип данных	null (Да/Нет)	Ключи
pub_id	Уникальный идентификатор издателя	CHAR(3)		PK
pub_name	Наименование издателя	VARCHAR(20)		
city	Город издателя	VARCHAR(15)		
state	Штат/Графство издателя	CHAR(2)	Да	
country	Страна издателя	VARCHAR(15)		

pub_id	pub_name	city	state	country
-----	-----	-----	-----	-----
P01	Abatis Publishers	New York	NY	USA
P02	Core Dump Books	San Francisco	CA	USA
P03	Schadenfreude Press	Hamburg	NULL	Germany
P04	Tenterhooks Press	Berkeley	CA	USA

Рис. 2.23. Содержимое таблицы publishers

Таблица titles

Таблица titles описывает собственно книги. У каждой книги в этой таблице есть уникальный идентификатор, который и является первичным ключом. Таблица titles содержит внешний ключ pub_id, указывающий на таблицу publishers для того, чтобы обозначить издателя книги.

Структура таблицы titles приведена в табл. 2.4, а ее содержимое показано на рис. 2.24.

Таблица 2.4. Структура таблицы titles

Имя столбца	Описание	Тип данных	null (Да/Нет)	Ключи
title_id	Уникальный идентификатор книги	CHAR(3)		PK
title_name	Название книги	VARCHAR(40)		
type	Предмет книги	VARCHAR(10)	Да	FK publishers (pub_id)
pub_id	Идентификатор издателя	CHAR(3)		
pages	Количество страниц	INTEGER	Да	
price	Цена за 1 экземпляр	DECIMAL(5,2)	Да	
sales	Общее количество проданных экземпляров	INTEGER	Да	
pubdate	Дата публикации	DATE	Да	
contract	Не совпадает с null, если автор/авторы подписал/подписали контракт	SMALLINT		

title_id	title_name	type	pub_id	pages	price	sales	pubdate	contract
T01	1977!	history	P01	107	21,99	566	2000-08-01	1
T02	200 Years of German Humor	history	P03	14	19,95	9566	1998-04-01	1
T03	Ask Your System Administrator	computer	P02	1226	39,95	25667	2000-09-01	1
T04	But I Did It Unconsciously	psychology	P01	510	12,99	13001	1999-05-31	1
T05	Exchange of Platitudes	psychology	P01	201	6,95	201440	2001-01-01	1
T06	How About Never?	biography	P01	473	19,95	11320	2000-07-31	1
T07	I Blame My Mother	biography	P03	333	23,95	1500200	1999-10-01	1
T08	Just Wait Until After School	children	P01	86	10	4095	2001-06-01	1
T09	Kiss My Boo-Boo	children	P01	22	13,95	5000	2002-05-31	1
T10	Not Without My Faberge Egg	biography	P01	NULL	NULL	NULL	NULL	0
T11	Perhaps It's a Glandular Problem	psychology	P01	826	7,99	94123	2000-11-30	1
T12	Spontaneous, Not Annoying	biography	P01	507	12,99	100001	2000-08-31	1
T13	What Are The Civilian Applications?	history	P03	802	29,99	10467	1999-05-31	1

Рис. 2.24. Содержимое таблицы titles

title_id	au_id	au_order	royalty_share
-----	-----	-----	-----
T01	A01	1	1.00
T02	A01	1	1.00
T03	A05	1	1.00
T04	A03	1	0.60
T04	A04	2	0.40
T05	A04	1	1.00
T06	A02	1	1.00
T07	A02	1	0.50
T07	A04	2	0.50
T08	A06	1	1.00
T09	A06	1	1.00
T10	A02	1	1.00
T11	A03	2	0.30
T11	A04	3	0.30
T11	A06	1	0.40
T12	A02	1	1.00
T13	A01	1	1.00

Рис. 2.25. Содержимое таблицы title_authors

Таблица title_authors

Объектные типы «Авторы», которым соответствует таблица authors, и «Книги», которым соответствует таблица titles, имеют связь «многие-ко-многим», потому что, с одной стороны, один автор способен написать много книг, а с другой – одна книга может быть написана несколькими авторами. Чтобы реализовать эту связь, необходима, как мы теперь знаем, стыковочная таблица. Таблица title_authors является стыковочной таблицей, соединяющей таблицы titles и authors (см. раздел «Связи» в этой главе). В стыковочной таблице столбцы title_id и au_id вместе формируют составной первичный ключ, а по отдельности каждый из них является внешним ключом, указывающим соответственно на таблицы titles и authors. Неключевые столбцы содержат столбец au_order, показывающий очередность имени автора на обложке книги (для книг с единственным автором значение этого столбца всегда равно 1), и столбец royalty_share, показывающий долю автора в общем гонораре за книгу (для книг с единственным автором значение этого столбца всегда равно 1). Табл. 2.5 отражает структуру таблицы title_authors, а рис. 2.25 показывает ее содержание.

Таблица 2.5. Структура таблицы title_authors

Имя столбца	Описание	Тип данных	null (Да/Нет)	Ключи
title_id	Уникальный идентификатор книги	CHAR(3)		PK, FK titles (title_id)
au_id	Идентификатор автора	CHAR(3)		PK, FK authors (au_id)
au_order	Очередность следования имени автора на обложке	SMALLINT		
royalty_share	Доля автора в общем авторском гонораре	DECIMAL(5,2)		

Таблица royalties

В таблице `royalties` хранятся данные о проценте отчислений от продажи книги в виде авторского гонорара, выплачиваемого всем авторам (отметим, не каждому автору), включая авансовый платеж, выплачиваемый всем авторам (подчеркнем, не каждому автору) произвольной книги. Первичным ключом таблицы `royalties` является столбец `title_id`. Таблица `royalties` имеет связь «один-к-одному» с таблицей `titles`, поэтому первичный ключ таблицы `royalties` является одновременно и ее внешним ключом, который указывает на первичный ключ таблицы `titles`.

Табл. 2.6 отображает структуру таблицы `royalties`, а рис. 2.26 – ее содержание.

title_id	Advance	royalty_rate
-----	-----	-----
T01	10000	0,05
T02	1000	0,06
T03	15000	0,07
T04	20000	0,08
T05	100000	0,09
T06	20000	0,08
T07	1000000	0,11
T08	0	0,04
T09	0	0,05
T10	NULL	NULL
T11	100000	0,07
T12	50000	0,09
T13	20000	0,06

Рис. 2.26. Содержимое таблицы royalties

Таблица 2.6. Структура таблицы royalties

Имя столбца	Описание	Тип данных	null (Да/Нет)	Ключи
title_id	Уникальный идентификатор книги	CHAR(3)		PK,FK titles (title_id)
advance	Предоплата автору (авторам)	DECIMAL(9,2)	Да	
royalty_rate	Доля общего дохода с продаж книги, выплачиваемая в виде авторского гонорара	DECIMAL(5,2)		

ОСНОВЫ SQL

3

Вы, наверное, заметили, что в предыдущей главе мы почти не упоминали об SQL. И этому есть причина, которую символически можно объяснить так:

SQL \neq Реляционная модель.

Дело в том, что язык SQL *основан* на реляционной модели, но не является прямой попыткой ее внедрения. Например, одно из отклонений от канонической реляционной модели состоит в том, что первичные ключи в SQL, в отличие от модели, не являются обязательными. Из этого следует, что с точки зрения SQL таблицы, у которых нет ключей, могут иметь строки, являющиеся точной копией друг друга. Значит, какие-то данные таких таблиц могут стать недоступными. Но подробно разбираться с несоответствиями между SQL и реляционной моделью мы не будем, а порекомендуем ознакомиться со статьями И. Ф. Кодда (E. F. Codd), Криса Дейта (Chris Date) или Фабьена Паскаля (Fabian Pascal), опубликованными в Internet. Однако подчеркнем: результатом этих несоответствий явилось то, что именно пользователи СУБД, а не сама СУБД должны строить реляционные модели. Еще одним результатом является то, что термины модели

и SQL, как это видно из табл. 2.1, не полностью взаимозаменяемы.

Теперь, когда с оговорками покончено, приступим к изучению SQL. Итак, любая программа SQL – это некая последовательность команд SQL, выполняемых в определенном порядке. Чтобы написать такую программу, следует выучить правила, управляющие синтаксисом SQL. В этой главе мы познакомимся с правилами, которые управляют формированием произвольной команды SQL, типами данных и значениями null.

Синтаксис SQL

На рис. 3.1 показан пример команды SQL «в разрезе». Постарайтесь немного отвлечься от семантики этой команды. Дело в том, что мы использовали ее только для того, чтобы объяснить синтаксис SQL. Начнем с определений.

Комментарий. Всякий *комментарий* – это необязательный текст, который вы печатаете на отдельной строке программы, чтобы объяснить эту программу. Комментарий должен начинаться с двух дефисов. Когда СУБД обнаруживает их, она игнорирует то, что стоит за ними, то есть

сам комментарий. Комментарии занимают целую строку.

Команда SQL. Любая команда SQL – это допустимая комбинация лексем, которую предваряет какое-нибудь ключевое слово. Здесь под лексемой понимается основная неделимая частица языка SQL в том значении, что грамматически никакую лексему нельзя разделить на более мелкие составные элементы. Лексемами являются ключевые слова, идентификаторы, операторы, литералы и другие символы (все они будут рассмотрены позднее).

Предложения. Любая команда SQL включает не менее одного предложения. В самом общем случае всякое предложение SQL – это фрагмент команды SQL, который начинается с какого-нибудь ключевого слова, является обязательным или необязательным и должен быть записан в определенном порядке. В рассматриваемом примере мы имеем четыре предложения, а именно: SELECT, FROM, WHERE, ORDER.

Ключевые слова. Произвольное ключевое слово, иногда называемое зарезервированным, – это такое слово, которое в языке SQL имеет определенное значение и при-

менение которого в SQL строго регламентировано. Следует иметь в виду, что использование любого ключевого слова вне контекста (например, в качестве идентификатора) будет считаться ошибкой. В табл. 3.1 перечислены ключевые слова SQL, а в табл. 3.2 – потенциальные слова SQL (которые пока не являются официально зарезервированными, но однажды могут ими стать).

Идентификаторы. Произвольный идентификатор – это такое слово, которое вы (или разработчик базы данных) применяете для того, чтобы именовать объекты произвольной базы данных, в том числе таблицы, столбцы, псевдоимена (псевдонимы) и представления. Идентификатор не может быть ключевым словом, и его длина не может превышать 128 знаков. Знаком в SQL может быть любой символ алфавита, включая символы латинского алфавита и латинские идеограммы (однако обратите внимание на советы, приведенные в конце настоящего раздела). В нашем примере именами, в частности, являются au_fname, au_lname, authors и state.

Завершающая точка с запятой. Запись каждой команды SQL должна заканчиваться точкой с запятой¹.

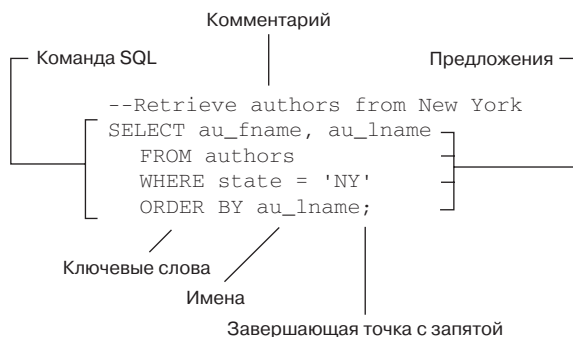


Рис. 3.1. Пример команды SQL с комментарием

Таблица 3.1. Ключевые слова SQL

ABSOLUTE	COMMIT	ELSE	INSERT	Null
ACTION	CONNECT	END	INT	ONLY
ADD	CONNECTION	END-EXEC	INTEGER	OPEN
ALL	CONSTRAINT	ESCAPE	INTERSECT	OPTION
ALLOCATE	CONSTRAINTS	EXCEPT	INTERVAL	OR
ALTER	CONTINUE	EXCEPTION	INTO	ORDER
AND	CONVERT	EXEC	IS	OUTER
ANY	CORRESPONDING	EXECUTE	ISOLATION	OUTPUT
ARE	COUNT	EXISTS	JOIN	OVERLAPS
AS	CREATE	EXTERNAL	KEY	PAD
ASC	CROSS	EXTRACT	LANGUAGE	PARTIAL
ASSERTION	CURRENT	FALSE	LAST	POSITION
AT	CURRENT_DATE	FETCH	LEADING	PRECISION
AUTHORIZATION	CURRENT_TIME	FIRST	LEFT	PREPARE
AVG	CURRENT_TIMESTAMP	FLOAT	LEVEL	PRESERVE
BEGIN	CURRENT_USER	FOR	LIKE	PRIMARY
BETWEEN	CURSOR	FOREIGN	LOCAL	PRIOR
BIT	DATE	FOUND	LOWER	PRIVILEGES
BIT_LENGTH	DAY	FROM	MATCH	PROCEDURE
BOTH	DEALLOCATE	FULL	MAX	PUBLIC
BY	DEC	GET	MIN	READ
CASCADE	DECIMAL	GLOBAL	MINUTE	REAL
CASCADED	DECLARE	GO	MODULE	REFERENCES
CASE	DEFAULT	GOTO	MONTH	RELATIVE
CAST	DEFERRABLE	GRANT	NAMES	RESTRICT
CATALOG	DEFERRED	GROUP	NATIONAL	REVOKE
CHAR	DELETE	HAVING	NATURAL	RIGHT
CHARACTER	DESC	HOURL	NCHAR	ROLLBACK
CHAR_LENGTH	DESCRIBE	IDENTITY	NEXT	ROWS
CHARACTER_LENGTH	DESCRIPTOR	IMMEDIATE	NO	SCHEMA
CHECK	DIAGNOSTICS	IN	NOT	SCROLL
CLOSE	DISCONNECT	INDICATOR	NULL	SECOND
COALESCE	DISTINCT	INITIALLY	NULLIF	SECTION
COLLATE	DOMAIN	INNER	NUMERIC	SELECT
COLLATION	DOUBLE	INPUT	OCTET_LENGTH	SESSION
COLUMN	DROP	INSENSITIVE	OF	SESSION_USER
SET	SUM	TRAILING	UPPER	WHENEVER
SIZE	SYSTEM_USER	TRANSACTION	USAGE	WHERE
SMALLINT	TABLE	TRANSLATE	USER	WITH

Таблица 3.1. Ключевые слова SQL (окончание)

SOME	TEMPORARY	TRANSLATION	USING	WORK
SPACE	THEN	TRIM	VALUE	WRITE
SQL	TIME	TRUE	VALUES	YEAR
SQLCODE	TIMESTAMP	UNION	VARCHAR	ZONE
SQLERROR	TIMEZONE_HOUR	UNIQUE	VARYING	
SQLSTATE	TIMEZONE_MINUTE	UNKNOWN	VIEW	
SUBSTRING	TO	UPDATE	WHEN	

Таблица 3.2. Потенциальные ключевые слова SQL

AFTER	EQUALS	OLD	RETURN	TEST
ALIAS	GENERAL	OPERATION	RETURNS	THERE
ASYNС	IF	OPERATORS	ROLE	TRIGGER
BEFORE	IGNORE	OTHERS	ROUTINE	TYPE
BOOLEAN	LEAVE	PARAMETERS	ROW	UNDER
BREADTH	LESS	PENDANT	SAVEPOINT	VARIABLE
COMPLETION	LIMIT	PREORDER	SEARCH	VIRTUAL
CALL	LOOP	PRIVATE	SENSITIVE	VISIBLE
CYCLE	MODIFY	PROTECTED	SEQUENCE	WAIT
DATA	NEW	RECURSIVE	SIGNAL	WHILE
DEPTH	NONE	REF	SIMILAR	WITHOUT
DICTIONARY	OBJECT	REFERENCING	SQLEXCEPTION	
EACH	OFF	REPLACE	SQLWARNING	
ELSEIF	OID	RESIGNAL	STRUCTURE	

SQL – это язык со свободным форматом предложений. Любая его команда может:

- быть напечатана как в верхнем, так и в нижнем регистре (например, ключевые слова `SELECT` и `select` считаются идентичными);
- продолжаться на следующей строке сколь угодно долго при условии, что вы не будете разбивать на две части слова, лексемы и строки в кавычках (то есть строковые константы);
- быть напечатана на одной строке с любыми другими командами;
- начинаться на любой позиции горизонтальной разметки экрана/листа.

Однако вам следует придерживаться определенного стиля написания команд SQL (см. рис. 3.2), например использовать верхний регистр для ключевых слов и нижний регистр для идентификаторов. Кроме того, можно начинать каждое предложение с новой строки с соответствующим отступом (см. подраздел «Дополнительные соглашения» в этой главе). Вопрос стиля нельзя игнорировать, потому что его отсутствие приводит к следующим ошибкам:

- неправильная орфография при написании какого-нибудь идентификатора или команды;
- отсутствие завершающей точки с запятой;
- неправильный порядок расположения предложений в теле команды;

- отсутствие кавычек у строковых констант (литералов) и констант даты и времени;
- наличие кавычек у цифровых констант;
- неправильное совместное применение имен таблиц и имен столбцов (например, для нашей типовой учебной базы такой ошибкой было бы использование `SELECT royalty_share FROM authors;` вместо `SELECT royalty_share FROM title_authors;`).

П

В литературе по SQL вводное ключевое слово любой команды часто называют *глаголом*, потому что оно явно указывает на то действие, которое необходимо совершить.

П

Поскольку употреблять ключевые слова в качестве идентификаторов запрещено, вы вполне можете встраивать любые ключевые слова внутрь идентификаторов как составную часть слова. Например: `group` и `max` нельзя использовать как идентификаторы (они являются зарезервированными словами – см. табл. 3.1), а идентификаторы `groups` и `max_price` вполне допустимы.

```
select au_fname
      , AU_LNAME
      FROM
authors WhErE      state
= 'NY' order
                bY
AU_lname
;
```

Рис. 3.2. Правил форматирования команд SQL немного. Команда, приведенная на этом рисунке, эквивалентна представленной на рис. 3.1

¹ Некоторые современные СУБД не требуют ставить точку с запятой в конце команды, хотя и не запрещают это (например, MS SQL Server). – Прим. науч. ред.

П

Применять пробелы в именах базы данных не рекомендуется. Но если очень хочется, можно вставить пробелы в любой идентификатор, тогда его надо будет заключить в одинарные кавычки, вот так: 'last name'. Тем не менее подчеркнем еще раз: лучше применять не пробелы, а знак подчеркивания (last_name) или идентификатор, набранный заглавными и прописными буквами (LastName).

П

Выражение – это любая разрешенная комбинация символов, которая служит для вычисления единого агрегированного значения данных. Составляя любое выражение, вы можете комбинировать математические или логические операторы, идентификаторы, константы, функции, имена столбцов и т.д. В табл. 3.3 перечислены часто используемые выражения вместе с примерами. Всякий раз, когда то или иное распространенное выражение возникнет по ходу изложения материала, мы будем разбирать его достаточно подробно.



Коммерческие СУБД налагают собственные ограничения на длину идентификатора и на алфавит. Поэтому имеет смысл проанализировать документацию по вашей СУБД, используя ключи поиска «идентификаторы» и «имена».

Более того, различные СУБД имеют свои собственные дополнительные ключевые слова, которые, очевидно, не могут быть идентификаторами в «родной» СУБД (но вполне могут быть идентификаторами по канонам SQL и, возможно, в других СУБД). Вот почему неплохо было бы организовать поиск в документации по вашей СУБД с использованием ключей поиска «ключевые слова» и «зарезервированные слова». Имейте в виду, что Microsoft SQL Server, Oracle, MySQL и PostgreSQL разрешают встроенные сопроводительные и многострочные комментарии, которые необходимо помещать между символами /* и */. Организуйте поиск в документации по ключам «комментарии». Имейте в виду, что комментарии MySQL могут начинаться со знака #.

Таблица 3.3. Типы выражений

Выражение	Пример
Case (Выбор)	CASE WHEN n <> 0 THEN x/n ELSE 0 END
Cast (Преобразование типов)	CAST(pubdate AS CHARACTER)
Datetime (Дата или время)	start_time + '01:30'
Interval (Интервал между датами или между отметками времени суток)	INTERVAL '7' DAY*2
Numeric (Числовой)	(sales*price)/12
String (Строковый)	'Dear ' au_fname ', '

Таблица 3.4. Категории типов данных

Тип данных	Хранит данные
Character string	Строки символов
Bit string	Строки битов
Exact numeric	Целые и действительные числа с плавающей десятичной точкой
Approximate numeric	Числа с плавающей точкой
Datetime	Значения даты и времени
Interval	Интервалы даты и времени

Типы данных

В разделе «Таблицы, столбцы и строки» главы 2 уже говорилось о том, что домен любого столбца накладывает определенные ограничения на то, какие значения разрешено хранить в этом столбце. В языке SQL для определения домена служит понятие типа данных.

Итак, *тип данных* имеет следующие характеристики:

- любой столбец в какой угодно таблице произвольной базы данных может содержать данные какого-нибудь, но только одного типа;
- любой тип данных из тех, что применяются в любой таблице произвольной базы, должен относиться к одной из категорий, перечисленных в табл. 3.4;
- тип данных налагает ограничения на те значения, которые разрешается хранить в любом столбце, и определяет те операции, которые с этими значениями можно выполнять. Например, столбец, для которого определен тип данных `integer` (целые числа), может содержать в качестве своего значения любое целое число, удовлетворяющее ограничениям конкретной СУБД, и для этого значения СУБД будет поддерживать обычные арифметические операции сложения, вычитания, умножения и деления (а также другие операции из тех, что определены над полем целых чисел). Но в этот столбец нельзя записать нечисловое значение вроде `'shadenfreude'`, и никакая СУБД не будет для этого столбца поддерживать символьные операции наподобие преобразования символов в верхний регистр (капитализация);

- тип данных любого столбца произвольной базы определяет порядок сортировки этого столбца. Так, целые числа 1, 2 и 10, записанные в некий столбец с типом данных `integer`, будут отсортированы в арифметическом порядке: 1, 2, 10. Те же целые числа, но записанные в какой-нибудь другой столбец как строки символов '1', '2' и '10', будут отсортированы по лексикографическому принципу: '1', '10' и '2'. Имейте в виду: *лексикографическое упорядочивание* сначала исследует первые знаки упорядочиваемых значений, а потом разбивает эти значения на упорядоченные группы, соответствующие первым знакам. Если среди этих групп есть такие, которые состоят из более чем одного элемента, выполняется сортировка уже в этих группах, но по второму знаку. Так продолжается до тех пор, пока во всех группах окажется или по одному члену, или все члены будут равны. Таким образом, при лексикографическом упорядочивании '10' стоит перед '2', поскольку первый символ числа 10 идет раньше, чем первый символ числа 2;
- значения литералов хранятся в столбцах в соответствии с типами данных этих столбцов. При этом считается, что литерал – это буквальная константа (ее еще называют константой в явном представлении), выражающая строго себя, а не какой-либо результат произвольного выражения (каковым может быть, например, арифметическая формула). В качестве примеров литералов можно привести 40 и 12.34 – числовые, '40' и 'ennui' – символьные, DATE '2002-05-10' и TIME '09:45:00' – литералы даты и времени (datetime).

С

Чтобы определить или изменить тип данных любого столбца, применяйте команды `CREATE TABLE` и `ALTER TABLE`. Подробнее об этом см. в главе 10.

П

Опытные разработчики баз данных подходят к назначению типов столбцов с осторожностью. Дело в том, что, если тип столбца назначен неправильно, велика вероятность потери информации при смене типа данных столбца.



Поставщики коммерческих СУБД решают многие вопросы практического применения типизации данных. И можно утверждать, что типы данных канонической СУБД SQL не всегда совпадают с теми типами данных, которые практикует конкретная СУБД. Причем совпадение не гарантировано даже в том случае, когда тип данных языка SQL и тип данных некой СУБД называются одинаково. Имея в виду этот факт, в следующих разделах, посвященных типизации данных, мы будем давать советы, разъясняющие эквивалентность или похожесть типов данных языка SQL и конкретных СУБД. Необходимо еще отметить, что СУБД оперируют большим количеством типов данных, чем язык. Это сделано для того, чтобы хранить некоторые специальные значения, например булевы (Boolean) или денежные (monetary). Если у вас возникла потребность разобраться со стандартными типами данных и с теми типами данных, которые попадают в расширение стандартных типов данных, следует проанализировать документацию по вашей СУБД с использованием ключа поиска «типы данных».

Подробнее о порядке сортировки читайте в разделе «Сортировка строк с помощью предложения `ORDER BY`» главы 4.

Строковые типы данных

Строковые, или текстовые, типы данных (Character string) используются для представления текста. Любая строка символов, или просто строка, отличается следующими характеристиками:

- является упорядоченной последовательностью некоторого количества (возможно, нулевого) знаков;

- ее длина может быть как фиксированной, так и переменной;
- учитывает регистр (например, при сортировке 'A' стоит перед 'a');
- в командах SQL любой строковый литерал должен быть заключен в одинарные кавычки;
- должна относиться к одному из типов данных, перечисленных в табл. 3.5.

Таблица 3.5. Строковые типы данных

Тип	Описание
CHARACTER	Строка фиксированной длины. Представляет собой фиксированное число символов. Любая строка, хранимая в столбце, определенном как CHARACTER(длина), может состоять из любого числа символов, не превышающего того числа, которое мы обозначили здесь как длина и которое должно быть целым, не меньше 1. Максимально допустимая длина строки определяется конкретной СУБД. Когда вы записываете в столбец, определенный как CHARACTER(длина), какую-нибудь строку, фактическая длина которой меньше той, которая определена для данного столбца, СУБД автоматически дополняет фактическую длину до полной пробелами (например, строка 'Jack' записалась бы в столбец CHARACTER (6) как строка 'Jack '). Имейте в виду, что CHARACTER и CHAR – синонимы
CHARACTER VARYING	Строка переменной длины. Представляет собой любое число символов, не превышающее некоего фиксированного числа. Любая строка, хранимая в столбце, определенном как CHARACTER VARYING(длина), может состоять из любого числа символов, не превышающего того числа, которое мы обозначили здесь как длина и которое должно быть целым, не меньше 1. Максимально допустимая длина строки определяется конкретной СУБД. Когда вы записываете в столбец, определенный как CHARACTER VARYING(длина), какую-нибудь строку, фактическая длина которой меньше определенной для данного столбца, СУБД, в отличие от случая с CHARACTER, не дополняет автоматически фактическую длину до полной пробелами, а хранит ее в том виде, в каком она есть (например, строка 'Jack' записалась бы в столбец CHARACTER VARYING(6) как строка 'Jack'). Имейте в виду, что CHARACTER VARYING, CHAR VARYING и CHARVAR – синонимы
NATIONAL CHARACTER	Этот тип данных совпадает с типом CHARACTER, только в столбце этого типа можно хранить лишь стандартизованные многобайтовые или двухбайтовые знаки (Unicode) (см. врезку в данном разделе). В командах SQL строки в формате NATIONAL CHARACTER следует записывать так, как строки в формате CHARACTER, но перед открывающей кавычкой надо ставить знак N, например: N'a*b'. Имейте в виду, что NATIONAL CHARACTER, NATIONAL CHAR и NCHAR – синонимы
NATIONAL CHARACTER VARYING	Этот тип данных совпадает с типом CHARACTER VARYING, только в столбце этого типа можно хранить лишь стандартизованные многобайтовые или двухбайтовые знаки (Unicode) (см. врезку в данном разделе и пояснение по типу данных NATIONAL CHARACTER). Имейте в виду, что NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING и NCHAR VARYING – синонимы

С Чтобы в произвольную строку вставить одинарную кавычку, введите подряд два символа одинарной кавычки. Например, если вы напечатаете `it''s`, получится строка `it's`. Двойная кавычка является отдельным знаком и специального обращения, как одинарная кавычка, не требует.

П Фактическая длина произвольной строки измеряется количеством знаков в этой строке и равна целому числу в интервале от 0 до числа, подставленного вместо слова, которое мы обозначили как *длина* при определении типа столбца. Строка, которая вообще не содержит символов, то есть две одинарных кавычки (") без пробела между ними, называется пустой, или *строкой нулевой длины*. Такая строка относится к типу данных `VARCHAR` нулевой длины.

П Коммерческие СУБД обрабатывают строки фиксированной длины быстрее, чем строки переменной длины.



Строковыми типами данных в рассматриваемых коммерческих СУБД являются:

- Microsoft Access – `text`, `memo`;
- Microsoft SQL Server – `varchar`, `text`, `nchar`, `nvarchar`, `ntext`;
- Oracle – `char`, `varchar`, `varchar2`, `nchar`, `nvarchar`, `nvarchar2`;
- MySQL – `char`, `varchar`, `nchar`, `nvarchar`, `text`, `tinytext`, `mediumtext`, `longtext`;
- PostgreSQL – `char`, `varchar`, `text`.

Имейте в виду, что СУБД Oracle воспринимает любую пустую строку как значение `null` (см. раздел «Значение `null`» в этой главе).

Двухбайтовая система кодирования символов Unicode

Любой компьютер хранит знаки (буквы, цифры, знаки пунктуации, символы, в том числе управляющие) в виде числовых значений. Говоря формальным языком, любая *система кодирования* – это взаимнооднозначное отображение множества знаков на некое подмножество натуральных чисел. При этом существенно, что мировые языки и операционные системы имеют разные национальные системы кодирования. Например, стандартные американские строковые константы применяют систему кодирования ASCII, присваивают значения $256 = 2^8$ различным знакам. Это немного. Это даже мало, потому что такого количества знаков едва хватает на весь латинский алфавит.

Так вот, Unicode – это единое множество чисел (16-разрядных двоичных или двухбайтовых), которое представляет знаки почти всех мировых языков. Unicode может закодировать $65\,536 = 2^{16}$ знаков. Консорциум Unicode разработал одноименный стандарт и следит за его применением. Действующие системы кодирования на основе Unicode представлены как в бумажном варианте, так и в Internet. Подробную информацию об этом стандарте можно найти на сайте www.unicode.org.

Таблица 3.6. Битовые типы данных

Тип	Описание
BIT	Строка фиксированной длины. Представляет собой фиксированное число битов. Любая строка, хранящаяся в любом столбце, определенном как BIT(<i>длина</i>), может состоять из любого числа битов, не превышающего того числа, которое мы обозначили здесь как <i>длина</i> и которое должно быть целым, не меньше 1. Максимально допустимая длина строки определяется конкретной СУБД. Когда вы записываете в столбец, определенный как BIT(<i>длина</i>), какую-нибудь строку, фактическая длина которой меньше определенной для данного столбца, СУБД, в отличие от того, что имело место для типа данных CHARACTER, скорее всего, выдаст сообщение об ошибке. В командах SQL строки битов записываются так же, как строки CHARACTER, но в строке BIT перед первой кавычкой должна стоять латинская буква «B». Например, B'01001011' – это строка типа BIT(8)
BIT VARYING	Строка переменной длины. Представляет собой любое число символов, не превышающее некоего фиксированного числа. Любая строка, хранящаяся в любом столбце, определенном как BIT VARYING(<i>длина</i>), может состоять из любого числа символов, не превышающего того числа, которое мы обозначили здесь как <i>длина</i> и которое должно быть целым, не меньше 1. Максимально допустимая длина строки определяется конкретной СУБД. Когда вы записываете в столбец, определенный как BIT VARYING(<i>длина</i>), какую-нибудь строку, фактическая длина которой меньше определенной для данного столбца, СУБД, аналогично тому, как это происходит для типа CHARACTER VARYING, хранит эту строку, как она есть, не дополняя ее пробелами. Например, строка B'0101' записалась бы в любой столбец BIT VARYING(8) как строка B'0101'

Битовые типы данных

Битовые типы данных (Bit String) служат для того, чтобы представлять двоичные числа. Любая строка битов отличается следующими характеристиками:

- является упорядоченной последовательностью, состоящей из какого-то числа, возможно из 0, битов;
- каждый бит имеет значение или 0, или 1;
- ее обычно применяют для хранения так называемых *больших бинарных объектов* (Binary Large Object – BLOB), в частности файлов приложений, оцифрованных звуков и изображений;
- в командах SQL любую строку битов следует заключать в одинарные кавычки;
- относится к одному из типов, перечисленных в табл. 3.6.

П Ни одна СУБД не пытается транслировать строки б итов, а оставляет их для приложений.

П Опытному программисту может пригодиться то, что кроме двоичной (основание системы счисления равно 2) можно применять и шестнадцатеричную форму (основание системы счисления равно 16) или гексоформу. Шестнадцатеричная система счисления применяется в качестве цифр десятичные цифры от 0 до 9 и еще латинские буквы от A до F включительно (регистр не имеет значения). Один гексознак эквивалентен 4 битам. В командах SQL строки битов в шестнадцатеричной форме должны иметь латинскую букву X перед первой кавычкой. Например, бинарная строка B'01001011' соответствует гексостроке X'4B'.



Битовыми типами данных в рассматриваемых коммерческих СУБД являются:

- Microsoft Access – yes, no, binary, OLE object;
- Microsoft SQL Server – binary, varbinary, image;
- Oracle – raw, long raw, blob, bfile;
- MySQL – blob, tinyblob, mediumblob, longblob;
- PostgreSQL – bit, varbit.

Точные числовые типы данных

Точные числовые типы данных (Exact numeric) применяются для представления точных числовых значений. Произвольное точное числовое значение имеет следующие характеристики:

- может быть положительным, отрицательным и нулем;
- является или целым, или рациональным числом, причем:
 - целым числом называется такое рациональное число, которое не содержит дробной составляющей (нет знаков после десятичной точки);
 - рациональным числом называется конечная десятичная дробь (есть знаки после десятичной точки);
- имеет произвольные, но фиксированные *точность* и *масштаб*, где:
 - точностью называется число значащих цифр в записи числа (то есть общее количество цифр в десятичной записи числа без учета десятичной точки);
 - масштабом называется число цифр справа от десятичной точки (очевидно, что для получения целого числа необходимо и достаточно установить масштаб равным 0, а также то, что масштаб не может быть больше точности);
- относится к одному из типов, перечисленных в табл. 3.7 (отдельные примеры приведены в примечаниях этого раздела).

Таблица 3.7. Точные числовые типы данных

Тип	Описание
NUMERIC	Представляет произвольное рациональное число, хранимое в столбце, который определяется как NUMERIC(<i>точность</i> [<i>, масштаб</i>]). Натуральное число (исключая 0), которое надо подставить как <i>точность</i> , ограничено сверху только требованиями конкретной СУБД. Натуральное число (включая 0), которое надо подставить как <i>масштаб</i> , не может превосходить точности. Если вы указываете этот тип данных, опустите масштаб, система по умолчанию назначит его равным 0, что сделает соответствующие числа целыми (Integer)
DECIMAL	Аналогичен типу NUMERIC, и некоторые СУБД определяют эти типы как эквивалентные. Разница, в зависимости от конкретной СУБД, может состоять в том, что СУБД может выбрать большее значение точности, чем указал пользователь в выражении DECIMAL(<i>точность</i> [<i>, масштаб</i>]). Значит, задавая <i>точность</i> , вы задаете только нижнюю границу для ее фактического значения. Имейте в виду, что DECIMAL и DEC – синонимы
INTEGER	Представляет произвольное целое число. Минимально допустимое и максимально допустимое значения для столбца, чей тип данных определен как INTEGER, зависят только от тех ограничений, которые налагает конкретная СУБД. При задании типа данных столбца как INTEGER не требуется вводить никаких аргументов. Имейте в виду, что INTEGER и INT– синонимы
SMALLINT	Во всем повторяет INTEGER, только в зависимости от конкретной СУБД соответствующий ему интервал допустимых значений может быть значительно уже. При задании типа данных произвольного столбца, как SMALLINT, тоже не надо вводить никаких аргументов

Таблица 3.8. Варианты точности и масштаба для рационального числа с фиксированной десятичной точкой 123.89

Спецификация Хранится столбца	
NUMERIC(5)	124
NUMERIC(5, 0)	124
NUMERIC(5, 1)	123.9
NUMERIC(5, 2)	123.89
NUMERIC(4, 0)	124
NUMERIC(4, 1)	123.9
NUMERIC(4, 2)	Выходит за пределы точности
NUMERIC(2, 0)	Выходит за пределы точности

П В табл. 3.8 показано, как хранится произвольное число 123.89 в столбцах с типом данных NUMERIC и с различными величинами точности и масштаба.

С Не заключайте числовые значения в кавычки.

П Если числа не участвуют в арифметических вычислениях, их следует хранить как строки. Например, телефонные номера и почтовые индексы надо хранить как строки символов. Дело в том, что так можно предотвратить безвозвратные потери данных. Например, если бы вы хранили почтовый индекс '02116' не как строку, а как число, то потеряли бы первый ноль.

П Вычисления, в которых участвуют только целые числа, выполняются гораздо быстрее, чем вычисления, в которых принимают участие действительные числа с фиксированной и плавающей десятичной точкой.



Точными числовыми типами данных в рассматриваемых коммерческих СУБД являются:

- Microsoft Access – decimal, integer, byte, long integer;
- Microsoft SQL Server – numeric, decimal, integer, smallint, bigint, tinyint;
- Oracle – numeric, decimal, integer, smallint, number;
- MySQL – numeric, decimal, integer, smallint, bigint, mediumint, tinyint;
- PostgreSQL – numeric, decimal, integer, smallint, bigint.

Действительные числовые типы данных

Действительные числовые типы данных (Approximate numeric), или числа с плавающей точкой, соответствующие приближенным действительным числам, применяются для того, чтобы хранить приближенные числовые значения. Любое *приближенное числовое значение* отличается следующими характеристиками:

- может представлять собой положительное или отрицательное число либо нуль;
- представляет собой рациональное приближение некоего действительного числа с плавающей точкой;
- его обычно применяют для того, чтобы формализовать или очень маленькие или, наоборот, очень большие количества чего-либо, или какие-нибудь научные вычисления;
- выражено в экспоненциальном представлении, то есть:
 - равно некому рациональному числу, умноженному на 10 в какой-то целой степени;
 - записано с помощью символа экспоненты E в форме $\alpha E \beta$, где:
 - рациональное число α , называемое *мантиссой*, выражает значащие цифры в десятичной записи числа;
 - целое число β , называемое *порядком*, выражает степень десятки;
 - и мантисса, и порядок могут иметь любой знак, например $2.5E2 = 2.5 \times 10^2 = 250$ (мантисса равна 2.5, а порядок – 2), но $-2.5E-2 = -2.5 \times 10^{-2} = -0.025$;

Таблица 3.9. Действительные числовые типы данных

Тип	Описание
<p>FLOAT</p>	<p>Представляет собой произвольное рациональное приближение действительного числа с плавающей точкой. Любое число с плавающей точкой хранится в столбце, который следует указывать как <code>FLOAT(<i>precision</i>)</code>. Считается, что вместо <i>precision</i> надо подставить значение точности, но выраженное не в количестве значащих десятичных цифр, а в количестве битов. Точность не должна быть меньше 1. Максимально допустимая величина точности определяется теми ограничениями, которые накладывает конкретная СУБД</p>
<p>REAL</p>	<p>Этот тип данных совпадает по своему определению с типом <code>FLOAT</code>, только здесь точность вводить не надо, так как ее автоматически определяет сама СУБД, то есть при указании данного типа не надо вводить никаких аргументов. Числа типа <code>REAL</code> часто называют числами одинарной точности с плавающей точкой</p>
<p>DOUBLE PRECISION</p>	<p>Этот тип данных совпадает по своему определению с типом <code>FLOAT</code>, только здесь точность вводить не надо, так как ее автоматически определяет сама СУБД. Вот почему точность <code>DOUBLE PRECISION</code> вдвое превышает точность <code>REAL</code> (числа <code>DOUBLE PRECISION</code> еще называют числами двойной точности с плавающей точкой). Это, в частности, означает, что при указании данного типа не надо вводить никаких аргументов</p>

- имеет фиксированную точность, но не имеет никакого масштаба как такового, где:

- учтено, что знак и величина экспоненты автоматически определяют масштаб;
- точность – это число бинарных битов, которое используется для того, чтобы хранить мантиссу;
- для преобразования десятичной точности в бинарную нужно умножить десятичную точность на 3.32193;
- для преобразования бинарной точности в десятичную следует умножить бинарную точность на 0.30103 (24 бита дают 7 знаков точности, а 53 бита – 15 знаков точности);

- относится к одному из типов, перечисленных в табл. 3.9.



Не заключайте числовые значения в кавычки.



Действительными числовыми типами данных в рассматриваемых коммерческих СУБД являются:

- Microsoft Access – `single`, `double`;
- Microsoft SQL Server – `float`, `real`;
- Oracle – `float`, `real`, `double precision`, `number`;
- MySQL – `float`, `real`, `double precision`;
- PostgreSQL – `real`, `double precision`.

Календарные типы данных

Для отображения даты и времени суток следует применять календарные типы данных (Datetime). Значения, относящиеся к одному из календарных типов, отличаются следующими характеристиками:

- время рассчитывается по отношению к скоординированному всемирному времени (UTC; Universal Coordinated Time), которое раньше называлось временем по Гринвичу (Greenwich Mean Time). Это время соответствует следующему требованию стандарта SQL-92: каждому сеансу SQL по умолчанию сопоставлен некий сдвиг по времени от UTC, чтобы он учитывался при определении начала и конца каждого сеанса (например, нормальный сдвиг

географической зоны от UTC для Сан-Франциско (Калифорния) равен -8 часов);

- дата формируется в соответствии с правилами Григорианского календаря (все коммерческие СУБД, рассматриваемые в настоящем издании, не признают дат, рассчитанных по другим календарям);
- расчет значений времени основан на 24-часовом формате, который иногда называют военным (применяйте обозначение 13:00, а не 1:00 PM);
- смысловые составляющие даты разделяются дефисами (-), а смысловые составляющие времени – двоеточием (:);
- любое из рассматриваемых значений относится к одному из типов, приведенных в табл. 3.10.

Таблица 3.10. Календарные типы данных

Тип данных	Описание
DATE	Тип данных для отображения даты. Каждое значение данных этого типа хранится в столбце, который следует описать как DATE, имеет три целочисленных поля (YEAR, MONTH, DAY), формат уууу-мм-дд и общую длину, равную 10. Например, 2002-06-14 означает 14 июня 2002 года. При указании этого типа данных не требуется вводить никаких аргументов
TIME	Тип данных для отображения времени суток. Каждое значение данных этого типа хранится в столбце, которому следует присвоить тип TIME, имеет три целочисленных поля (HOUR, MINUTE, SECOND), формат hh:mm:ss и общую длину, равную 8 знаков, включая двоеточия. Например, 22:06:57 означает 22 часа, 6 минут, 57 секунд. Для этого типа данных не требуется вводить никаких аргументов, но вы можете ввести значение одного необязательного аргумента TIME(<i>точность</i>), чтобы задать доли секунды. Вместо необязательного аргумента <i>точность</i> следует вводить натуральные числа, имея в виду необходимое число десятичных разрядов в записи дробной части секунды. Ясно, что <i>точность</i> не может быть меньше 0. Максимально допустимое значение <i>точности</i> зависит от выбора конкретной СУБД, но минимальное значение среди максимально допустимых значений <i>точности</i> по множеству тех СУБД, которые рассматриваются в настоящей книге, равно 6. Поля HOUR и MINUTE представляют собой натуральные числа, а поле SECOND – рациональное число в виде десятичной дроби. Формат значений произвольного столбца, чей тип данных определен как TIME, – это hh:mm:ss.ssss..., где hh обозначает 2 знака поля HOUR, mm – 2 знака поля MINUTE, ss.ssss... – знаки в записи значения поля SECOND (длина значения столбца TIME равна 8 знакам, включая двоеточия и, возможно, еще одно двоеточие и какое-то число знаков в дробной части секунды). Вот пример значения поля TIME с дробной частью секунды: 22:06:57.13333. В табл. 3.11 перечислены области допустимых значений рассмотренных полей

Таблица 3.10. Календарные типы данных (окончание)

Тип данных	Описание
TIMESTAMP	Представляет собой комбинацию значения данных типа DATE и значения данных типа TIME, разделенных пробелом. Формат данного типа таков: уууу-мм-дд hh:mm:ss. Общая длина любого значения этого типа равна 19 знаков, например, 2002-06-14 22:06:57 соответствует рассматриваемому формату и означает 22 часа, 6 минут, 57 секунд, 14 июня, 2002 года. Создавая столбец типа TIMESTAMP, вы можете задать один необязательный параметр – TIMESTAMP(<i>точность</i>). Вместо необязательного аргумента <i>точность</i> следует вводить натуральные числа, имея в виду необходимое вам число десятичных разрядов в записи дробной части секунды. Очевидно, что <i>точность</i> не может быть меньше 0. Максимально допустимое значение <i>точности</i> зависит от выбора конкретной СУБД, но минимальное значение среди максимально допустимых значений <i>точности</i> по множеству тех СУБД, которые рассматриваются в настоящей книге, равно 6. Тогда формат изменится на уууу-мм-дд hh:mm:ss.ssss..., длина значения столбца TIMESTAMP равна 20 знакам, включая двоеточия и, возможно, еще одно двоеточие и какое-то число знаков в дробной части секунды
TIME WITH TIME ZONE	Во всем эквивалентен типу TIME, только его формат включает дополнительное поле TIME_ZONE_OFFSET, чтобы показывать отклонение от UTC в часах. Это поле особенное. Дело в том, что его формат совпадает с форматом не поля, а целого интервального типа данных – INTERVAL HOUR TO MINUTE (интервалы длиной, выраженной в часах, с возможным добавком, выраженным в минутах) – и может содержать значения, указанные в табл. 3.11 (см. следующий раздел). Чтобы ввести любое значение в столбец, чей тип данных TIME WITH TIME ZONE, вам для присвоения какого-нибудь значения вашей временной зоне (то есть вашему часовому поясу) надо присоединить к предложению для формата типа данных TIME предложение AT TIME ZONE <i>time_zone_offset</i> . Вот пример задания значения столбца TIME WITH TIME ZONE: 22:06:57 AT TIME ZONE -08:00. Это означает 22 часа, 06 минут, 57 секунд местного времени, которому соответствует сдвиг от UTC, равный -8 часов 00 минут. Вы можете модифицировать формат TIME предложением AT LOCAL. В этом случае вы сообщите системе, что ваша временная зона соответствует той, которая установлена для текущего сеанса как зона по умолчанию. Если предложение AT опущено, считается, что для всех значений времени данного столбца этого типа данных по умолчанию задано AT LOCAL
TIMESTAMP WITH TIME ZONE	Во всем эквивалентен типу TIMESTAMP, только его формат содержит дополнительное поле TIME_ZONE_OFFSET, чтобы показывать отклонение от UTC в часах. Синтаксис здесь тот же, что и для типа данных TIME WITH TIME ZONE, только теперь вам надо еще включать в значение столбца какую-нибудь дату, например: 2002-06-14 22:06:57 AT TIME ZONE – 08:00

П О том, как определить системное время, рассказывается в разделе «Извлечение значений текущих даты и времени» главы 5.

П Можно сравнивать значения даты и времени, если в их форматы включены одни и те же поля. О том, как это сделать, рассказывается в главе 4 (раздел «Фильтрация строк с помощью предложения WHERE») и в главе 5 (раздел «Операции с данными даты и времени»).

П Поле SECOND может принимать значения до 61.999... включительно, а не 59.999, что, казалось бы, естественнее. Это сделано для того, чтобы в значение какой-нибудь конкретной минуты (хотя эта необходимость возникает редко и только в специальных случаях) можно было вводить так называемые секунды перескока, которые, в свою очередь, нужны для того, чтобы синхронизировать земные часы с так называемым звездным временем (существует определенная погрешность, поскольку на оборот Земли вокруг оси требуется примерно (а не точно) 24 часа, а на оборот Земли вокруг Солнца – 365 суток). Поля типа DATETIME и их допустимые значения приведены в табл. 3.11.

С Чтобы получить произвольную константу (литерал) даты и времени, напечатайте сначала имя конкретного типа данных даты и времени, потом пробел, затем значение нужной вам константы в одинарных кавычках, например: DATE 'yyyy-mm-dd', или TIME 'hh:mm:ss', или TIMESTAMP 'yyyy-mm-dd hh:mm:ss'.

П SQL-92 не воспринимает даты обратного исторического отсчета (то есть до новой эры/до рождества Христова, в английском обозначении – B.C.E/B.C), но ваша СУБД вполне может воспринимать и обрабатывать их.

Таблица 3.11. Поля типа DATETIME и их допустимые значения

Поле	Интервал допустимых значений
YEAR	от 0001 до 9999
MONTH	от 01 до 12
DAY	от 01 до 31
HOUR	от 00 до 23
MINUTE	от 00 до 59
SECOND	от 00 до 61.999... (см. советы)
TIME_ZONE_OFFSET	от -12:59 до +13:00

П

Отметки времени (timestamp) часто применяются для того, чтобы строить уникальные ключи, или для того, чтобы пометить события, связанные со строкой, в которую такая отметка входит в качестве столбца (например, в целях регистрации каких-то событий).

П

В применении типа данных `TIME WITH TIME ZONE` нет никакого резона, поскольку часовые пояса сами по себе не имеют никакого смысла, если только они не связаны с конкретной датой. Дело в том, что сдвиги часовых поясов от UTC зависят от времени года, в том числе и от так называемого сезонного времени. Поэтому, если надо учитывать сдвиг от UTC, используйте тип данных `TIMESTAMP WITH TIME ZONE`.



Календарными типами данных в рассматриваемых коммерческих СУБД являются:

- Microsoft Access – `date/time`;
- Microsoft SQL Server – `datetime`, `smalldatetime`;
- Oracle – `date`, `timestamp`;
- MySQL – `date`, `time`, `datetime`, `timestamp`;
- PostgreSQL – `date`, `time`, `timestamp`.

Коммерческие СУБД позволяют вводить данные в следующих форматах:

- месяц-день-год;
- день-месяц-год;
- и некоторых других.

Те же СУБД позволяют применять при вводе 24-часовой или 12-часовой формат (время AM/PM). При этом надо иметь в виду, что иногда формат ввода может отличаться от формата отображения.

В СУБД Microsoft Access заключайте литералы даты и времени в решетки `#`, а не в кавычки, и опускайте префикс в виде имени типа данных.

В СУБД Microsoft SQL Server при записи литералов даты и времени опускайте префикс в виде имени типа данных.

Интервальные типы данных



В отношении интервальных данных коммерческие СУБД, рассматриваемые в этой книге, соответствуют стандарту SQL-92 не более чем местами. Дело в том, что некоторые коммерческие СУБД имеют свои собственные, не подпадающие под SQL-92 типы интервальных данных, и свои собственные, не определенные в SQL-92 функции, которые вычисляют интервалы и производят все арифметические действия с данными даты и времени. Поэтому в настоящем разделе приводится справочная информация.

Интервальные типы данных следует применять для того, чтобы обозначать расстояние между датами или двумя отметками времени суток. Произвольное *интервальное значение* отличается следующими характеристиками:

- хранит количественную меру промежутка времени между двумя значениями даты или/и времени суток так, что если вы вычтете из одного значения другое, то получите этот самый интервал (например, между 09:00 и 13:00 интервал составляет 04:30, то есть 4 часа 30 минут);
- применяется для того, чтобы организовывать приращение какого-нибудь значения даты или/и времени (см. в главе 5 раздел «Операции с данными даты и времени»);
- его формат содержит те же самые поля, что и тип DATETIME (то есть YEAR, HOUR, SECOND и т.д.), и те же самые разделители, что и типы даты и времени, но численные значения, которые вводятся в эти поля, могут предваряться или знаком «+» (вперед), или знаком «-» (назад), чтобы показать соответствие направлений конкретного интервала и оси времени;
- имеет отношение к следующим двум уровням измерения времени:
 - интервал Year-month (год-месяц) выражен в годах и полном количестве месяцев;
 - интервал Day-time (дневное время) выражен в днях, часах, минутах и секундах;
- его квалификатор может состоять из единственного поля или из нескольких полей, причем:
 - однополевой квалификатор указывается как YEAR, или MONTH, или DAY, или HOUR, или SECOND, чтобы произвольный однополевой столбец, определенный как INTERVAL HOUR, смог бы содержать интервалы наподобие «4 часа» или «25 часов»;
 - многополевой квалификатор обозначается в виде start_field TO end_field, чтобы произвольный столбец, определяемый как, скажем, INTERVAL DAY TO MINUTE, мог бы содержать следующие интервалы: «2 дня, 5 часов, 10 минут», где:
 - вместо start_field будет проставлено или поле YEAR, или поле MONTH, или поле DAY, или поле HOUR, или поле MINUTE;
 - вместо end_field будет проставлено или поле YEAR, или поле MONTH, или поле DAY, или поле HOUR, или поле MINUTE, или поле SECOND, чтобы end_field определяло заведомо большую единицу измерения времени по сравнению с start_field;
- формат любого однополевого столбца может включать значение точности, определяющее длину (то есть количество позиций для знаков) единственного поля (например, INTERVAL HOUR(2)), причем:

Таблица 3.12. Интервальные типы данных

Тип	Описание
Year-month	Эти интервалы содержат или некое значение в годах, или в месяцах, или в годах и месяцах. Допустимыми типами столбцов для таких интервалов являются: INTERVAL YEAR, INTERVAL YEAR(<i>точность</i>), INTERVAL MONTH, INTERVAL MONTH(<i>точность</i>), INTERVAL YEAR TO MONTH, INTERVAL YEAR(<i>точность</i>) TO MONTH
Day-time	Эти интервалы могут содержать значения, выраженные в некой комбинации дней, часов, минуты секунд. Некоторые из допустимых типов столбцов для таких интервалов: INTERVAL MINUTE, INTERVAL DAY(<i>точность</i>), INTERVAL DAY TO HOUR, INTERVAL DAY(<i>точность</i>) TO SECOND, INTERVAL MINUTE(<i>точность</i>) TO SECOND (<i>дробная точность</i>)

- если значение точности опущено, система по умолчанию считает его равным 2;
- если единственное поле – это SECOND, его формат может дополнительно включать еще и значение дробной точности, записываемое при форматировании поля через запятую после первого значения точности, и определенное количество десятичных долей секунды справа от десятичной точки (по умолчанию эта дробная точность считается равной 6), например INTERVAL SECOND (5, 2);
- формат любого многопольного столбца может быть точным как для start_field, так и для end_field (если только end_field – это не SECOND, но тогда у end_field может быть дробная точность), например INTERVAL DAY(3) TO MINUTE или INTERVAL MINUTE(2) TO SECOND(4);
- относится к одному из типов, перечисленных в табл. 3.12.

C

Чтобы получить интервальный литерал, напечатайте INTERVAL, пробел, интервальное значение в одинарных кавычках, например: INTERVAL '15-3' для интервала, равного 15 годам и 3 месяцам в будущем, или INTERVAL '-22:06:5.5' для интервала 22 часа, 6 минут и 5,5 секунды тому назад.



Интервальными типами данных в рассматриваемых коммерческих СУБД являются:

- Microsoft Access – не поддерживает;
- Microsoft SQL Server – не поддерживает;
- Oracle – Interval;
- MySQL – не поддерживает;
- PostgreSQL – Interval.

Значение null

Чтобы как-то отобразить отсутствующие или неизвестные данные, вы можете воспользоваться значением `null`. Оно имеет следующие характеристики:

- в командах SQL ключевое слово `NULL` представляет значение `null`;
- значение `null` применяют вместо такого значения, которое неизвестно, никогда не будет известно или будет известно позже либо вообще неприемлемо по какой-то причине;
- значение `null` – это не ноль, не пустая строка (") и не строка пробелов (в отношении Oracle утверждение о пустых строках неверно; см. примечание с пометкой **DBMS** в конце этого раздела). Например, если какое-то значение столбца `price` равно `NULL`, это свидетельствует о том, что цена пока не известна или не определена, а не о том, что некий товар вообще не имеет никакой цены или его цена равна нулю;
- значение `null` не принадлежит ни к какому типу данных, и поэтому его можно записать в любой столбец, за исключением тех, которые определены как `NOT NULL` (см. раздел «Запрет значения `null` с помощью ограничения `NOT NULL`» в главе 10);
- значение `null` можно при необходимости найти и идентифицировать предложением `IS NULL` (см. раздел «Проверка на значение `null` с помощью оператора `IS NULL`» в главе 4);
- значения `null` не равны друг другу. Поэтому вы не сможете определить, соответствует ли какое-нибудь значение `null` другому значению в вашей базе. Интересно, что именно эта ситуация позволя-

ет построить модель трехзначной логики (см. раздел «Комбинирование условий с помощью операторов `AND`, `OR` и `NOT`» в главе 4);

- хотя ни одно значение `null` не равно никакому другому значению `null`, предложение `DISTINCT` воспринимает все значения `null` в любом произвольно выбранном столбце как копии одного и того же значения (см. раздел «Удаление повторяющихся строк с помощью ключевого слова `DISTINCT`» в главе 4);
- при сортировке произвольного столбца, содержащего `NULL`, эти значения в зависимости от СУБД, на которой вы работаете, будут либо больше, либо меньше любого из значений этого столбца, не являющихся значениями `null` (см. раздел «Сортировка строк с помощью предложения `ORDER BY`» в главе 4);
- значения `null` «размножаются» в процессе вычислений. Дело в том, что результатом многих выражений, операций и функций, в которых есть хотя бы одно значение `null`, будет `NULL`, например: $(12 * \text{NULL}) / 4 = \text{NULL}$ (см. главу 5);
- функции агрегатов данных, например `SUM()`, `AVG()` и `MAX()`, игнорируют `NULL` в процессе вычислений (см. главу 6);
- если столбец, по которому осуществляется группировка предложением `GROUP BY`, содержит значения `null`, все они будут помещены в одну отдельную группу (см. раздел «Группирование строк с использованием предложения `GROUP BY`» в главе 6);
- значения `null` влияют на результаты объединений (см. раздел «Использование объединений» в главе 7);
- значения `null` могут создавать проблемы в подзапросах (см. раздел «Значения `NULL` в подзапросах» в главе 8).

```
SELECT MAX(au_id)
FROM authors
WHERE au_lname = 'XXX'
```

```
MAX(au_id)
-----
NULL
```

Рис. 3.3. Пример получения значения null из столбца, для которого эти значения недопустимы

С Не употребляйте ключевое слово NULL в кавычках. Иначе ваша СУБД интерпретирует его как строку символов 'NULL', а не как значение null.


П Значение null можно получить даже из столбца, который не является обнуляемым (то есть является необнуляемым). Рассмотрим, например, рис. 3.3. Здесь столбец `au_id` таблицы `authors` не является обнуляемым, но команда `SELECT` возвращает NULL в качестве максимального `au_id`.

П Мы перечислили лишь малую часть проблем и сложностей, связанных со значениями null. Имея в виду эти проблемы, некоторые известные специалисты по базам данных рекомендуют не применять значения null, а использовать какие-нибудь значения по умолчанию или более сложные схемы замещения отсутствующих данных. Однако существуют задачи, для решения которых значения null необходимы. Поэтому настоящие профессионалы, хотя и не исключают применение значений null совсем, стараются делать это как можно реже. Отсюда вывод: если значения null применяются, будьте предельно осторожными при интерпретации результатов запросов.

П Дополнительная информация о значениях null приводится в разделах «Проверка на значение null с использованием функции `COALESCE()`» и «Сравнение выражений с помощью функции `NULLIF()`» главы 5.

П Любой столбец, в который разрешено вводить NULL, называется *обнуляемым*.

П Термин *нулевое значение* по отношению к значению null является некорректным и может привести к ошибкам. Помните, что любое значение null показывает, главным образом, *отсутствие значения*.

 Формат представления значений null, возникающих как результат какой-либо операции, отличается в различных коммерческих СУБД. Эти значения могут быть отображены как NULL, (NULL), <NULL> или просто пробел. Некоторые СУБД предложат вам самостоятельно выбрать формат отображения значений null.

Последняя версия СУБД Oracle распознает пустую строку (') как значение null. В будущем эта трактовка вполне может измениться. Вот почему Oracle рекомендует не принимать пустые строки за значения null. Кроме того, отождествление пустых строк и значений null может создать проблемы при переносе между разными СУБД. Возьмем для примера нашу типовую базу. Столбец `au_fname` таблицы `authors` определен как NOT NULL (то есть необнуляемый). В то же время в среде СУБД Oracle имя автора Kellsey (значение первичного ключа равно A06) занесено в виде одиночного пробела (' '), но в некоторых других СУБД имя этого же автора будет уже храниться в виде пустой строки (''). Подробнее об учебной базе данных читайте в разделе «Наша типовая база данных» главы 2.

ВЫБОР ДАННЫХ ИЗ ПРОИЗВОЛЬНОЙ ТАБЛИЦЫ

4

В этой главе мы будем изучать команду SELECT, выполняющую основную задачу в SQL, так как 90% работы с базой данных занимают выборка и манипулирование данными, осуществляемые этой самой командой, хотя, возможно, и сильно усложненной. Итак, команда SELECT выбирает строки, столбцы и значения из одной или нескольких таблиц произвольной базы данных. Приведем синтаксис этой команды:

```
SELECT columns
FROM tables
[JOIN joins]
[WHERE search_condition]
[GROUP BY grouping_columns]
[HAVING search_condition]
[ORDER BY sort_columns]
```

Поскольку каждое предложение здесь имеет особый смысл, разберем их по порядку:

- предложения SELECT, FROM, ORDER BY и WHERE – в этой главе;
- предложения GROUP BY и HAVING – в главе 6;
- предложение JOIN – в главе 7.

Прежде чем продолжить изучение темы, напомним, что *моноширинным курсивом* выделяются заменяемые идентификаторы и выражения, а в квадратные скобки заключаются необязательные предложения

(см. во введении подраздел «Дополнительные соглашения»).

По установившейся традиции из всех команд *запросами* будем иногда называть только команды SELECT, хотя вполне возможно, что в документации по некоторым СУБД и в справочной литературе *все* команды SQL могут именоваться запросами.

Отметим еще одно: хотя команда SELECT представляет собой мощный инструмент, она неопасна. Дело в том, что с ее помощью вы не сможете ни добавлять, ни удалять, ни изменять данные и объекты любой базы данных (представляющие опасность инструменты описываются в главе 9).

Листинг 4.1. Выбрать и перечислить названия городов, где живут авторы, занесенные в нашу типовую базу. Результаты см. на рис. 4.1

```

SELECT city
FROM authors;
```

```

city
-----
Bronx
Boulder
San Francisco
San Francisco
New York
Palo Alto
Sarasota
```

Рис. 4.1. Результат работы программы, приведенной в листинге 4.1

Листинг 4.2. Выбрать и перечислить имена и фамилии авторов, занесенных в нашу типовую базу, а также названия штатов и городов, где они живут. Результаты см. на рис. 4.2

```

SELECT au_fname, au_lname, city, state
FROM authors;
```

au_fname	au_lname	city	state
Sarah	Buchman	Bronx	NY
Wendy	Heydemark	Boulder	CO
Hallie	Hull	San Francisco	CA
Klee	Hull	San Francisco	CA
Christian	Kells	New York	NY
	Kellsey	Palo Alto	CA
Paddy	O'Furniture	Sarasota	FL

Рис. 4.2. Результат работы программы, приведенной в листинге 4.2

Выбор столбцов с помощью предложений SELECT и FROM

В этом разделе рассматривается простейшая форма команды SELECT. С ее помощью вы сможете выбрать один столбец, несколько столбцов или все столбцы произвольной таблицы. При этом предложение SELECT перечисляет в этой форме столбцы, которые надо показать, а предложение FROM обозначает таблицу, из которой следует выбирать эти самые столбцы.

Выбор одного столбца из таблицы

Напечатайте следующее (см. листинг 4.1 и рис. 4.1):

```

SELECT column
FROM table;
```

Подразумевается, что:

- идентификатор *column* будет заменен реальным именем какого-нибудь столбца;
- идентификатор *table* будет заменен именем таблицы, которая содержит столбец *column*.

Выбор нескольких произвольных столбцов в одной таблице

Следует напечатать (см. листинг 4.2 и рис. 4.2):

```

SELECT columns
FROM table;
```

Подразумевается, что:

- идентификатор *columns* будет заменен одним или несколькими реальными именами столбцов, разделенными запятыми;

■ идентификатор *table* будет заменен именем таблицы, которая содержит столбцы *columns*.

В результате выполнения последней команды выбранные столбцы будут распечатаны в том порядке, в каком они перечислены в *columns*, а не в том порядке, в котором они определены в таблице *table*.

Выбор всех столбцов в одной таблице

Напечатайте следующее (см. листинг 4.3 и рис. 4.3):

```
SELECT *
FROM table;
```

Подразумевается, что вместо *table* вы подставите имя какой-нибудь реальной таблицы.

В результате выполнения этой команды столбцы будут распечатаны в том порядке, в каком они представлены в таблице.

Листинг 4.3. Выбрать и перечислить все столбцы таблицы *authors*. Результаты выполнения листинга см. на рис. 4.3

Листинг

```
SELECT *
FROM authors;
```

П Имейте в виду, что при выборке столбцов из таблиц обязательными являются только предложения **SELECT** и **FROM**.

П Свойство замкнутости таблиц (о котором упоминалось в советах из раздела «Таблицы, столбцы и строки» главы 2) гарантирует, что результатом любой команды **SELECT** будет некая таблица.

П Результат, показанный на рис. 4.1, содержит дублированные строки, потому что в нашей типовой базе есть два автора, живущих в Сан-Франциско. Чтобы удалить дублированные строки, обратитесь к разделу «Удаление повторяющихся строк с помощью ключевого слова **DISTINCT**» в этой главе.

au_id	au_fname	au_lname	phone	address	city	state	zip
A01	Sarah	Buchman	718-496-7223	75 West 205 St	Bronx	NY	10468
A02	Wendy	Heydemark	303-986-7020	2922 Baseline Rd	Boulder	CO	80303
A03	Hallie	Hull	415-549-4278	3800 Waldo Ave, #14F	San Francisco	CA	94123
A04	Klee	Hull	415-549-4278	3800 Waldo Ave, #14F	San Francisco	CA	94123
A05	Christian	Kells	212-771-4680	114 Horatio St	New York	NY	10014
A06		Kellsey	650-836-7128	390 Serra Mall	Palo Alto	CA	94305
A07	Paddy	O'Furniture	941-925-0752	1442 Main St	Sarasota	FL	34236

Рис. 4.3. Результат работы программы, приведенной в листинге 4.3

Листинг 4.4. Назвать город, штат и страну каждого издателя

```

SELECT city, state, country
FROM publishers;
    
```

city	state	country
-----	-----	-----
New York	NY	USA
San Francisco	CA	USA
Hamburg	NULL	Germany
Berkeley	CA	USA

Рис. 4.4. Результат выполнения команды, представленной в листинге 4.4. Обратите внимание, что столбец `state` для Германии не имеет смысла (в этой стране не штаты, а земли). Поэтому в столбце `state` напротив Германии стоит ключевое слово `NULL`, которое в данном случае является идентификатором значения `null`, отличающегося от какого-нибудь «невидимого» значения, например пустой строки или строки пробелов

П

Те строки, которые вы получаете в результате исполнения наших запросов, могут быть упорядочены не так, как на соответствующих рисунках. Чтобы разобраться, в чем здесь дело, обратитесь к разделу «Сортировка строк с помощью предложения `ORDER BY`» в этой главе.

П

Для указания значения `null` в какой-либо таблице или в каком-либо результате мы будем применять ключевое слово `NULL` (см. раздел «Значение `null`» в главе 3, листинг 4.4 и рис. 4.4).

П

О том, как производить выборку столбцов из нескольких таблиц сразу, рассказывается в главе 7.

П

Все представленные в настоящей главе результаты являются не более чем сырыми неформатированными значениями. Например, значениям денежных сумм может не хватать знака валюты измерения, а числа могут иметь неподходящее количество десятичных разрядов. Дело в том, что форматируют данные не инструменты выборки данных, о которых мы сейчас ведем речь, а инструменты подготовки отчетов.

П

Применять команду `SELECT *` во встроенном языке SQL рискованно, потому что число столбцов в какой-нибудь таблице может измениться и это вызовет сбой в вашей программе. С другой стороны, команда `SELECT *` полезна в интерактивном SQL, особенно тогда, когда вы точно не знаете имена всех столбцов в какой-нибудь таблице.

П

Любая операция, которая выбирает определенные столбцы (то есть не все) из какой-нибудь таблицы, называется *проекцией*.

Создание псевдонимов столбцов с помощью предложения AS

Ранее при выдаче результатов запросов СУБД мы использовали заголовки столбцов по умолчанию. И при выдаче любого результата заголовком столбца по умолчанию является исходное имя столбца в той таблице, где он был определен при создании базы данных. Но для заголовков столбцов вы можете применять не только значения по умолчанию, но и псевдонимы. Чтобы создать псевдоним произвольного столбца, можно воспользоваться предложением AS.

Любой псевдоним столбца – это какое-нибудь имя (альтернативное, вспомогательное, идентификатор), которое вы указываете для того, чтобы управлять тем, как заголовки столбцов отображаются при выдаче результатов. Вам следует применять псевдонимы столбцов в тех случаях, когда исходные имена столбцов не имеют никакого смысла для непосвященного, или слишком длинные, или слишком короткие, или их очень трудно напечатать и запомнить и т.п.

Для управления отображением результатов определите псевдонимы столбцов в предложении SELECT команды SELECT. С этой целью в предложении SELECT сразу после исходного имени произвольного столбца напечатайте ключевое слово AS и нужный вам псевдоним, заключенный или в одинарные, или в двойные кавычки. Однако имейте в виду, что у этого правила есть вариации:

- вы можете совсем опустить кавычки, если псевдоним содержит только буквы, цифры и знаки подчеркивания в любой комбинации;
- кавычки обязательны, если псевдоним содержит пробелы, знаки пунктуации или специальные символы;
- само ключевое слово AS не является обязательным, например два следующих предложения эквивалентны:
 - SELECT au_fname AS "First name";
 - SELECT au_fname "First name".
- если вы хотите, чтобы какой-то столбец сохранил при распечатке результатов свой заголовок по умолчанию, опустите предложение AS, связанное с этим столбцом.

Создание псевдонимов столбцов

Следует напечатать:

```
SELECT column1 AS alias1,
       column2 AS alias2,
       column3 AS alias3,
       ...
       columnN AS aliasN
FROM table;
```

Предполагается, что:

- *column1, column2, column3, ..., columnN* будут заменены реальными (исходными, «родными») именами столбцов;
- *alias1, alias2, alias3, ..., aliasN* будут заменены соответствующими псевдонимами;
- *table* будет заменено именем той таблицы, которая содержит столбцы с названиями *column1, column2, column3, ..., columnN*.

Листинг 4.5. Ключевое слово AS обозначает произвольный псевдоним любого столбца для отображения результатов. Представленная команда демонстрирует альтернативные синтаксические конструкции предложения AS. Имеет смысл выбрать какую-то одну из этих конструкций и придерживаться ее

```
SELECT au_fname AS "First name",
       au_lname AS "Last name",
       city AS City,
       state,
       zip AS "Postal code"
FROM authors;
```

В листинге 4.5 показаны возможные синтаксические вариации предложения AS, а на рис. 4.5 – результат работы программы, приведенной в этом листинге.

Мы постараемся не опускать без надобности ключевое слово AS и заключать псевдонимы в двойные кавычки всегда, если не возникнет потребности в чем-либо другом. Смысл этого в единообразии и, следовательно, в большей прозрачности материала. Дополнительное преимущество заключается в независимости наших программ от среды СУБД (это называется переносимостью программ; см. советы в конце данного раздела). То есть программа, которая представлена в листинге 4.5, должна быть записана так:

```
SELECT au_fname AS "First name",
       au_lname AS "Last name",
       city AS "City",
       state,
       zip AS "Postal code"
FROM authors;
```

П Предложение AS применяют еще и для того, чтобы именовать производные столбцы (то есть те столбцы, значения которых определяются выражениями; подробнее о производных столбцах см. в разделе «Создание производных столбцов» в главе 5).

First name	Last name	City	state	Postal code
-----	-----	-----	----	-----
Sarah	Buchman	Bronx	NY	10468
Wendy	Heydemark	Boulder	CO	80303
Hallie	Hull	San Francisco	CA	94123
Klee	Hull	San Francisco	CA	94123
Christian	Kells	New York	NY	10014
	Kellsey	Palo Alto	CA	94305
Paddy	O'Furniture	Sarasota	FL	34236

Рис. 4.5. Результат выполнения команды, представленной в листинге 4.5

П С помощью предложения `AS` также можно создавать псевдонимы таблиц (подробнее об этом см. в разделе «Создание псевдонимов таблиц с помощью предложения `AS`» в главе 7).

П Никакой псевдоним не может изменить имени любого столбца произвольной таблицы.

П Зарезервированные слова тоже можно применять в качестве псевдонимов, но тогда их надо заключать в кавычки. Например, запрос `SELECT SUM(sales) AS "Sum" FROM titles;` использует в качестве псевдонима столбца зарезервированное слово `SUM`. Подробнее о таких словах мы говорили в разделе «Синтаксис SQL» главы 3.



В Microsoft Access и PostgreSQL ключевое слово `AS` является обязательным при создании псевдонимов столбцов или таблиц. Если, работая в среде Oracle или PostgreSQL, вам нужно будет заключить псевдоним в кавычки, придется использовать только двойные кавычки (одинарные в этом случае недопустимы).

Если произвольный псевдоним не заключен в кавычки, Oracle по умолчанию отобразит его заглавными буквами.

Имейте в виду, что Oracle автоматически обрежет справа любой псевдоним так, чтобы его длина была равна количеству знаков, определенному при создании таблицы для значений того столбца, которому вы назначаете этот псевдоним. Например, если бы вы задали тип данных для столбца как `CHAR(5)` и собрались бы назначить ему псевдоним `Postal code`, то этот псевдоним был бы распечатан как `Posta`.

Следует отметить, что разные СУБД могут иметь собственные ограничения на применение в псевдонимах спецсимволов, символов пунктуации и пробелов. Поэтому есть смысл проанализировать документацию по вашей СУБД с использованием ключей поиска `SELECT` или `AS`.

Листинг 4.6. Перечислить штаты, где живут авторы, занесенные в нашу типовую базу данных. Результат исполнения запроса показан на рис. 4.6

```
SELECT state
FROM authors;

state
-----
NY
CO
CA
CA
NY
CA
FL
```

Рис. 4.6. Результат исполнения запроса, представленного в листинге 4.6. Можно видеть дубликаты штатов CA и NY

Листинг 4.7. Перечислить различные штаты, в которых живут авторы, учитываемые в нашей типовой базе данных. Результат исполнения запроса показан на рис. 4.7

```
SELECT DISTINCT state
FROM authors;

state
-----
NY
CO
CA
FL
```

Рис. 4.7. Результат исполнения запроса, представленного в листинге 4.7. Здесь уже нет дубликатов CA, ни дубликатов NY

Удаление повторяющихся строк с помощью ключевого слова DISTINCT

Структура реляционных баз данных такова, что столбцы очень часто содержат одинаковые значения в разных строках. Поэтому вполне естественно желать от произвольного запроса такого результата, в котором каждая строка была бы уникальной. Например, если бы мы запустили запрос (см. листинг 4.6), который должен перечислить штаты, где живут авторы, занесенные в нашу типовую базу, результат был бы точно таким, как на рис. 4.6. Мы видим, что этот результат содержит ненужные дубликаты. Для устранения этого неудобства служит ключевое слово `DISTINCT`, с помощью которого можно убрать лишние значения из результата запроса.

Удаление дубликатов строк

Вам следует напечатать:

```
SELECT DISTINCT columns
FROM table;
```

Предполагается, что:

- вместо *columns* вы подставите одно или несколько реальных имен столбцов, перечисленных через запятую;
- вместо *table* подставите реальное имя той таблицы, из которой выбираются эти столбцы.

Реализация этого подхода продемонстрирована в листинге 4.7 и на рис. 4.7.

П

Имейте в виду, что если в предложение `SELECT DISTINCT` включить более одного столбца, то в результате уникальность любой строки будет определяться уникальностью соответствующей комбинации *всех* значений столбцов, включенных в предложение, на этой самой строке среди аналогичных комбинаций, соответствующих другим

строкам. В этой связи обратите внимание и сравните запрос, представленный в листинге 4.8 и на рис. 4.8, и запрос, представленный в листинге 4.9 и на рис. 4.9. Мы видим, что результат первого запроса содержит один дубликат строки, имеющей два столбца, а результат второго запроса дубликатов строк не содержит. Однако и в первом, и во втором результатах возможны повторяющиеся значения в столбцах.

П

Несмотря на то что значения `null` никогда не бывают равны друг другу (поскольку считаются неизвестными), предложение `DISTINCT`, напротив, считает их дубликатами. Вот почему команда `SELECT DISTINCT` вернет только одно значение `null`, независимо от того, сколько значений `null` она встретит (см. раздел «Значение `null`» в главе 3).

П

Синтаксис команды `SELECT` содержит и необязательное ключевое слово `ALL`. Однако часто оно не встречается. Дело в том, что это слово указывает на действие, которое является действием по умолчанию (то есть было бы выполнено и без всяких указаний): отобразить все строки, в том числе дубликаты. То есть запрос `SELECT columns FROM table;` эквивалентен запросу `SELECT ALL columns FROM table;`, а синтаксическая диаграмма такова: `SELECT [ALL | DISTINCT] columns FROM table;`.

П

Предложение `DISTINCT` можно применять вместе с функциями агрегирования (имеются в виду функции, аргументами которых являются множества значений данных; подробнее об этой функции предложения `DISTINCT` читайте в разделе «Исключение повторных значений с помощью предложения `DISTINCT`» в главе 6).

П

Если в таблице произвольной базы данных первичный ключ определен правильно, то, как нам теперь известно, каждая строка этой таблицы будет уникальной. Поэтому запросы `SELECT DISTINCT * FROM table;` и `SELECT * FROM table;` вернули бы для такой таблицы (ее имя здесь обозначено как `table`) один и тот же результат.

Листинг 4.8. Перечислить города и штаты, в которых живут авторы, занесенные в нашу типовую базу данных. Результат представлен на рис. 4.8

Листинг	
<pre>SELECT city, state FROM authors;</pre>	
city	state
-----	-----
Bronx	NY
Boulder	CO
San Francisco	CA
San Francisco	CA
New York	NY
Palo Alto	CA
Sarasota	FL

Рис. 4.8. Результат запроса, приведенного в листинге 4.8, содержит один дубликат строки для Сан-Франциско (Калифорния)

Листинг 4.9. Перечислить различные пары (город, штат), чтобы адрес каждого автора, занесенного в нашу базу, содержал такую пару в качестве элемента

Листинг	
<pre>SELECT DISTINCT city, state FROM authors;</pre>	
city	state
-----	-----
Bronx	NY
Boulder	CO
San Francisco	CA
New York	NY
Palo Alto	CA
Sarasota	FL

Рис. 4.9. Результат запроса, представленного в листинге 4.9. Обратите внимание, что уникальным здесь считается сочетание «город-штат», а не значение какого-либо столбца. По этому критерию здесь нет дубликатов строк

Порядок сортировки

Смысл сортировки по возрастанию и убыванию прозрачен для чисел и дат, а смысл сортировки строк символов довольно запутан. Для сортировки символов любая СУБД применяет так называемую упорядочивающую последовательность, или схему сличения знаков. Эта схема задает некое отношение предшествования ($a < b$ обозначает «а предшествует b»), но выбор алфавита зависит от того, какой язык вы применяете. Например, у европейских языков – латинский алфавит, у иврита – древнееврейский, а у китайского – иероглифы. Так вот, схема сличения знаков алфавита кроме упорядочивания его символов должна решить проблему чувствительности к регистру ('A' < 'a?'), проблему диакритического знака ударения ('Á' < 'A?'), проблему чувствительности к длине строки (для многобайтовых знаков или знаков Unicode) и некоторые другие вопросы, в частности особенности строения языка, диалекты и т.п. Имейте в виду, что в стандарте SQL не заданы ни алфавиты, ни схемы сличения символов. Поэтому каждая СУБД по умолчанию применяет собственную стратегию сортировки и собственную схему сличения символов. Обычно СУБД предоставляют в распоряжение пользователя команды и инструментарий, с помощью которых можно посмотреть на текущий алфавит и на текущую схему сличения. Например, в среде Microsoft SQL Server вы можете запустить команду `exec sp_helpsort`. А для получения исчерпывающей информации на эту тему вам следует проанализировать документацию по вашей СУБД с использованием ключей поиска «сличение» и «порядок сортировки».

Сортировка строк с помощью предложения ORDER BY

При выполнении запроса СУБД выдает строки в случайном порядке, так как реляционная модель определяет, что порядок строк не имеет никакого значения для табличных операций. Зато вы сами можете расположить строки с использованием предложения ORDER BY и отсортировать их по какому-нибудь столбцу или группе столбцов либо в восходящем (от самого нижнего к самому верхнему) либо в нисходящем (от самого верхнего к самому нижнему) порядке. О значении «верха и низа» подробнее говорится в тексте врезки этого раздела. Имейте в виду, что предложение ORDER BY всегда является последним предложением команды SELECT.

Выполнение сортировки по одному произвольному столбцу

Следует напечатать:

```
SELECT columns
FROM table
ORDER BY sort_column [ASC | DESC];
```

Предполагается, что:

- вместо *columns* вы подставите имя одного столбца или, через запятую, имена нескольких столбцов, которые надо выбрать;
- вместо *sort_column* вы подставите имя столбца, по которому хотите отсортировать результат запроса;
- вместо *table* вы подставите имя таблицы, содержащей и столбцы *columns*, и столбец *sort_column*, который, что важно, совсем не должен быть среди столбцов *columns*;
- определите восходящий порядок сортировки опцией ASC, или нисходящий

порядок сортировки опцией DESC, или не зададите никакого порядка сортировки, и тогда по умолчанию будет назначен порядок по возрастанию ASC (см. листинги 4.10, 4.11 и рис. 4.10, 4.11).

Сортировка по нескольким столбцам

Необходимо напечатать:

```
SELECT columns
FROM table
ORDER BY sort_column1 [ASC | DESC],
        sort_column2 [ASC | DESC],
        sort_column3 [ASC | DESC],
        ...
        sort_columnN [ASC | DESC];
```

Предполагается, что:

- вместо *columns* вы подставите имя одного столбца или, через запятую, имена нескольких столбцов, которые надо выбрать;
 - вместо *sort_column1*, *sort_column2*, *sort_column3*, ..., *sort_columnN* вы подставите в соответствии с правилами синтаксиса имена тех столбцов, по которым хотите отсортировать результат запроса;
- вместо *table* вы подставите имя той таблицы, которая содержит и столбцы *columns*, и все столбцы *sort_column1*, *sort_column2*, *sort_column3*, ..., *sort_columnN*, которые, что важно, вовсе не должны быть среди столбцов *columns*;
- для каждого из столбцов *sort_column1*, *sort_column2*, *sort_column3*, ..., *sort_columnN* вы определите восходящий порядок сортировки опцией ASC или нисходящий порядок сортировки опцией DESC либо не укажете никакого порядка

Листинг 4.10. Перечислить имена, фамилии, города и штаты проживания авторов, занесенных в нашу типовую базу данных, по алфавиту (этот порядок совпадает с порядком по умолчанию, что делает опцию ASC практически ненужной). Результат исполнения запроса см. на рис. 4.10

Листинг

SELECT au_fname, au_lname, city, state
FROM authors
ORDER BY au_lname ASC;

au_fname	au_lname	city	state
-----	-----	-----	-----
Sarah	Buchman	Bronx	NY
Wendy	Heydemark	Boulder	CO
Hallie	Hull	San Francisco	CA
Klee	Hull	San Francisco	CA
Christian	Kells	New York	NY
	Kellsey	Palo Alto	CA
Paddy	O'Furniture	Sarasota	FL

Рис. 4.10. Результат исполнения запроса, представленного в листинге 4.10. Фамилии отсортированы в порядке возрастания

Листинг 4.11. Перечислить имена, фамилии, города и штаты проживания авторов, которые занесены в нашу типовую базу данных, в нисходящем (то есть в обратном алфавитном) порядке. Здесь ключевое слово `DESC` является обязательным. Результат исполнения этого запроса см. на рис. 4.11

Листинг

```
SELECT au_fname, au_lname, city, state
FROM authors
ORDER BY au_fname DESC;
```

au_fname	au_lname	city	state
-----	-----	-----	-----
Wendy	Heydemark	Boulder	CO
Sarah	Buchman	Bronx	NY
Paddy	O'Furniture	Sarasota	FL
Klee	Hull	San Francisco	CA
Hallie	Hull	San Francisco	CA
Christian	Kells	New York	NY
	Kellsey	Palo Alto	CA

Рис. 4.11. Результат исполнения запроса, представленного в листинге 4.11. Сортировка выполнена по столбцу имен в нисходящем порядке. Обратите внимание, что у автора по фамилии Келлси (Kellsey) нет имени и оно представлено пустой строкой (“”). Поэтому Келлси стоит последним (вспомните о значении null). Он был бы первым, если бы сортировка проводилась тоже по именам, но в восходящем (то есть прямом алфавитном) порядке

сортировки, и тогда по умолчанию будет назначен восходящий порядок `ASC` (см. листинг 4.12 и рис. 4.12);

- в соответствии с определенным выше порядком СУБД сначала отсортирует все строки по `sort_column1`; потом СУБД отсортирует те строки, которые после предыдущего шага имеют равные значения `sort_column1`, по `sort_column2`;
- наконец, СУБД отсортирует те строки, которые после предыдущего шага имеют равные значения `sort_column(N-1)`, по `sort_columnN`.

Если столбец/столбцы, по которому/которым проводится сортировка, принадлежит/принадлежат множеству столбцов выборки, обозначенному нами как `columns`, SQL позволяет указать в предложении запроса `ORDER BY` этот/эти столбец/столбцы не по имени, а по расположению во множестве `columns`.

Выполнение сортировки по нескольким столбцам с указанием их местоположения

Необходимо напечатать:

```
SELECT columns
FROM table
ORDER BY sort_num1 [ASC | DESC],
        sort_num2 [ASC | DESC],
        sort_num3 [ASC | DESC],
        ...
        sort_numN [ASC | DESC];
```

Предполагается, что:

- вместо `columns` вы подставите имя одного столбца или, через запятую, имена нескольких столбцов, которые надо выбрать;

- вместо `sort_num1, sort_num2, sort_num3, ..., sort_numN` вы подставите натуральные числа, расположенные в интервале от 1 до числа, равного мощности множества `columns`, и соответствующие позициям во множестве `columns`, которые занимают столбцы `sort_num1, sort_num2, sort_num3, ..., sort_numN`;
- вместо `table` вы подставите имя таблицы, содержащей и столбцы `columns`, и все столбцы `sort_num1, sort_num2, sort_num3, ..., sort_numN`, которые, что важно, должны быть среди столбцов `columns`;
- для каждого из столбцов `sort_num1, sort_num2, sort_num3, ..., sort_numN` вы используете восходящий порядок сортировки опцией `ASC`, или нисходящий порядок сортировки опцией `DESC`, или не укажете никакого порядка сортировки, и тогда по умолчанию будет назначен восходящий порядок `ASC` (см. листинг 4.13 и рис. 4.13);
- СУБД сначала отсортирует в соответствии с определенным выше порядком все строки по `sort_num1`;
- потом СУБД отсортирует строки, которые после предыдущего шага имеют равные значения `sort_num1`, по `sort_num2`;
- наконец, СУБД отсортирует те строки, которые после предыдущего шага имеют равные значения `sort_num(N-1)`, по `sort_numN`.

Листинг 4.12. Перечислить имена, фамилии, города и штаты проживания авторов, которые занесены в нашу типовую базу данных, по убыванию городов внутри восходящего порядка штатов (то есть сначала сортируем в алфавитном порядке по штату, потом сортируем то, что попало в один штат, в обратном алфавитном порядке по городу). Результат см. на рис. 4.12

Листинг

```
SELECT au_fname, au_lname, city, state
FROM authors
ORDER BY state ASC,
        city DESC;
```

au_fname	au_lname	city	state
-----	-----	-----	----
Hallie	Hull	San Francisco	CA
Klee	Hull	San Francisco	CA
	Kellsey	Palo Alto	CA
Wendy	Heydemark	Boulder	CO
Paddy	O'Furniture	Sarasota	FL
Christian	Kells	New York	NY
Sarah	Buchman	Bronx	NY

Рис. 4.12. Результат исполнения запроса, представленного в листинге 4.12

Листинг 4.13. Перечислить имена, фамилии, города и штаты проживания авторов, занесенных в нашу типовую базу данных, так, чтобы результат был сначала отсортирован в восходящем (алфавитном) порядке по столбцу штата (столбец 4 в предложении SELECT), а потом в нисходящем (обратном алфавитном) порядке по столбцу фамилии (столбец 2 в предложении SELECT). Результат исполнения этого запроса см. на рис. 4.13

Листинг

```
SELECT au_fname, au_lname, city, state
FROM   authors
ORDER BY 4 ASC, 2 DESC;
```

au_fname	au_lname	city	state
	Kellsey	Palo Alto	CA
Hallie	Hull	San Francisco	CA
Klee	Hull	San Francisco	CA
Wendy	Heydemark	Boulder	CO
Paddy	O'Furniture	Sarasota	FL
Christian	Kells	New York	NY
Sarah	Buchman	Bronx	NY

Рис. 4.13. Результат исполнения запроса, представленного в листинге 4.13

П Стандарт SQL требует, чтобы при сортировке значений null они трактовались или как превосходящие (аналог максимума) все другие значения, не являющиеся значениями null, или как уступающие (аналог минимума) всем другим значениям, не являющимся значениями null. И поскольку требование стандарта SQL не является однозначным, одни СУБД трактуют значения null как наивысшие, а другие – как самые низшие (см. примечание **DBMS** в этом разделе, раздел «Значение null» главы 3, листинг 4.14 и рис. 4.14).

П Можно осуществлять сортировку по столбцам, не входящим в состав выбираемых столбцов, перечисленных в предложении SELECT (см. листинг 4.15 и рис. 4.15). Однако, если столбцы, по которым осуществляется сортировка, определены их относительным местоположением, они должны быть перечислены в предложении SELECT (см. листинг 4.13 и рис. 4.13).

П В предложении ORDER BY вы можете указать псевдонимы столбцов вместо их имен (см. листинг 4.16, рис. 4.16 и раздел «Создание псевдонимов столбцов с помощью предложения AS» в данной главе).

П Хотя это и нелепо, можно указать один и тот же столбец в предложении ORDER BY несколько раз.

П

Если значения в столбцах, по которым осуществляется сортировка предложением `ORDER BY`, повторяются, то строки, соответствующие этой комбинации значений, будут перечислены при выдаче результата в каком-нибудь случайном порядке. Хотя некоторые из наших примеров соответствуют как раз этому случаю (см. рис. 4.10, 4.12, 4.13), вы обязательно должны выбирать достаточно много столбцов в предложении `ORDER BY`, чтобы комбинация значений в столбцах сортировки была уникальной. Это особенно важно тогда, когда результат запроса должен быть предъявлен какому-нибудь конечному пользователю.

П

В соответствии со стандартом ANSI предложение `ORDER BY` является частью объявления `CURSOR` некоего курсора, а не команды `SELECT`. Но курсоры как таковые, будучи объектами, определяемыми внутри прикладных программ, не рассматриваются в этой книге. Однако хотелось бы подчеркнуть, что наиболее популярные направления практического применения SQL позволяют использовать предложение `ORDER BY` в любой команде `SELECT` (хотя бы потому, что СУБД строит курсор автоматически и невидимо для вас).

П

Вы можете выполнять сортировку и по результатам выражений. В главе 5 как раз рассказывается, как создавать выражения, применяя функции и операторы (см. листинг 4.17 и рис. 4.17).

П

В предложении `ORDER BY` вполне допустимо применять одновременно и имена столбцов, и относительные места расположения столбцов, и выражения.

Листинг 4.14. Значения `null`, находящиеся в столбце, по которому проводится сортировка, в зависимости от применяемой вами СУБД будут перечислены или самыми первыми, или самыми последними. Результат исполнения данного запроса см. на рис. 4.14

Листинг		
<pre>SELECT pub_id, state, country FROM publishers ORDER BY state ASC;</pre>		
pub_id	state	country
-----	-----	-----
P03	NULL	Germany
P02	CA	USA
P04	CA	USA
P01	NY	USA

Рис. 4.14. Результат исполнения запроса, представленного в листинге 4.14. Мы видим, что этот результат отсортирован по столбцу штата в восходящем (то есть в алфавитном) порядке. СУБД, в среде которой исполнен этот запрос, трактует любое значение `null` как низшее из всех возможных (как самую первую букву алфавита). Другая СУБД, которая трактует значение `null` как высшее из всех возможных (как самую последнюю букву алфавита), поставила бы ту же самую строку самой последней

Листинг 4.15. Обратите внимание, что столбец почтового индекса (zip) не входит в состав столбцов, которые надо выбрать. Результат исполнения этого запроса см. на рис. 4.15

Листинг

```
SELECT city, state
FROM authors
ORDER BY zip ASC;
```

city	state
-----	-----
New York	NY
Bronx	NY
Sarasota	FL
Boulder	CO
San Francisco	CA
San Francisco	CA
Palo Alto	CA

Рис. 4.15. Результат исполнения запроса, представленного в листинге 4.15. Обратите внимание: информация отсортирована по возрастанию почтового индекса (восходящий порядок сортировки), хотя на первый взгляд может показаться, что строки вообще никак не упорядочены. Такое впечатление складывается потому, что столбец, по которому осуществляется сортировка, не показан явно

Листинг 4.16. Использовать в предложении ORDER BY не имена столбцов, а их псевдонимы. Результат исполнения запроса см. на рис. 4.16

Листинг

```
SELECT au_fname AS "First name",
       au_lname AS "Last name",
       state
FROM authors
ORDER BY state ASC,
       "Last name" ASC,
       "First name" ASC;
```

First name	Last name	state
-----	-----	-----
Hallie	Hull	CA
Klee	Hull	CA
	Kellsey	CA
Wendy	Heydemark	CO
Paddy	O'Furniture	FL
Sarah	Buchman	NY
Christian	Kells	NY

Рис. 4.16. Результат исполнения запроса, представленного в листинге 4.16

П

Имейте в виду, что последовательность, в которой на экране появляются неупорядоченные строки, зависит от того, в каком виде они хранятся в своей таблице и в своей СУБД. Вам не следует полагаться на порядок по умолчанию, поскольку он меняется всякий раз, когда добавляются новые строки, когда старые строки модифицируются или когда создается какой-нибудь индекс.

П

Сортировка по столбцам, заданным их относительным местом расположения, полезна в запросах UNION (подробнее об этом мы поговорим в разделе «Комбинирование строк с помощью оператора UNION» главы 7).



Коммерческие СУБД Microsoft Access, Microsoft SQL Server и PostgreSQL при сортировке воспринимают значение null как низшее из всех возможных, а Oracle и MySQL – как высшее из всех возможных.

Коммерческие СУБД накладывают ограничения на тип столбцов в предложении ORDER BY. Так, в Microsoft SQL Server нельзя сортировать по столбцам ntext, text и image, а в Oracle нельзя сортировать по столбцам blob, clob, nclob и bfile. Для получения исчерпывающей информации об этих ограничениях следует проанализировать документацию по вашей СУБД с использованием ключей поиска SELECT и ORDER BY.

В среде Microsoft Access нельзя применять псевдонимы столбцов в предложении ORDER BY. Например, чтобы исполнить в Access запрос, который представлен в листинге 4.17, вам пришлось бы или перепечатать предложение ORDER BY в следующем виде: ORDER BY price * sales DESC, или обозначить столбец, по которому осуществляется сортировка, его относительным местом расположения: ORDER BY 4 DESC.

Листинг 4.17. Выполнить сортировку по выражению. Результат исполнения запроса см. на рис. 4.17

Листинг			
<pre>SELECT title_id, price, sales, price * sales AS "Revenue" FROM titles ORDER BY "Revenue" DESC;</pre>			
title_id	price	sales	Revenue
-----	-----	-----	-----
T07	23.95	1500200	35929790.00
T05	6.95	201440	1400008.00
T12	12.99	100001	1299012.99
T03	39.95	25667	1025396.65
T11	7.99	94123	752042.77
T13	29.99	10467	313905.33
T06	19.95	11320	225834.00
T02	19.95	9566	190841.70
T04	12.99	13001	168882.99
T09	13.95	5000	69750.00
T08	10.00	4095	40950.00
T01	21.99	566	12446.34
T10	NULL	NULL	NULL

Рис. 4.17. Результат исполнения запроса, представленного в листинге 4.17. Мы видим, что данные о книгах упорядочены в соответствии с уменьшением валового дохода по этим книгам (выражение, вычисляющее произведение цены и объема продаж)

Таблица 4.1. Типы условий

Условие	Операторы SQL
Сравнение	=, <>, <, <=, >, >=
Сопоставление с образцом	LIKE
Фильтрация диапазона	BETWEEN
Фильтрация списка	IN
Проверка на значение null	IS NULL

Фильтрация строк с помощью предложения WHERE

До сих пор в результат выполнения каждого запроса включались все строки, какие только есть в соответствующей таблице. Если не все из них вам нужны, можно применить предложение WHERE и отфильтровать лишнее из результата. Необходимо подчеркнуть, что именно способность фильтровать результаты характеризует команду SELECT как очень мощную. В предложении WHERE вам следует указать какое-нибудь *условие отбора*, включающее одно или несколько таких условий, которым должны удовлетворять строки произвольной таблицы, чтобы быть пропущенными через фильтр. Здесь под условием поиска, или *предикатом*, как его еще называют, понимается логическое выражение, которое может принимать три значения, а именно: «истина», «ложь» и «неизвестно» (по-английски, соответственно, true, false и unknown). При этом существенно, что значение «неизвестно» появилось в этой схеме благодаря значению null (подробнее об этом мы поговорим в следующем разделе). Фильтрование предложением WHERE осуществляется таким образом, что те и только те строки, для которых заданное условие принимает значение истины, включаются в результат исполнения запроса, а все остальные строки из этого результата исключаются.

Чтобы составить различные типы условий (см. табл. 4.1), SQL предоставляет в распоряжение программиста целый ряд операторов, где под произвольным оператором понимается набор символов или ключевое слово, определяющие конкретные действия, которые необходимо совершить над некими значениями или какими-либо

другими элементами (подробнее об операторах мы поговорим в главе 5). В этом разделе мы рассмотрим условия сравнения (см. табл. 4.1), а обо всех других условиях поговорим в конце главы. С помощью логических операторов AND, OR и NOT вы можете комбинировать и инвертировать любые условия (см. следующий раздел).

Любой *оператор сравнения* (см. табл. 4.2) должен сравнивать два значения и в результате этого сравнения принять одно из трех возможных значений: «истина», «ложь» или «неизвестно». При этом тип данных, к которому относятся сравниваемые значения, должен определить то, как именно эти значения сравниваются:

- строки символов сравниваются лексикографически, когда символ > означает «следует за...», а символ < – «предшествует» (см. разделы «Типы данных» в главе 3 и «Сортировка строк с помощью предложения ORDER BY» в данной главе);
- числа сравниваются арифметически, когда символ > означает «больше», а символ < – «меньше»;
- данные даты и времени сравниваются хронологически, когда символ > означает «позже», а символ < – «раньше» (при этом два сравниваемых значения даты и времени должны иметь один и тот же формат, то есть состоять из одних и тех же полей year, month, day и т.д.).

Имейте в виду, что сравнивать следует данные даты и времени только одного формата. Если вы попытаетесь сравнить разноформатные данные, ваша СУБД может:

- выдать сообщение об ошибке;
- сравнить эти значения неправильно и выдать результат совсем без строк;

Таблица 4.2. Операторы сравнения

Оператор	Описание
=	Равно
<>	Не равно
<	Меньше
<=	Не больше
>	Больше
>=	Не меньше

Листинг 4.18. Распечатать список авторов, занесенных в нашу типовую базу, чтобы фамилия каждого из них отличалась от Хал (Hull). Результат исполнения этого запроса см. на рис. 4.18

```

Листинг

SELECT au_id, au_fname, au_lname
FROM authors
WHERE au_lname <> 'Hull';
    
```

au_id	au_fname	au_lname
A01	Sarah	Buchman
A02	Wendy	Heydemark
A05	Christian	Kells
A06		Kellsey
A07	Paddy	O'Furniture

Рис. 4.18. Результат исполнения запроса, представленного в листинге 4.18

Листинг 4.19. Распечатать список названий из книг, занесенных в нашу типовую базу данных, для которых контракт пока не подписан. Результат исполнения этого запроса см. на рис. 4.19

```

Листинг

SELECT title_name, contract
FROM titles
WHERE contract = 0;
    
```

title_name	contract
Not Without My Faberge Egg	0

Рис. 4.19. Результат исполнения запроса, представленного в листинге 4.19

- попытаться преобразовать эти значения к единому формату и, если преобразование пройдет успешно, выдать результат сравнения или, если преобразование к единому формату выполнить не удастся, выдать сообщение об ошибке.

Фильтрация строк с помощью произвольного сравнения

Напечатайте:

```

SELECT columns
FROM table
WHERE test_column op value;
    
```

Предполагается, что:

- вместо *columns* вы подставите имя одного столбца или, через запятую, имена нескольких столбцов, которые надо выбрать;
- вместо *table* вы подставите имя той таблицы, которая содержит столбцы *columns*;
- формируя условие поиска, вы:
 - вместо *test_column* подставите имя какого-нибудь одного столбца в таблице *table*, причем этот столбец совсем не должен входить в состав столбцов *columns*;
 - вместо *op* подставите какой-нибудь из операторов сравнения, перечисленных в табл. 4.2;
 - вместо *value* подставите какое-нибудь значение, которое надо сравнить со значением столбца *test_column* (см. листинги и рис. 4.18– 4.20).

П Если в команде SELECT принимают участие и предложение ORDER BY, и предложение WHERE, следует печатать предложение WHERE перед предложением ORDER BY.

П Значение null никогда ни с чем не совпадает, в том числе с любым другим значением null. Поэтому строки, в которых есть значения null, через фильтр никогда не пройдут и среди строк результата не появятся. Чтобы извлечь строки, содержащие значения null, обратитесь к разделу «Проверка на значение null с помощью оператора IS NULL» в данной главе. Общую информацию о значениях null вы можете почерпнуть из раздела «Значение null» в главе 3.

П Как правая, так и левая часть сравнения (а сравнение – это часть предложения WHERE, которая не является ключевым словом WHERE) может иметь гораздо более сложную структуру по сравнению с той, что показана в начале текущего подраздела. Наиболее общая форма произвольного сравнения такова: `expr1 op expr2`, где `expr1` и `expr2` – выражения. А произвольное выражение – это, в свою очередь, произвольная, не запрещенная синтаксисом, комбинация имен столбцов, литералов, функций и операторов, которая на любой строке принимает единственное значение (короче говоря, выражение – это функция строки; см. листинг 4.21 и рис. 4.21). Подробнее о выражениях мы поговорим в главе 5.

П В предложении WHERE нельзя применять агрегатные функции, в частности SUM() или COUNT(). Подробнее об этом мы поговорим в главе 6.

П Любая операция, которая выбирает определенные строки из произвольной таблицы, называется ограничением.

Листинг 4.20. Распечатать названия книг, выпущенных не ранее 2001 года. Результат исполнения см. на рис. 4.20

```
SELECT title_name, pubdate
FROM titles
WHERE pubdate >= DATE '2001-01-01';
```

title_name	pubdate
Exchange of Platitudes	2001-01-01
Just Wait Until After School	2001-06-01
Kiss My Boo-Boo	2002-05-31

Рис. 4.20. Результат исполнения запроса, представленного в листинге 4.20

Листинг 4.21. Распечатать названия книг, валовая прибыль от продажи которых составила более миллиона долларов. Обратите внимание, что применяемое в этом запросе условие поиска использует некое арифметическое выражение. Результат исполнения запроса см. на рис. 4.21

```
SELECT title_name,
       price * sales AS 'Revenue'
FROM titles
WHERE price * sales > 1000000;
```

title_name	Revenue
Ask Your System Administrator	1025396.65
Exchange of Platitudes	1400008.00
I Blame My Mother	35929790.00
Spontaneous, Not Annoying	1299012.99

Рис. 4.21. Результат исполнения запроса, представленного в листинге 4.21



Хотя стандарт SQL гласит, что регистр имеет значение внутри закавыченных строк символов, в конечном счете только схема сличения вашей СУБД будет определять, чувствительны ли к регистру сравнения строк символов ('A' = 'a') или нет ('A' ≠ 'a'). Отметим, что по умолчанию Microsoft Access, Microsoft SQL Server и MySQL выдают нечувствительные к регистру сравнения строк символов, а Oracle и PostgreSQL – наоборот, чувствительные к регистру сравнения строк символов.

В среде Microsoft Access, печатая литералы даты и времени, следует опускать ключевое слово DATE, а сами литералы нужно заключать не в кавычки, а в символы #. Поэтому, чтобы выполнить в этой среде запрос, представленный в листинге 4.20, замените дату в предложении WHERE следующей #2001-01-01#.

В среде Microsoft SQL Server, печатая литералы даты и времени, следует опускать ключевое слово DATE. Поэтому, чтобы выполнить в этой среде запрос, представленный в листинге 4.20, замените дату в предложении WHERE на '2001-01-01'.

Чтобы в среде PostgreSQL сравнить значение произвольного столбца, где типом данных определен NUMERIC или DECIMAL, с неким действительным числом (то есть с рациональным числом с плавающей десятичной точкой), вам придется преобразовать это число в тип NUMERIC или DECIMAL (целые и рациональные числа с фиксированной точкой). Подробно эта тема раскрыта в разделе «Преобразование типов данных с помощью функции CAST()» главы 5.

Комбинирование условий с помощью операторов AND, OR и NOT

Практика показывает, что определять несколько условий в одном предложении WHERE приходится довольно часто. В частности, у вас возникнет такая необходимость, когда вы захотите отобрать строки, чтобы они удовлетворяли некоему ограничению, основанному на значениях сразу нескольких столбцов. В этом и других подобных случаях вы можете применить операторы AND и OR, чтобы скомпоновать несколько условий в одно *объединенное условие*. Операторы AND и OR вместе с оператором NOT называются *логическими*, или *Булевыми, операторами*. Они предназначены для работы с так называемыми значениями истины, которых существует всего три: «истина», «ложь» и «неизвестно».

Если вы программировали на других языках (или изучали логику высказываний, называемую пропозициональной), то знакомы с системой двухзначной логики, обозначаемой как система 2VL. Такая логика именно потому и называется двухзначной, что в ее системе результатом любого выражения всегда является только одно из двух: или «истина», или «ложь». То есть 2VL предполагает абсолютное знание, когда про любое высказывание точно известно, истинно оно или ложно. Но базы данных моделируют реальный мир, наши знания о котором всегда были и будут неполными. Вот почему в базах данных появились значения null. Они моделируют неопределенность реального мира, выраженную в неизвестных значениях (см. раздел «Значение null» в главе 3).

Итак, поскольку система 2VL не может смоделировать реальный недостаток знаний, язык SQL применяет трехзначную

логику (3VL). Основное отличие трехзначной логики от двухзначной состоит в том, что результатом любого логического выражения в рамках трехзначной логики может быть одно, но уже не из двух, а из трех значений: или «истина», или «ложь», или «неизвестно». При этом в запросах трехзначная логика работает так, что, если и только если на произвольной строке выражение истинно, эта строка включается в результат исполнения запроса. В двух других случаях она из результата запроса исключается. Однако строки со значениями null тоже можно извлечь. Чтобы понять, как это делается, обратитесь к разделу «Проверка на значение null с помощью оператора IS NULL» в этой главе.

Оператор AND

Перечислим основные характеристики оператора AND:

- объединяет два условия и принимает значение true (истина) тогда и только тогда, когда каждое из этих условий принимает значение true;
- в табл. 4.3, называемой таблицей истинности, перечислены возможные варианты объединения двух условий оператором AND, где:
 - крайний левый столбец показывает меру истинности первого условия;
 - верхняя строка показывает меру истинности второго условия;
 - каждая клетка отображает результат объединения оператором AND этих условий;
- с помощью нужного количества операторов AND можно объединить любое количество условий, и для того, чтобы произвольная строка была включена в итоговый результат, каждое из этих условий должно принять значение true;

Таблица 4.3. Таблица истинности AND

AND	True	False	Unknown
True	True	False	Unknown
False	False	False	False
Unknown	Unknown	False	Unknown

Листинг 4.22. Распечатать названия книг-биографий, которые продаются дешевле 20 долларов за каждую. Результат исполнения этого запроса см. на рис. 4.22

Листинг

```
SELECT title_name, type, price
FROM titles
WHERE type = 'biography' AND price < 20;
```

title_name	type	price
-----	-----	-----
How About Never?	biography	19.95
Spontaneous, Not Annoying	biography	12.99

Рис. 4.22. Результат исполнения запроса, представленного в листинге 4.22

- коммутативен (то есть его результат не зависит от порядка записи первого и второго условий – предложение WHERE condition1 AND condition2 эквивалентно предложению WHERE condition2 AND condition1);
- вы можете заключить в круглые скобки одно, или оба, или ни одного из объединяемых оператором AND условий, но имейте в виду: несмотря на то что круглые скобки не являются обязательными, возникают ситуации, когда они просто необходимы (подробнее об этом читайте в примечаниях к данному разделу).

Примеры использования оператора AND приведены в листингах 4.22–4.23 и на рис. 4.22–4.23.

Оператор OR

Перечислим отличительные черты оператора OR:

- объединяет два условия и принимает значение true (истина) тогда и только тогда, когда хотя бы одно из этих условий принимает значение true;
- таблица истинности оператора OR представлена в табл. 4.4;
- с помощью соответствующего количества операторов OR можно объединить любое количество условий так, чтобы результирующее условие приняло значение true тогда и только тогда, когда хотя бы одно составляющее условие приняло значение true;
- коммутативен (то есть его результат не зависит от порядка записи первого и второго условий точно так же, как у оператора AND);

- вы можете заключить в круглые скобки одно, или оба, или ни одного условия из объединяемых оператором OR (некоторые составные условия требуют круглых скобок).

Примеры использования оператора OR приведены в листингах 4.24–4.25 и на рис. 4.24–4.25. Обратите внимание, что листинг 4.25 демонстрирует влияние на результат запроса того факта, что в условиях предложения WHERE, объединенных оператором OR, есть значения null. Вы вполне могли бы ожидать в качестве результата запроса, представленного в листинге 4.25, распечатки всех строк таблицы publishers. Но на рис. 4.25 нет строки с первичным ключом P03. Это произошло потому, что отсутствующая строка содержит значение null в столбце state (напомним еще раз: значение null стоит там потому, что в Германии нет штатов). Поскольку, как мы теперь знаем, на этом самом значении null оба условия предложения WHERE принимают значение unknown и, следовательно, ни одно из них не равно true, та строка, для которой это имеет место, должна быть исключена из результата запроса. Чтобы как следует разобраться в этой логике, внимательно прочитайте раздел «Проверка на значение null с помощью оператора IS NULL» в данной главе.

Оператор NOT

Перечислим основные характеристики оператора NOT:

- в отличие от операторов AND и OR, не связывает двух условий, а инвертирует одно-единственное условие;
- таблица истинности оператора NOT представлена в табл. 4.5;

Листинг 4.23. Перечислить из нашей типовой базы данных имена, фамилии и штаты проживания авторов, чьи фамилии начинаются с H–Z. Результат исполнения этого запроса см. на рис. 4.23

Листинг		
<pre>SELECT au_fname, au_lname, state FROM authors WHERE au_lname >= 'H' AND au_lname <= 'Zz' AND state <> 'CA';</pre>		
au_fname	au_lname	state
-----	-----	----
Wendy	Heydemark	CO
Christian	Kells	NY
Paddy	O' Furniture	FL

Рис. 4.23. Результат исполнения запроса, представленного в листинге 4.23. Рассматривая этот результат, вспомните, что результат сравнения строк символов зависит от схемы сличения вашей СУБД (см. раздел «Сортировка строк с помощью предложения ORDER BY» в данной главе)

Таблица 4.4. Таблица истинности OR

OR	True	False	Unknown
True	True	True	True
False	True	False	Unknown
Unknown	True	Unknown	Unknown

Таблица 4.5. Таблица истинности NOT

Условие (Condition)	Инверсия (NOT Condition)
True	False
False	True
Unknown	Unknown

Листинг 4.24. Распечатать из нашей типовой базы данных имена и фамилии авторов, которые живут или в штате Нью-Йорк, или в штате Колорадо, или в городе Сан-Франциско. Результат исполнения данного запроса см. на рис. 4.24

Листинг			
<pre>SELECT au_fname, au_lname, city, state FROM authors WHERE (state = 'NY') OR (state = 'CO') OR (city = 'San Francisco');</pre>			
au_fname	au_lname	city	state
-----	-----	-----	-----
Sarah	Buchman	Bronx	NY
Wendy	Heydemark	Boulder	CO
Hallie	Hull	San Francisco	CA
Klee	Hull	San Francisco	CA
Christian	Kells	New York	NY

Рис. 4.24. Результат исполнения запроса, представленного в листинге 4.24

- в сравнениях следует ставить оператор NOT перед именем столбца или выражением, значение которого предполагается инвертировать, но не перед соответствующим оператором (например, предложение WHERE NOT state = 'CA' скомпоновано правильно, а предложение WHERE state NOT = 'CA' — неверно, хотя и читается «естественнее»);
- всегда действует только на одно условие, поэтому, чтобы инвертировать несколько условий, придется повторить для каждого из них отдельный оператор NOT. Например, чтобы перечислить все названия книг, которые одновременно

Листинг 4.25. Распечатать идентификаторы, названия, штаты и страны тех издательств, которые или находятся в штате Калифорния, или не находятся в штате Калифорния. Этот, на первый взгляд, бессмысленный запрос был специально придуман для того, чтобы продемонстрировать последствия попадания значений null в условия предложения WHERE, соединенные оператором OR. Результат исполнения запроса см. на рис. 4.25

Листинг			
<pre>SELECT pub_id, pub_name, state, country FROM publishers WHERE (state = 'CA') OR (state <> 'CA');</pre>			
pub_id	pub_name	state	country
-----	-----	-----	-----
P01	Abatis Publishers	NY	USA
P02	Core Dump Books	CA	USA
P04	Tenterhooks Press	CA	USA

Рис. 4.25. Результат исполнения запроса, представленного в листинге 4.25

не являются биографиями и оцениваются не ниже 20 долларов, вам пришлось бы напечатать:

```
SELECT title_id, type, price
FROM titles
WHERE NOT type = 'biography'
      AND NOT price < 20;
```

что было бы правильным решением поставленной задачи. Но если бы вы напечатали:

```
SELECT title_name, type, price
FROM titles
WHERE NOT type = 'biography'
      AND price < 20;
```

то получили бы запрос, написанный корректно с точки зрения синтаксиса, но работающий неправильно с точки зрения решения поставленной задачи. Просто его результат был бы неверным;

- применять или не применять оператор NOT в сравнениях – это дело вкуса и стиля. Например, предложения WHERE NOT state = 'CA' и WHERE state <> 'CA' эквивалентны;
- инвертируемое условие разрешается заключать в круглые скобки.

Примеры использования оператора NOT приведены в листингах 4.26–4.27 и на рис. 4.26–4.27.

Одновременное использование операторов AND, OR и NOT

SQL позволяет строить сложные условия посредством совместного использования операторов AND, OR, NOT в одном логическом выражении. При этом ваша СУБД будет определять очередность применения этих операторов на основе правил предшествования SQL. Об этих правилах подробно рассказывается в разделе «Определение последовательности вычисления» главы 5. Сейчас вы должны уяснить, что при вычислении значения сложного логического выражения, содержащего несколько разных логических операторов, при прочих равных условиях первым применяется оператор NOT, затем оператор AND, а последним – оператор OR. Мы использовали выражение «... при прочих равных условиях...», так как можно изменить порядок вычисления операторов, просто расставив круглые скобки. Общее правило таково, что сначала надо вычислить выражения во всех скобках, а потом выражение, составленное из этих скобок,

Листинг 4.26. Перечислить из нашей типовой базы данных фамилии и имена авторов, которые не живут в Калифорнии. Результат исполнения этого запроса см. на рис. 4.26

Листинг		
<pre>SELECT au_fname, au_lname, state FROM authors WHERE NOT (state = 'CA');</pre>		
au_fname	au_lname	state
-----	-----	----
Sarah	Buchman	NY
Wendy	Heydemark	CO
Christian	Kells	NY
Paddy	O'Furniture	FL

Рисунок 4.26. Результат исполнения запроса, представленного в листинге 4.26

Листинг 4.27. Распечатать названия книг, цена которых не ниже 20 долларов и объем продаж которых превышает 15 000 копий. Результат исполнения этого запроса см. на рис. 4.27

Листинг		
<pre>SELECT title_name, sales, price FROM titles WHERE NOT (price < 20) AND (sales > 15000);</pre>		
title_name	sales	price
-----	-----	-----
Ask Your System Administrator	25667	39.95
I Blame My Mother	1500200	23.95

Рис. 4.27. Результат исполнения запроса, представленного в листинге 4.27

Листинг 4.28. Если бы мы захотели перечислить книги из нашей базы, темой которой является или история, или биография и цена которых ниже 20 долларов, то этот запрос не сработал бы. Причина состоит в том, что по умолчанию оператор AND выполняется раньше оператора OR. Результат исполнения этого запроса см. на рис. 4.28

Листинг		
<pre>SELECT title_id, type, price FROM titles WHERE type = 'history' OR type = 'biography' AND price < 20;</pre>		
title_id	type	price
-----	-----	-----
T01	history	21.99
T02	history	19.95
T06	biography	19.95
T12	biography	12.99
T13	history	29.99

Рис. 4.28. Результат исполнения запроса, представленного в листинге 4.28. Мы видим, что в распечатку включены две книги, цена которых превышает 20 долларов. Следовательно, данный запрос не решает поставленной задачи и его надо откорректировать

объединенных логическими операторами. Например, на основе правил предшествования, которые приняты вашей СУБД по умолчанию, сложное условие $x \text{ AND NOT } y \text{ OR } z$ эквивалентно сложному условию $(x \text{ AND } (\text{NOT } y)) \text{ OR } z$. В этой связи хотелось бы подчеркнуть, что для того, чтобы последовательность применения операторов была максимально прозрачной, разумно всегда применять скобки и не полагаться на правила предшествования, действующие по умолчанию. Например, если бы нужно было распечатать из нашей базы названия книг, тема которых или история, или биография, и чтобы они при этом стоили не более 20 долларов каждая, то запрос, который представлен у нас в листинге 4.28, этой задачи не решил бы. Все дело в том, что по умолчанию (то есть без скобок) оператор AND вычисляется раньше, чем оператор OR, поэтому весь запрос листинга 4.28 будет выполнен в такой последовательности:

1. Найти и выбрать все книги-биографии, которые стоят менее 20 долларов.
2. Найти и выбрать все книги по истории (независимо от их цены).
3. В качестве результата распечатать оба множества строк со столбцами `title_id`, `type`, `price` (уникальный идентификатор книги, ее тема, цена; см. рис. 4.28).

Чтобы заставить запрос выбирать строки в соответствии с поставленной задачей, оператор OR должен срабатывать раньше оператора AND. В соответствии с этой идеей легко «починить» запрос листинга 4.28 расстановкой скобок. Исправленный запрос, представленный в листинге 4.29, выполняется в такой последовательности:

1. Найти и выбрать все книги-биографии и все книги по истории (независимо от их цены).

2. Из всего множества книг, прошедших фильтр на предыдущем шаге, выбрать только те, которые стоят менее 20 долларов.
3. В качестве результата распечатать последнее отобранное множество строк со столбцами `title_id`, `type`, `price` (уникальный идентификатор книги, ее тема, цена; см. рис. 4.29).

П

Хотя во всех примерах данного раздела операторы `AND`, `OR` и `NOT` работают исключительно с условиями сравнения, они могут работать и с условиями любого другого типа.

П

Одна очень распространенная ошибка состоит в том, чтобы при формировании предложения `WHERE` перед оператором `OR` ставить условие, а после оператора `OR` — не условие, а только его часть в виде литерала или функции. Пример этой ошибки: `WHERE state = 'NY' OR 'CA'`. Правильное решение: `WHERE state = 'NY' OR state = 'CA'`.

П

Дословный перевод вполне осмысленных утверждений языка общения на язык SQL ведет к бессмысленным и даже неверным командам SQL. Например, директива «Перечислить все книги по ценам менее 10 долларов за копию и более 30 долларов за копию» имеет четкий прикладной смысл. Но примененный в ней союз «и», будучи переведен на SQL, превратит весь запрос в бессмыслицу:

```
SELECT title_name, price
FROM titles
WHERE price<10 AND price>30;
```

Очевидно, что результатом этого запроса (синтаксически абсолютно корректного) является пустое множество строк, поскольку трудно представить себе такую книгу, которая *одновременно* (добавим к этому — в одной и той же стране и у одного и того

Листинг 4.29. Чтобы «исправить» запрос, представленный в листинге 4.28, были введены круглые скобки, чтобы заставить СУБД выполнить оператор `OR` раньше, чем оператор `AND`. Результат исполнения этого запроса см. на рис. 4.29

Листинг		
<pre>SELECT title_id, type, price FROM titles WHERE (type = 'history' OR (type = 'biography') AND price < 20;</pre>		
title_id	type	price
-----	-----	-----
T02	history	19.95
T06	biography	19.95
T12	biography	12.99

Рис. 4.29. Результат исполнения откорректированного запроса, представленного в листинге 4.29. Задача решена

Таблица 4.6. Логическая эквивалентность условий

Условие	Эквивалентное условие
NOT (p AND q)	(NOT p) OR (NOT q)
NOT (p OR q)	(NOT p) AND (NOT q)
NOT (NOT p)	p

же продавца) стоила бы и менее 10 долларов, и более 30 долларов. Но применение оператора AND требует от СУБД именно такого понимания. В то же время, если подумать над смыслом директивы на русском языке, становится ясно, что при переводе ее на SQL условия отбора книг по критерию «цены» надо соединять не оператором AND (потому что по-русски было бы не «цена и цена», а «книги и книги»), а оператором OR, так как именно он выбирает книги, удовлетворяющие хотя бы одному из критериев, налагаемых на цену книг, а не обоим критериям одновременно (в русском варианте смысл критерия «книги и книги» \neq «книги по цене 1 или по цене 2»): WHERE price<10 OR price>30. Если мы заменим этим предложением соответствующее предложение WHERE в приведенном запросе, то он заработает правильно.



С помощью операторов одно и то же условие можно выразить разными способами (см. табл. 4.6). Первые две эквивалентности называются *логическими законами Моргана*, а последняя эквивалентность — *двойным отрицанием*.

А теперь немного отвлечемся от темы книги. Применяя математическую логику и приведенные в табл. 4.6 правила эквивалентности, вы всегда сможете перенести оператор NOT в глубь любого сложного условия, то есть преобразовать это условие так, что оператор NOT будет применяться только к выражениям, не содержащим логических операторов. Приведем пример такого преобразования:

$$\begin{aligned}
 & \text{NOT } ((p \text{ AND } q) \text{ OR } (\text{NOT } p \text{ AND } r)) = \\
 & = \text{NOT } (p \text{ AND } q) \text{ AND NOT } (\text{NOT } p \text{ AND } r) = \\
 & = (\text{NOT } p \text{ OR NOT } q) \text{ AND } (p \text{ OR NOT } r)
 \end{aligned}$$



Если вы работаете в среде MySQL, имейте в виду, что False AND Unknown будет равно Unknown, а не False, как это, казалось бы, следует из табл. 4.3.

Сравнение по шаблону оператором LIKE

Все приведенные нами запросы были примерами того, как выбирать строки на основе знания точных значений какого-нибудь столбца или столбцов. Но, если точные значения не известны или известны лишь частично, рассмотренные выше приемы не сработают. В этом случае вам пригодится оператор `LIKE`, который позволяет решать задачи выборки на основе неполной информации (кто-то говорит вам: «Имя этого автора *начинается на Кел...*») и выбирать строки, значения которых *похожи* друг на друга в каком-либо смысле (вам нужно ответить на вопрос: «Кто из авторов, занесенных в нашу базу, живет *в районе* Сан-Франциско?»). Перечислим основные характеристики оператора `LIKE`:

- работает только со строками символов, но не с числами и не со значениями даты и времени;
- всегда применяет какой-нибудь шаблон значений, которому должны соответствовать отбираемые строки. Шаблоном называется произвольная закавыченная строка, содержащая какую-либо комбинацию буквенных символов, соответствие которым должно быть абсолютным, и произвольную комбинацию метасимволов. Здесь под *метасимволами* понимаются специальные операторы, которые работают как подстановочные знаки и нужны для того, чтобы описывать структуру данных. В шаблоне значений такой метасимвол задает определенную структуру того элемента данных, который должен быть подставлен на его место в этот шаблон для проверки соответствия. В табл. 4.7 перечислены допустимые операторы-метасимволы, а в

Таблица 4.7. Метасимвольные операторы

Оператор-метасимвол	Соответствует
%	Строке, состоящей из любого количества произвольных символов
—	Любому одному символу

табл. 4.8 – некоторые ходовые примеры шаблонов значений;

- проверка на совпадение с шаблоном значений может учитывать или не учитывать регистр букв (см. примечание DBMS в разделе «Фильтрация строк с помощью предложения WHERE» в этой главе);
- вы всегда можете инвертировать любое условие LIKE, поставив перед оператором LIKE оператор NOT (то есть оператор NOT LIKE – это инверсия оператора LIKE);
- вы можете объединять условия LIKE с любым количеством других условий логическими операторами AND и OR;

■ когда некий шаблон значений совсем не содержит метасимволов, оператор LIKE работает как оператор сравнения (=) – см. табл. 4.2, а оператор NOT LIKE – соответственно как оператор сравнения (<>). К примеру, предложение WHERE city LIKE 'New York' эквивалентно предложению WHERE city = 'New York'.

Фильтрация строк по произвольному шаблону значений

Напечатайте:

```
SELECT columns
FROM table
WHERE test_column [NOT] LIKE
      'pattern';
```

Таблица 4.8. Примеры применения метасимволов в шаблонах

Шаблон	Соответствует
'A%'	Соответствует любому строковому значению длиной не меньше одного символа, начинающемуся с заглавной английской (это важно) «А», включая просто одну заглавную букву «А». Пример строковых значений, соответствующих данному шаблону: 'Anonymous', 'AC/DC'
'%s'	Любому строковому значению длиной не меньше одного символа, заканчивающемуся строчной буквой «s», включая просто одну строчную букву «s». Имейте в виду, что строковое значение, состоящее из «s» и следующих за ней пробелов, не отвечает данному шаблону. Вот еще пара строковых значений: 'DMBSes', 'Victoria Falls'
'%in%'	Любому строковому значению длиной не меньше двух символов, содержащему сочетание строчных букв «in». Этому шаблону соответствуют строковые значения 'in', 'inch', 'Pine', 'linchpin', 'lynchpin'
'____'	Любому строковому значению длины, равному 4. Этому шаблону соответствуют 'АБВГ', 'Я ем' и 'Jack'
'Qua__'	Любому строковому значению длиной 5 символов, начинающемуся с Qua.... Этому шаблону соответствуют 'Quack', 'Quaff' и 'Quake'
'_re_'	Любому строковому значению длины, равному 4 символам; его вторым и третьим значениями является пара строчных английских букв «re». Под этот шаблон подходят следующие строковые значения: 'Tree', 'area' и 'fret'
'_re%'	Любому строковому значению длиной не меньше 3 символов; в нем на втором и третьем местах стоят английские строчные буквы «re». Под этот шаблон подходят следующие строковые значения: 'Tree', 'area', 'fret', 'fretful' и 'freeze'
'%re_'	Любому строковому значению длиной не меньше 3 символов; в нем на втором и третьем местах, считая с конца, стоят английские строчные буквы «e» и «r» соответственно. Под этот шаблон подходят следующие строковые значения: 'Tree', 'area', 'fret', 'red' и 'Blood red'

Подразумевается, что:

- идентификатор *columns* будет заменен одним или несколькими, разделенными запятыми, реальными именами каких-нибудь столбцов;
- идентификатор *table* будет заменен именем той таблицы, которая содержит эти столбцы *columns*;
- вы, формируя условие поиска, сделаете следующее:
 - вместо *test_column* подставите имя какого-нибудь одного столбца в таблице *table*, причем этот столбец совсем не должен входить в состав столбцов *columns*;
 - вместо *pattern* подставите шаблон значений, которые СУБД будет сопоставлять со значениями в столбце *test_column*;
- шаблон значений – это строка символов наподобие тех, что приведены в табл. 4.8;
- если нужно отобрать строки, которые не соответствуют выбранному шаблону значений, воспользуйтесь не оператором LIKE, а оператором NOT LIKE (см. листинги и рис. 4.30–4.33).

Одной из особенностей SQL является то, что можно включать в шаблон значений метасимволы не только как таковые (то есть не только как операторы и подстановочные, специальные знаки), но и как обыкновенные символы. Чтобы СУБД перестала воспринимать произвольный метасимвол в качестве оператора и специально го символа, ей надо об этом объявить с помощью другого оператора и специального символа, называемого *переключающим символом*. Посредством переключающего символа вы сможете в нужный момент объяснить СУБД, чтобы она воспринимала

Листинг 4.30. Распечатать из нашей базы данных имена и фамилии авторов, чьи фамилии начинаются на Kel. Результат исполнения запроса см. на рис. 4.30

Листинг	
<pre>SELECT au_fname, au_lname FROM authors WHERE au_lname LIKE "Kel%";</pre>	
au_fname	au_lname
-----	-----
Christian	Kells
	Kellsey

Рис. 4.30. Результат исполнения запроса, представленного в листинге 4.30

Листинг 4.31. Распечатать из нашей базы данных имена и фамилии авторов, в фамилиях которых на местах 3-го и 4-го знаков есть буква «l». Результат исполнения этого запроса см. на рис. 4.31

Листинг	
<pre>SELECT au_fname, au_lname FROM authors WHERE au_lname LIKE '__l1%';</pre>	
au_fname	au_lname
-----	-----
Hallie	Hull
Klee	Hull
Christian	Kells
	Kellsey

Рис. 4.31. Результат исполнения запроса, представленного в листинге 4.31

Листинг 4.32. Распечатать имена и фамилии авторов, учитываемых в нашей базе данных, которые живут в районе залива Сан-Франциско (почтовый индекс этого места начинается с цифр 94...). Результат исполнения запроса см. на рис. 4.32

Листинг				
<pre>SELECT au_fname, au_lname, city, state, zip FROM authors WHERE zip LIKE '94___';</pre>				
au_fname	au_lname	city	state	zip
-----	-----	-----	-----	-----
Hallie	Hull	San Francisco	CA	94123
Klee	Hull	San Francisco	CA	94123
	Kellsey	Palo Alto	CA	94305

Рис. 4.32. Результат исполнения запроса, представленного в листинге 4.32

тот или иной метасимвол, скажем, знак процентов или знак подчеркивания, когда будет определять, соответствует или нет некая строка заданному шаблону значений. Чтобы освободить нужный вам метасимвол от его специального значения, следует в определенном месте тела команды напечатать переключающий символ и сразу после него без пробела – нужный вам метасимвол. А для того, чтобы обозначить этот самый переключающий символ, надо применить ключевое слово ESCAPE. Если, например, в качестве переключающего символа используется восклицательный знак, то сочетание знаков !% в шаблоне значений будет означать буквально %, и более ничего. Такие метасимволы называются *переключенными*. При этом *непереключенные* метасимволы (которые не являются переключенными) сохраняют свое специальное значение. Символ, определенный вами как переключающий, не может быть составным элементом шаблона. Например, если надо организовать поиск подстроки '50% OFF!', придется выбрать в качестве переключающего символа не восклицательный знак, а что-то другое. Наконец, шаблоны значений, использующие переключенные символы, принято тоже называть переключенными.

В табл. 4.9 приведены примеры *переключенных* и *непереключенных шаблонов*, где переключающим символом назначен восклицательный знак.

Включение в шаблон значений знака метасимвола

Напечатайте:

```
SELECT columns
FROM table
WHERE test_column [NOT] LIKE
      'pattern'
      ESCAPE 'escape_char';
```

Предполагается, что:

- за исключением предложения `ESCAPE`, весь синтаксис именно такой, как у команды `SELECT` в подразделе «Фильтрация строк по произвольному шаблону значений» настоящей главы;
- вы сделаете следующее (см. листинг 4.34 и рис. 4.34):
 - замените `escape_char` каким-нибудь одним знаком, который и будет переключающим символом в этом запросе;
 - не будете включать переключающий символ, который пока обозначен как `escape_char`, в шаблон значений.

П

Вместо `test_column` вы вполне можете подставить произвольное выражение.

П

Оператор `NOT` можно ставить и перед оператором `LIKE`, и перед шаблоном `test_column`. Это значит, что оператор `NOT`, который может находиться перед оператором `LIKE`, работает независимо от оператора `NOT`, который может стоять перед шаблоном значений (посмотрите, однако, подраздел «Оператор `NOT`» из предшествующего раздела). Например, предложение `WHERE phone NOT LIKE '212-%'` эквивалентно предложению `WHERE NOT phone LIKE '212-%'`. В этой связи вы вполне могли бы написать следующее нелепое по сути, но синтаксически правильное предложение `WHERE NOT phone NOT LIKE '212-%'`, задача которого – отобрать авторов, чей телефон начинается на 212.

П

Поиск по шаблонам, включающим метасимволы, требует слишком много времени. Это особенно заметно в том случае, когда метасимвол `%` применяется в начале некоего шаблона. Поэтому не рекомендуется применять метасимволы, если есть другие способы поиска.

Таблица 4.9. Переключенные и непереключенные шаблоны значений

Шаблон	Соответствует
'100%'	Непереключенный шаблон. Соответствует строке символов длиной не менее 3 символов, начинающейся со 100...
'100!%'	Переключенный шаблон. Соответствует строго подстроке '100%'
'_op'	Непереключенный шаблон. Соответствует строке длиной строго 3 символа, на втором и третьем местах которой стоят буквы латинского алфавита «o» и «p» соответственно
'!_op'	Переключенный шаблон. Соответствует строго подстроке '_op'

Листинг 4.33. Перечислить из нашей базы данных авторов, живущих за пределами территориальной области, которая соответствует междугородним телефонным кодам 212, 415 и 303. Обратите внимание на три разных способа исключить ненужные телефонные номера с помощью шаблона. Советуем воспользоваться первым способом, потому что проверка соответствий однознаковому (`_`) шаблону выполняется намного быстрее, чем многознаковому (`%`). Результат исполнения запроса см. на рис. 4.33

Листинг

```
SELECT au_fname, au_lname, phone
FROM authors
WHERE phone NOT LIKE '212-____-____'
AND phone NOT LIKE '415-____-____'
AND phone NOT LIKE '303-____-____';
```

au_fname	au_lname	phone
-----	-----	-----
Sarah	Buchman	718-496-7223
	Kellsey	650-836-7128
Paddy	O' Furniture	941-925-0752

Рис. 4.33. Результат исполнения запроса, представленного в листинге 4.33

Листинг 4.34. Перечислить из нашей типовой базы названия книг, которые содержат знак процентов (именно как знак %, а не как оператор-метасимвол). Заметьте, что СУБД воспринимает только тот знак процентов в шаблоне значений, который следует за переключающим символом (в данном случае за восклицательным знаком), а все остальные знаки процентов в том же самом шаблоне (который теперь стал переключенным) сохраняют функциональность операторов-метасимволов. Результат исполнения запроса см. на рис. 4.34

Листинг

```
SELECT title_name
FROM titles
WHERE title_name LIKE '%!%%' ESCAPE
'!';
```

title_name

Рис. 4.34. Результат исполнения запроса, представленного в листинге 4.34. Мы видим пустое множество строк! Все дело в том, что в названиях книг, занесенных в нашу типовую базу данных, нет знака процентов

Таблица 4.10. Примеры шаблонов [] и [^]

Шаблон	Соответствует
'[a-c]'	'bat', 'cat', но не 'fat'
'[bcf]at'	'bat', 'cat', 'fat', но не 'eat'
'[^c]at'	'bat', 'fat', но не 'cat'
'se[^n]%'	Всем строкам длиной не менее двух символов; они начинаются на «se», но не заканчиваются на «n»



СУБД Microsoft Access не поддерживает предложение ESCAPE. Если же вы работаете в этой среде и намерены переключить какой-нибудь метасимвол шаблона, вам следует заключить этот метасимвол в квадратные скобки. Например, чтобы запустить в Access запрос, представленный в листинге 4.34, нужно переписать предложение WHERE следующим образом: WHERE title_name LIKE '%[%]%'.

Некоторые коммерческие СУБД, в том числе Microsoft SQL Server, поддерживают так называемые *регулярные выражения* стандарта POSIX, когда метасимвол в виде квадратных скобок с указанным в них диапазоном значений соответствует любому символу из этого диапазона, а метасимвол в виде квадратных скобок, в которых сначала стоит символ крышки, а потом некий диапазон, соответствует любому символу не из диапазона (в математике говорят «из дополнения к диапазону»). В табл. 4.10 приведены примеры таких шаблонов. Но имейте в виду, что синтаксис регулярных выражений зависит от конкретной СУБД, поэтому, если вы хотите их применять, организуйте сначала поиск в документации по своей СУБД с использованием ключа WHERE или LIKE.

Отдельные СУБД допускают применение оператора LIKE для поиска числовых значений, а также значений даты и времени.

Сравнение с диапазоном с помощью оператора BETWEEN

Оператор BETWEEN применяют для того, чтобы определить, находится или нет значение некоего столбца произвольной строки абстрактной таблицы в пределах выбранного диапазона. Перечислим основные характеристики оператора BETWEEN:

- работает со строками символов, числами и значениями даты и времени;
- его диапазон состоит из двух частей (символически обозначаемых как *low_value* и *high_value*), разделенных оператором AND:
 - меньшее значение *low_value*, определяющее нижнюю границу диапазона;
 - большее значение *high_value*, определяющее верхнюю границу диапазона, и такое, что оно не меньше нижней границы;
- это очень удобное, сжатое и экономичное предложение, но вы можете исключить его из предложения WHERE, построив с помощью оператора AND эквивалентное, но более длинное предложение WHERE, например: предложение WHERE *test_column* BETWEEN *low_value* AND *high_value* эквивалентно предложению WHERE (*test_column* >= *low_value*) AND (*test_column* <= *high_value*);
- обозначает замкнутый диапазон, верхняя и нижняя границы которого включаются в него. Но вы можете определить и открытый диапазон, исключаящий границы. Для этого, правда, вам надо отказаться от оператора BETWEEN и использовать в предложении WHERE операторы сравнения (<) и (>), например: WHERE (*test_column* > *low_value*) AND (*test_column* < *high_value*);

- имейте в виду, что сравнения строк могут быть (или не быть) чувствительными к регистру букв. Это зависит от того, с какой СУБД вы работаете (см. примечание DBMS в разделе «Фильтрация строк с помощью предложения WHERE» этой главы);
- оператор BETWEEN можно инвертировать оператором NOT в оператор NOT BETWEEN;
- условия, выраженные предложениями BETWEEN, можно объединять с другими условиями с использованием операторов AND и OR.

Фильтрация строк по любому диапазону

Напечатайте:

```
SELECT columns
FROM table
WHERE test_column [NOT] BETWEEN
      low_value AND high_value;
```

Подразумевается, что:

- идентификатор *columns* будет заменен одним или несколькими, разделенными запятыми, реальными именами каких-нибудь столбцов;
- идентификатор *table* будет заменен именем той таблицы, которая содержит столбцы *columns*;
- формируя условие поиска, вы сделаете следующее:
 - вместо *test_column* подставите имя какого-нибудь одного столбца в таблице *table*, причем он совсем не должен входить в состав столбцов *columns*;
 - вместо *low_value* и *high_value* подставите границы диапазона значений, которые СУБД будет сопоставлять со значениями в столбце *test_column*, при этом большее значение *high_value*, определяющее верхнюю гра-

Листинг 4.35. Перечислить из нашей базы всех авторов, у которых почтовый индекс не попадает в интервал от 20 000 до 89 999. Результат исполнения запроса см. на рис. 4.35

Листинг		
<pre>SELECT au_fname, au_lname, zip FROM authors WHERE zip NOT BETWEEN '20000' AND '89999';</pre>		
au_fname	au_lname	zip
-----	-----	-----
Sarah	Buchman	10468
Hallie	Hull	94123
Klee	Hull	94123
Christian	Kells	10014
	Kellsey	94305

Рис. 4.35. Результат исполнения запроса, представленного в листинге 4.35

Листинг 4.36. Перечислить названия книг из нашей базы, цена которых составляет от 10 до 19,95 долларов включительно. Результат исполнения этого запроса см. на рис. 4.36

Листинг	
<pre>SELECT title_id, price FROM titles WHERE price BETWEEN 10 AND 19.95;</pre>	
title_id	price
-----	-----
T02	19.95
T04	12.99
T06	19.95
T08	10.00
T09	13.95
T12	12.99

Рис. 4.36. Результат исполнения запроса, представленного в листинге 4.36

ницу диапазона, должно быть не меньше *low_value*, определяющего нижнюю границу диапазона;

- оба граничных значения, определяющие диапазон, должны относиться к одному и тому же типу данных, который, в свою очередь, должен совпадать или быть совместимым с типом данных, назначенным столбцу *test_column*;
- если нужно отобразить строки, которые не соответствуют выбранному диапазону значений, вы укажете не оператор BETWEEN, а оператор NOT BETWEEN (см. листинги и рис. 4.35–4.37).

П Вместо *test_column* вы вполне можете подставить произвольное выражение.

П Оператор NOT можно ставить и перед оператором BETWEEN, и перед столбцом *test_column*. Это значит, что оператор NOT, стоящий перед оператором BETWEEN, работает независимо от оператора NOT, который находится перед столбцом *test_column* (посмотрите, тем не менее, подраздел «Оператор NOT» и советы в разделе «Сравнение по шаблону оператором LIKE» в данной главе).

П Задача определения любого символьного диапазона требует немалых интеллектуальных усилий. Допустим, что нам требуется организовать поиск авторов, чьи фамилии начинаются с буквы «F». Предложение WHERE last_name BETWEEN 'F' AND 'G' не сработает. Когда мы так говорим, то имеем в виду, что оно, будучи вполне корректным синтаксически, даст неверный результат. В данном же случае в результат вошел бы автор, в фамилии которого есть буква «G» (подчеркнем, есть буква «G», а не начинается с буквы «G», ведь интервал – замкнутый), в дополнение ко всем тем авторам, фамилии которых начинаются с буквы «F». Правильный способ указать интервал для решения подобных задач – отметить верхнюю границу двумя буквами (почти всегда), первой и последней, например: WHERE last_name BETWEEN 'F' AND 'Fz'.

П

В листинге 4.38 показано, как переписать запрос, представленный в листинге 4.36, для открытого диапазона, исключающего значения 10 и 19,95 доллара. Результат работы откорректированного запроса см. на рис. 4.38.



В среде СУБД PostgreSQL из примеров в листингах 4.36 и 4.38 вам следует преобразовать числа с плавающей десятичной точкой в тип `DECIMAL` (см. раздел «Преобразование типов данных с помощью функции `CAST()`» в главе 5). Преобразуйте литералы с плавающей точкой следующим образом: `CAST(19.95 AS DECIMAL)`.

В среде СУБД Microsoft Access, печатая литералы даты и времени в предложении `WHERE`, вам следует опускать ключевое слово `DATE`, а также заключать каждый такой литерал не в кавычки, а в знаки решетки (`#`). Чтобы, например, выполнить запрос, представленный в листинге 4.37, замените даты в предложении `WHERE` следующими `#2000-01-01#` и `#2000-12-31#`.

В среде СУБД Microsoft SQL Server, печатая литералы даты и времени в предложении `WHERE`, вам следует опускать ключевое слово `DATE`. Чтобы, например, выполнить запрос, представленный в листинге 4.37, замените даты в предложении `WHERE` следующими `'2000-01-01'` и `'2000-12-31'`.

Имейте в виду, что некоторые СУБД допускают, чтобы значение, подставляемое вместо *low_value*, превосходило (не предшествовало) значение, подставляемое вместо *high_value*. Чтобы разобраться в этом вопросе как следует, организуйте поиск в документации по своей СУБД с использованием ключей поиска `WHERE` или `BETWEEN`.

Листинг 4.37. Перечислить названия книг, которые вышли из печати в 2000 году. Результат исполнения запроса см. на рис. 4.37

```

Листинг
SELECT title_id, pubdate
FROM titles
WHERE pubdate BETWEEN DATE '2000-01-01'
AND DATE '2000-12-31';

```

title_id	pubdate
-----	-----
T01	2000-08-01
T03	2000-09-01
T06	2000-07-31
T11	2000-11-30
T12	2000-08-31

Рис. 4.37. Результат исполнения запроса, представленного в листинге 4.37

Листинг 4.38. Перечислить названия книг, цена которых составляет от 10 до 19,95 доллара. Результат исполнения запроса см. на рис. 4.38

```

Листинг
SELECT title_id, price
FROM titles
WHERE (price > 10)
AND (price < 19.95);

```

title_id	price
-----	-----
T04	12.99
T09	13.95
T12	12.99

Рис. 4.38. Результат исполнения запроса, представленного в листинге 4.38

Фильтрация с помощью оператора IN

Чтобы определить, соответствует ли определенное значение какому-либо значению из произвольного списка, применяют оператор IN. Перечислим основные его характеристики:

- работает со строками символов, числами и значениями даты и времени;
- представляет собой одно или несколько неупорядоченных значений, разделенных запятыми и заключенных в круглые скобки;
- это очень удобное, сжатое и экономичное предложение, но вы можете исключить его из предложения WHERE, построив с помощью оператора OR эквивалентное, но более длинное предложение WHERE, например так: предложение WHERE *test_column* IN (*value1*, *value2*, *value3*) эквивалентно предложению WHERE (*test_column* = *value1*) OR (*test_column* = *value2*) OR (*test_column* = *value3*);
- имейте в виду, что сравнения строк могут быть (или не быть) чувствительными к регистру букв (это зависит от того, с какой СУБД вы работаете; см. примечание DBMS в разделе «Фильтрация строк с помощью предложения WHERE» в данной главе);
- его можно преобразовать оператором NOT в оператор NOT IN;
- условия, выраженные предложениями IN, можно объединять с другими условиями операторами AND и OR.

Фильтрация строк по произвольному списку

Напечатайте:

```
SELECT columns
FROM table
WHERE test_column [NOT] IN
      (value1, value2,...);
```

Подразумевается, что:

- идентификатор *columns* будет заменен одним или несколькими, разделенными запятыми, реальными именами каких-нибудь столбцов;
- идентификатор *table* будет заменен именем таблицы, которая содержит столбцы *columns*;
- вы, формируя условие поиска, выполните следующее:
 - вместо *test_column* подставьте имя какого-нибудь столбца в таблице *table*, причем этот столбец совсем не должен входить в состав столбцов *columns*;
 - вместо *value1*, *value2*, ... подставьте одно или несколько разделенных запятыми значений, которые СУБД будет сопоставлять со значениями в столбце с именем *test_column*, при этом значения списка *value1*, *value2*, ... могут быть перечислены в любом порядке и их тип должен или совпадать, или быть совместимым с типом столбца *test_column*;
- если нужно отобразить строки, которые не соответствуют выбранному списку значений, вы укажете не оператор IN, а оператор NOT IN (см. листинги и рис. 4.39–4.41).

132 Выбор данных из произвольной таблицы

П

Вместо *test_column* вы вполне можете подставить произвольное выражение.

П

Оператор NOT можно ставить и перед оператором IN, и перед *test_column*. Это значит, что оператор NOT, стоящий перед оператором IN, работает независимо от оператора NOT, расположенного перед столбцом *test_column*, по которому проводится фильтрация (см. подраздел «Оператор NOT» и советы в разделе «Сравнение по шаблону оператором LIKE» в настоящей главе).

П

Если список, по которому вы хотите фильтровать строки, очень обширный, вы можете организовать фильтрацию именно оператором IN, а не с помощью многочисленных сравнений, соединенных операторами OR, чтобы ваш код был более читабельным. Кстати сказать, один оператор IN обычно срабатывает гораздо быстрее, чем несколько операторов OR.

П

Порядок вычисления сложных условий гораздо проще понять, если вместо многочисленных операторов OR применять один оператор IN (см. раздел «Комбинирование условий с помощью операторов AND, OR и NOT» в данной главе).

П

Оператор IN можно применять еще для того, чтобы проверить, соответствует ли произвольное заданное значение любому значению в некоем подзапросе.

П

Оператор NOT IN эквивалентен объединению операторами AND нескольких условий-неравенств. Например, следующий запрос эквивалентен запросу из листинга 4.39:

```
SELECT au_fname, au_lname, state
FROM authors
WHERE state <> 'NY'
      AND state <> 'NJ'
      AND state <> 'CA';
```

Листинг 4.39. Перечислить авторов, учитываемых в нашей базе, которые живут не в штате Нью-Йорк, не в штате Нью-Джерси, не в штате Калифорния. Результат исполнения этого запроса см. на рис. 4.39

```
SELECT au_fname, au_lname, state
FROM authors
WHERE state NOT IN ('NY', 'NJ', 'CA');
```

au_fname	au_lname	state
-----	-----	-----
Wendy	Heydemark	CO
Paddy	O' Furniture	FL

Рис. 4.39. Результат исполнения запроса, представленного в листинге 4.39

Листинг 4.40. Перечислить названия книг из нашей типовой базы, за которые аванс, выплаченный автору каждой книги, составил или 0 долларов, или 1000 долларов, или 5000 долларов. Результат исполнения этого запроса см. на рис. 4.40

```
SELECT title_id, advance
FROM royalties
WHERE advance IN
      (0.00, 1000.00, 5000.00);
```

title_id	advance
-----	-----
T02	1000.00
T08	0.00
T09	0.00

Рис. 4.40. Результат исполнения запроса, представленного в листинге 4.40

Листинг 4.41. Перечислить названия (точнее, однозначные идентификаторы и даты публикации) книг из нашей базы, опубликованных 1 января 2000, 2001 и 2002 годов. Результат исполнения этого запроса см. на рис. 4.41

Листинг	
<pre>SELECT title_id, pubdate FROM titles WHERE pubdate IN (DATE '2000-01-01', DATE '2001-01-01', DATE '2002-01-01');</pre>	
title_id	pubdate
-----	-----
T05	2001-01-01

Рис. 4.41. Результат исполнения запроса, представленного в листинге 4.41



В среде СУБД PostgreSQL при работе с запросом из листинга 4.40 следует преобразовать числа с плавающей десятичной точкой в тип данных DECIMAL (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5). Модифицируйте литералы с плавающей точкой следующим образом:

```
WHERE advance IN
    (CAST(0.00 AS DECIMAL),
     CAST(1000.00 AS DECIMAL),
     CAST(5000.00 AS DECIMAL))
```

В среде СУБД Microsoft Access, печатая литералы даты и времени в предложении WHERE, следует опускать ключевое слово DATE и окружать каждый такой литерал не кавычками, а знаками решетки (#). Чтобы, например, запустить запрос, представленный в листинге 4.41, измените предложение WHERE следующим образом:

```
WHERE pubdate IN
    (#2000-01-01#,
     #2001-01-01#,
     #2002-01-01#)
```

В среде СУБД Microsoft SQL Server, печатая литералы даты и времени в предложении WHERE, следует опускать ключевое слово DATE. Чтобы, например, запустить запрос, представленный в листинге 4.41, измените предложение WHERE следующим образом:

```
WHERE pubdate IN
    ('2000-01-01',
     '2001-01-01',
     '2002-01-01')
```

Проверка на значение null с помощью оператора IS NULL

В разделе «Значение null» главы 3 мы говорили о том, что значения null замещают отсутствующие или неизвестные значения. Это создает определенную проблему с обнаружением этих значений, потому что ни опция LIKE, ни опция BETWEEN, ни опция IN, ни другие опции предложения WHERE не могут обнаружить значений null, поскольку неизвестное значение не может удовлетворить каким бы то ни было известным условиям. Между тем находить эти значения иногда необходимо. Например, обратите внимание, что строка P03 таблицы publishers (издатели) как раз имеет значение null в столбце state (штат) просто потому, что в Германии нет штатов (см. листинги и рис. 4.43 и 4.44). Но для того, чтобы «отловить» это значение null, мы не смогли бы применить так называемые дополняющие сравнения (то есть инвертированные), так как искомое значение null является неопределенным.

Во избежание подобных проблем используйте опцию IS NULL, которая должна проверить, является ли произвольное заданное значение значением null.

Перечислим основные характеристики оператора IS NULL:

- работает со столбцами любых типов данных;
- его можно инвертировать в оператор IS NOT NULL;
- можно объединять условия IS NULL с другими условиями операторами AND и OR.

Листинг 4.42. Перечислить места нахождения всех издателей, учитываемых в нашей типовой базе. Результат исполнения этого запроса см. на рис. 4.42

```

Листинг
SELECT pub_id, city, state, country
FROM publishers;
```

pub_id	city	state	country
-----	-----	----	-----
P01	New York	NY	USA
P02	San Francisco	CA	USA
P03	Hamburg	NULL	Germany
P04	Berkeley	CA	USA

Рис. 4.42. Результат исполнения запроса, представленного в листинге 4.42

Листинг 4.43. Перечислить идентификаторы и адреса издателей, учитываемых в нашей типовой базе данных, которые находятся в штате Калифорния. Результат исполнения этого запроса см. на рис. 4.43

```

Листинг
SELECT pub_id, city, state, country
FROM publishers
WHERE state = 'CA';
```

pub_id	city	state	country
-----	-----	----	-----
P02	San Francisco	CA	USA
P04	Berkeley	CA	USA

Рис. 4.43. Результат исполнения запроса, представленного в листинге 4.43

Листинг 4.44. Этот запрос является неудачной попыткой (правильное решение той же задачи показано в листинге 4.45) выбрать издателей, учитываемых в нашей типовой базе данных, которые расположены вне штата Калифорния. Результат исполнения этого запроса см. на рис. 4.44

```

Листинг

SELECT pub_id, city, state, country
FROM publishers
WHERE state <> 'CA';
    
```

pub_id	city	state	country
-----	-----	-----	-----
P01	New York	NY	USA

Рис. 4.44. Результат исполнения запроса, представленного в листинге 4.44. Мы видим, что, кроме тех издателей, которые находятся в штате Калифорния, данный результат игнорирует строку P03, хотя соответствующий издатель расположен не в Калифорнии, а в Германии. Строго говоря, условия `state = 'CA'` и `state <> 'CA'` не являются взаимодополняющими, потому что значения null не соответствуют ни одному значению, независимо от типа. Следовательно, их нельзя обнаружить ни одним из типов условий, которые мы рассматривали

Выбор строк, содержащих значения null

Напечатайте:

```

SELECT columns
FROM table
WHERE test_column IS [NOT] NULL;
    
```

Подразумевается, что:

- идентификатор *columns* будет заменен одним или несколькими, разделенными запятыми, реальными именами каких-нибудь столбцов;
- идентификатор *table* будет заменен именем таблицы, которая содержит столбцы *columns*;
- вы, формируя условие поиска, вместо *test_column* подставите имя какого-нибудь одного столбца в таблице *table*, причем он совсем не должен входить в состав столбцов *columns*, но именно среди его значений вы будете искать значения null;
- если вы захотите отобразить строки, которые содержат в столбце *test_column* значения, не являющиеся значениями null, вы укажете не оператор IS NULL, а оператор IS NOT NULL (см. листинги и рис. 4.45–4.46).

П Вместо *test_column* вы вполне можете подставить произвольное выражение.

П Оператор NOT можно ставить и перед ключевым словом NULL, и перед *test_column*. Это значит, что оператор NOT, стоящий перед ключевым словом NULL, работает независимо от оператора NOT, расположенного перед столбцом *test_column*, по которому проводится фильтрование (см. советы в разделе «Сравнение по шаблону оператором LIKE» в данной главе).

П

Из результата произвольного запроса вам удастся исключить предложением `WHERE` некую строку, содержащую значение `null`, только при условии, что столбец этой строки, который содержит значение `null`, окажется столбцом, объявленным в предложении `WHERE` как `test_column`. Но если значение `null` находится в другом столбце, строка окажется в результате запроса. К примеру, следующий запрос выберет все строки таблицы `publishers` (см. рис. 4.42), так как значение `null`, которое находится в столбце `state`, ни с чем не сравнивается, поскольку задачу столбца `test_column` здесь выполняет столбец `state`:

```
SELECT pub_id, city, state, country
FROM publishers
WHERE country <> 'Canada';
```

П

Повторим еще раз, что никакая пустая строка (") не равна значению `null`. Например, в нашей типовой базе данных в таблице `authors` есть столбец `au_fname` (имя), который, в свою очередь, содержит пустую строку (имеется в виду строка как тип данных) в строке (имеется в виду строка как структурный элемент таблицы), соответствующей автору A06 с фамилией Kellsey. Так вот, условием, применяемым в предложении `WHERE` для того, чтобы отыскать автора, у которого вместо имени пустая строка, является `WHERE au_fname = ''`, а не `WHERE au_fname IS NULL`. Тем не менее обязательно ознакомьтесь с примечаниями **DBMS**, относящимися к Oracle, в разделе «Значение `null`» главы 3.

П

Чтобы предотвратить появление значений `null` в произвольном столбце таблицы, обратитесь к разделу «Запрет значения `null` с помощью ограничения `NOT NULL`» в главе 10.

Листинг 4.45. Перечислить всех издателей, которые находятся вне штата Калифорния. Результат исполнения этого запроса см. на рис. 4.45

```
Листинг
SELECT pub_id, city, state, country
FROM publishers
WHERE state <> 'CA'
OR state IS NULL;
```

pub_id	city	state	country
-----	-----	-----	-----
P01	New York	NY	USA
P03	Hamburg	NULL	Germany

Рис. 4.45. Результат исполнения запроса, представленного в листинге 4.45. Теперь мы видим, что издатель P03 есть в результате

Листинг 4.46. Выбрать книги-биографии, даты публикации (прошлые или будущие) которых известны. Результат исполнения этого запроса см. на рис. 4.46

```
Листинг
SELECT title_id, type, pubdate
FROM titles
WHERE type = 'biography'
AND pubdate IS NOT NULL;
```

title_id	type	pubdate
-----	-----	-----
T06	biography	2000-07-31
T07	biography	1999-10-01
T12	biography	2000-08-31

Рис. 4.46. Результат исполнения запроса, представленного в листинге 4.46. Если бы не условие `IS NOT NULL`, этот результат содержал бы и книгу с названием, определяемым первичным ключом T10

ОПЕРАТОРЫ И ФУНКЦИИ

5

Операторы и функции нужны для того, чтобы вычислять различные выражения по данным, которые извлекаются из столбцов или/и которые определяются вашей СУБД. С помощью операторов и функций вы можете выполнять:

- арифметические операции (например, всем сотрудникам фирмы сократить зарплату на 10%);
- строковые операции (например, объединить личные данные и печатать почтовые адреса на почтовых конвертах);
- операции с данными даты и времени (например, вычислить интервалы между парами дат);
- разнообразные системные операции (например, выяснить у СУБД, сколько сейчас времени).

Произвольный *оператор* – это некий символ или какое-нибудь ключевое слово, обозначающее операцию, на вход которой подается один или несколько элементов, а эти элементы, называемые *операндами*, представляют собой выражения SQL.

В разделе «Синтаксис SQL» главы 3 мы уже говорили о том, что любое выражение – это синтаксически не запрещенная комбинация символов и лексем, которая на любом наборе значений, подставляемых вместо этих символов и лексем, принимает одно-единственное значение определенного типа или значение null. К примеру, в арифметической операции умножения `price*2` оператором является «*», а «`price`» и «`2`» – это операнды.

Итак, любая *функция* – это встроенная поименованная процедура, выполняющая специальную задачу. Большинство функций содержат в скобках свои *аргументы*, которыми, по сути, являются значения, передаваемые на вход этой функции для того, чтобы она использовала их для выполнения своей задачи. Аргументами функций могут быть имена столбцов, константы, вложенные функции или более сложные выражения. Например, в функции `UPPER(au_lname)` ключевое слово `UPPER` – это собственно функция, а `au_lname` – ее аргумент.

Создание производных столбцов

Операторы и функции нужны для того, чтобы с их помощью создавать так называемые производные столбцы. Любой *производный столбец* – это результат вычислительного процесса, получаемый в ходе исполнения какого-то запроса SELECT, содержащего в качестве одного из своих предложений такое, которое отличается от простой ссылки на один какой-нибудь столбец. Производные столбцы служат исключительно для визуализации и никогда не становятся постоянными столбцами таблиц.

Очень часто значения производного столбца вычисляются по значениям существующих столбцов базы данных, но вы в принципе можете создать любой производный столбец с помощью какого-нибудь выражения с константами (например, строки символов, числа или даты) или какого-нибудь системного значения (скажем, системного времени). В частности, запрос, представленный в листинге 5.1, реализует одно тривиальное арифметическое вычисление. И этому запросу не требуется вообще никакого предложения FROM, так как он не использует никаких данных (результат исполнения этого запроса см. на рис. 5.1). Здесь неплохо было бы освежить в памяти материал из раздела «Таблицы, столбцы и строки» главы 2, где говорилось о свойстве замкнутости реляционной модели, которое, в свою очередь, состоит в том, что результатом любой операции должна быть обязательно какая-нибудь таблица. На первый взгляд, наблюдается противоречие с этим свойством. Но на самом деле никакого противоречия нет, просто результатом является простейшая таблица, состоящая из одного столбца и одной строки (то есть

Листинг 5.1. Здесь представлено одно выражение с константами в предложении SELECT. Данный запрос не требует никакого предложения FROM, поскольку никаких данных из таблицы не выбирается. Результат исполнения этого запроса показан на рис. 5.1

```

Листинг
SELECT 2 + 3;

2 + 3
-----
5

```

Рис. 5.1. Результат исполнения запроса, представленного в листинге 5.1. Мы видим, что этим результатом является таблица, состоящая из одной строки и одного столбца

Листинг 5.2. Выбрать один столбец и одно выражение с константами. Результат исполнения этого запроса см. на рис. 5.2

```

Листинг
SELECT au_id, 2 + 3
FROM authors;

au_id      2 + 3
-----
A01        5
A02        5
A03        5
A04        5
A05        5
A06        5
A07        5

```

Рис. 5.2. Результат исполнения запроса, представленного в листинге 5.2. Мы видим, что одна и та же константа повторяется в каждой из выбранных строк

Листинг 5.3. Перечислить идентификаторы книг вместе с ценами, сниженными на 10%. Имейте в виду, что, если из предложения `SELECT` удалить оба предложения `AS`, соответствующие производные столбцы будут названы по умолчанию. Результат исполнения этого запроса см. на рис. 5.3

```

SELECT title_id,
       price,
       0.10 AS "Discount",
       price * (1 - 0.10) AS "New price"
FROM titles;

```

title_id	price	Discount	New price
-----	-----	-----	-----
T01	21.99	0.10	19.79
T02	19.95	0.10	17.95
T03	39.95	0.10	35.96
T04	12.99	0.10	11.69
T05	6.95	0.10	6.25
T06	19.95	0.10	17.95
T07	23.95	0.10	21.56
T08	10.00	0.10	9.00
T09	13.95	0.10	12.56
T10	NULL	0.10	NULL
T11	7.99	0.10	7.19
T12	12.99	0.10	11.69
T13	29.99	0.10	26.99

Рис. 5.3. Результат исполнения запроса, представленного в листинге 5.3

таблица 1.1) и содержащая одно значение, равное в данном случае числу 5. Если бы мы выбрали не только константу, но и какой-нибудь столбец, результатом служила бы таблица из двух столбцов, первым из которых был бы выбираемый реальный столбец, а вторым – та самая константа, появляющаяся в каждой строке результата (см. листинг 5.2 и рис. 5.2).

Если вы не присвоите производному столбцу имя с помощью предложения `AS`, это сделает ваша СУБД (см. листинг 5.3 и рис. 5.3, а также раздел «Создание псевдонимов столбцов с помощью предложения `AS`» в главе 4). Несмотря на возможность присваивать названия производным столбцам по умолчанию, мы постараемся давать им имена явным образом.



В среде Oracle предложение `FROM` является обязательной составляющей команды `SELECT`. Поэтому, если вы хотите выбрать этой командой какое-нибудь выражение с константами, вам придется использовать предложение `FROM`, но с псевдотаблицей `DUAL`, которую ваша СУБД автоматически создает как раз на такой случай (имеет смысл организовать поиск в документации по вашей СУБД Oracle с использованием ключа `DUAL`). Чтобы в среде Oracle выполнить запрос из листинга 5.1, нужно добавить в него предложение `FROM`, которое якобы выбирает нужную константу из псевдотаблицы `DUAL`. Вот так:

```

SELECT 2 + 3
FROM DUAL;

```

В среде СУБД PostgreSQL, выполняя запрос из листинга 5.3, следует преобразовывать числа с плавающей десятичной точкой в тип данных `DECIMAL` (см. раздел «Преобразование типов данных с помощью функции `CAST()`» в этой главе). Измените формулу в производном столбце, содержащем значение с плавающей точкой, следующим образом:

```
Price * CAST((1 - 0.10) AS DECIMAL)
```

Арифметические операции

Произвольный *унарный (одноместный) арифметический оператор* – это такой оператор, который выполняет какую-либо математическую операцию, но над единственным числовым операндом. Простейшим примером унарного арифметического оператора является оператор отрицания (обозначаемый знаком «минус»), меняющий знак своего операнда. Кстати сказать, унарный оператор тождества (обозначаемый знаком «плюс») всегда оставляет свой операнд неизменным.

Произвольный *бинарный (двухместный) арифметический оператор* – это такой оператор, который выполняет какую-нибудь математическую операцию над двумя числовыми операндами. Такими операторами, конечно, являются обычные математические операторы сложения (+), вычитания (–), умножения (*) и деления (/). Более наглядно все арифметические операторы SQL представлены в табл. 5.1, где под *expr* понимается произвольное числовое выражение.

Изменение знака произвольного числа

Напечатайте `-expr`.

Предполагается, что вместо *expr* вы подставите какое-нибудь числовое выражение (см. листинг 5.4 и рис. 5.4).

Сложение, вычитание, умножение или деление

Напечатайте:

- `expr1 + expr2` – если хотите сложить два числовых выражения;
- `expr1 - expr2` – если хотите вычесть из одного числового выражения другое числовое выражение;

Таблица 5.1. Арифметические операторы

Оператор	Операция
<code>-expr</code>	Инвертирует знак числового выражения
<code>+expr</code>	Оставляет знак числового выражения без изменения
<code>expr1 + expr2</code>	Суммирует два числовых выражения
<code>expr1 - expr2</code>	Вычитает из одного числового выражения другое числовое выражение
<code>expr1 * expr2</code>	Перемножает два числовых выражения
<code>expr1 / expr2</code>	Делит одно числовое выражение на другое числовое выражение

Листинг 5.4. Изменить знак числа с помощью оператора отрицания. Результат исполнения этого запроса см. на рис. 5.4

Листинг

```
SELECT title_id,
       -advance AS "Advance"
FROM royalties;
```

title_id	Advance
-----	-----
T01	-10000.00
T02	-1000.00
T03	-15000.00
T04	-20000.00
T05	-100000.00
T06	-20000.00
T07	-1000000.00
T08	0.00
T09	0.00
T10	NULL
T11	-100000.00
T12	-50000.00
T13	-20000.00

Рис. 5.4. Результат исполнения запроса, представленного в листинге 5.4. Обратите внимание, что у нуля знака нет (он не может быть положительным или отрицательным)

Листинг 5.5. Перечислить идентификаторы книг, которые являются биографиями, в порядке убывания дохода от продажи (доход от продажи есть число проданных копий, умноженное на цену одной копии). Результат исполнения этого запроса см. на рис. 5.5

```

SELECT title_id,
       price * sales AS "Revenue"
FROM titles
WHERE type = 'biography'
ORDER BY price * sales DESC;
```

title_id	Revenue
T07	35929790.00
T12	1299012.99
T06	225834.00
T10	NULL

Рис. 5.5. Результат исполнения запроса, представленного в листинге 5.5

- $expr1 * expr2$ – если хотите перемножить два числовых выражения;
- $expr1 / expr2$ – если хотите разделить одно числовое выражение на другое числовое выражение.

Предполагается, что вместо $expr1$, $expr2$ вы подставите какие-нибудь числовые выражения (см. листинг 5.5 и рис. 5.5).

П Результат любой арифметической операции равен null, если в ней присутствует значение null.

П Если вы хотите объединить несколько операторов в одно выражение, вам могут понадобиться круглые скобки (см. раздел «Определение последовательности вычислений» в этой главе).

П Унарные операторы часто называют *одноместными* операторами, а бинарные операторы – *двухместными*.



СУБД предоставляют в распоряжение программиста не только арифметические операторы, но и функции, и дополнительные арифметические операторы. Они включают статистические, финансовые, некоторые так называемые научные, тригонометрические функции и функции преобразования (организуйте поиск в документации по своей СУБД с использованием ключей «оператор» и «функция»).

Если вы включите в одно арифметическое выражение операнды разных числовых типов данных, СУБД преобразует (или, как говорят, *приведет*) все операнды к типу данных самого сложного из них и выдаст результат всего выражения как элемент данных именно этого типа. Этот процесс называется *повышением*. Например, если вы сложите числа INTEGER (целые числа) и FLOAT (рациональные числа с плавающей точкой), СУБД преобразует целые числа в рациональные с плавающей точкой (приведет целые числа к рациональным с плавающей точкой) и выдаст сумму как рациональное число с плавающей точкой. Но

в некоторых случаях вам понадобится преобразовывать один тип данных в другой тип данных явным образом, то есть не полагаясь на СУБД. Этот вопрос обсуждается в разделе «Преобразование типов данных с помощью функции CAST()» данной главы.

При делении целого на целое (то есть тип INTEGER на тип INTEGER) следует быть предельно осторожным. В зависимости от того, с какой именно СУБД вы работаете, дробная часть результата такого деления может быть не усечена совсем, усечена частично или усечена полностью (то есть отброшена). Например, вы могли бы предполагать, что два производных столбца, которые порождаются запросом из листинга 5.6, содержат одни и те же результаты, поскольку в обоих случаях значения столбца pages (тип INTEGER) делится на одно и то же число. Однако в первом случае делителем является 10 (тип INTEGER), а во втором – 10.00 (тип FLOAT). Так вот, СУБД Microsoft Access, Oracle и MySQL выдадут результат, которого вы ожидаете (см. рис. 5.6а), а СУБД Microsoft SQL Server и PostgreSQL усекут дробную часть частного (см. рис. 5.6б).

Листинг 5.6. В этом запросе первый из двух производных столбцов получается делением значений столбца pages (то есть числа страниц в книге) на целое число 10 (тип данных INTEGER). Второй же производный столбец получен делением значений того же самого столбца, но на рациональное число с плавающей десятичной точкой 10.00 (тип данных FLOAT). Вы могли бы ожидать, что значения в обоих производных столбцах будут совпадать. На самом деле все зависит от того, с какой конкретно СУБД вы работаете. Разные результаты исполнения этого запроса, зависящие от СУБД, см. на рис. 5.6а и 5.6б

Листинг

```
SELECT      title_id,
            pages,
            pages/10      AS "pages/10",
            pages/10.0    AS "pages/10.0"
FROM titles;
```

title_id	pages	pages/10	pages/10.0
-----	-----	-----	-----
T01	107	10.7	10.7
T02	14	1.4	1.4
T03	1226	122.6	122.6
T04	510	51.0	51.0
T05	201	20.1	20.1
T06	473	47.3	47.3
T07	333	33.3	33.3
T08	86	8.6	8.6
T09	22	2.2	2.2
T10	NULL	NULL	NULL
T11	826	82.6	82.6
T12	507	50.7	50.7
T13	802	80.2	80.2

Рис. 5.6а. Результат исполнения запроса, представленного в листинге 5.6, в среде СУБД Microsoft Access, Oracle и MySQL. Обратите внимание, что деление целого на целое дает рациональное число с плавающей десятичной точкой

title_id	pages	pages/10	pages/10.0
-----	-----	-----	-----
T01	107	10	10.7
T02	14	1	1.4
T03	1226	122	122.6
T04	510	51	51.0
T05	201	20	20.1
T06	473	47	47.3
T07	333	33	33.3
T08	86	8	8.6
T09	22	2	2.2
T10	NULL	NULL	NULL
T11	826	82	82.6
T12	507	50	50.7
T13	802	80	80.2

Рис. 5.6б. Результат исполнения запроса, представленного в листинге 5.6, в среде СУБД Microsoft SQL Server и PostgreSQL. Обратите внимание, что деление целого на целое дает целое

Таблица 5.2. Порядок вычисления операторов (от высшей очередности к низшей)

Оператор	Описание
+, -	Унарные операторы тождества и отрицания
*, /	Бинарные арифметические операторы умножения и деления
+, -	Бинарные арифметические операторы сложения и вычитания
=, <>, <, <=, >, >=	Операторы сравнения
NOT	Логический оператор NOT
AND	Логический оператор AND
OR	Логический оператор OR

Листинг 5.7. В этом запросе первое и второе предложения показывают, как применять круглые скобки, чтобы переопределять правила предшествования, а третье и четвертое предложения показывают, как применять круглые скобки для того, чтобы переопределять правила ассоциативности. Результат исполнения запроса см. на рис. 5.7

Листинг

```
SELECT 2 + 3 * 4 AS "2+3*4",
       (2 + 3) * 4 AS "(2+3)*4",
       6 / 2 * 3 AS "6/2*3",
       6 / (2 * 3) AS "6/(2*3)";
```

2+3*4	(2+3)*4	6/2*3	6/(2*3)
-----	-----	-----	-----
14	20	9.00	1.00

Рис. 5.7. Результат исполнения запроса, представленного в листинге 5.7

Определение последовательности вычисления

Если в каком-нибудь выражении принимают участие несколько операторов, их приоритет в процессе вычисления такого выражения определяется правилами предшествования, или *очередностью* вычисления операторов (эти правила задаются стандартом SQL, с одной стороны, и дополняются и корректируются вашей СУБД, с другой стороны). Эти правила очень просты. Операторы выполняются в порядке убывания их очередности, то есть чем выше очередность оператора, тем раньше этот оператор исполняется (вычисляется). В частности, очередность арифметических операторов (+, -, *, / и т.п.) выше, чем очередность операторов сравнения (<, =, > и т.п.), у которых очередность выше, чем у логических операторов (NOT, AND, OR). Отсюда следует, что выражение `a or b * c >= d` эквивалентно выражению `a or ((b * c) >= d)`. Другими словами, операторы с низкой очередностью менее *обязывающие*, чем операторы с высокой очередностью. В табл. 5.2 операторы перечислены в порядке убывания их очередности, начиная с высшей. Операторы, оказавшиеся в одной строке этой таблицы, имеют одну и ту же очередность.

Отметим, что, если в произвольном выражении любые соседние операторы, не разделенные круглыми скобками, имеют равные очередности вычисления, фактический порядок их вычисления (то есть тот, в котором они будут вычислены в конкретном выражении) определяется действующим *правилом ассоциативности*. Имейте в виду, что SQL устанавливает так называемую ассоциативность слева направо.

Вы всегда можете применить круглые скобки так, чтобы добиться нужной очередности вычисления операторов, несмотря на все правила предшествования и ассоциативности (см. листинг 5.7 и рис. 5.7).

П Хороший стиль программирования предполагает, что вы будете применять круглые скобки в сложных выражениях даже тогда, когда делать это необязательно (просто для того, чтобы ваша программа была максимально читабельной).



В табл. 5.2 нет нескольких стандартных операторов SQL (например, `IN` и `EXISTS`) и нескольких нестандартных операторов, зависящих от выбора конкретной СУБД. Чтобы окончательно прояснить вопрос о порядке вычисления, который применяет ваша СУБД, организуйте поиск в ее документации по ключу «правила предшествования» или «очередность» (*precedence*).

Чтобы запустить запрос из листинга 5.7 в среде СУБД Oracle, вам придется добавить в предложение `SELECT` специальную таблицу `DUAL`. Подробнее на эту тему см. в примечании **DBMS** в разделе «Создание производных столбцов» этой главы.

Объединение строк с помощью оператора ||

Чтобы объединить, или *конкатенировать*, связать строки, применяют *оператор конкатенации* ||. Перечислим его основные характеристики:

- знаком оператора конкатенации является символ ||, состоящий из двух последовательных символов |;
- конкатенация не вставляет пробел между строками;
- конкатенация — это бинарный (двухместный) оператор, который объединяет две строки в одну, например: выражение 'formal' || 'dehyde' дает в результате строку 'formaldehyd';
- для получения сложной строки оператор конкатенации можно применять несколько раз, например: выражение 'a' || 'b' || 'c' || 'd' дает в результате строку 'abcd';
- конкатенация любой строки с пустой строкой оставляет первую строку без изменения, например: выражение 'a' || '' || 'b' дает в результате строку 'ab';
- результатом любой операции конкатенации, в которой одним из ее операндов служит значение null, является значение null, например: выражение 'a' || NULL || 'b' дает в результате NULL (однако советуем прочитать об исключении, которое относится к СУБД Oracle, в примечании с пометкой **DBMS** этого раздела);

- чтобы связать произвольную строку с произвольной нестрокой (например, с каким-нибудь значением даты и времени или числом), вам придется сначала преобразовать эту самую нестроку в некую строку, если только СУБД не сделает это автоматически (см. раздел «Преобразование типов данных с помощью функции CAST()» в этой главе).

Объединение двух строк

Напечатайте `string1 || string2`.

Предполагается, что:

- вы замените `string1` и `string2` теми строками, которые хотите конкатенировать (объединить);
- каждый операнд представляет собой строковое выражение, как, например, заголовок (имя) столбца, которому назначено содержать строки символов, произвольный строковый литерал, результат какой-нибудь операции или функции, выдающей некую строку (см. листинги и рис. 5.8–5.11).



Вы можете применять оператор конкатенации везде, где используется какое-либо выражение, в частности в предложениях SELECT, WHERE, ORDER BY.



С помощью оператора конкатенации можно объединять строки битов, например: `B'0100' || B'1011'`, что дает в результате `B'01001011'`.

П

Чтобы удалить ненужные пробелы из произвольной объединенной строки, можно применить функцию `TRIM()`. В разделе «Типы данных» главы 3 мы говорили, что значения `CHAR` дополняются длинными хвостами, которые иногда создают уродливые цепи в конкатенированных строках. Так вот, функция `TRIM()` удалит ненужные пробелы. Например, на рис. 5.8 вы можете видеть, что перед именем `Kellsey` стоит ненужный пробел. Следовательно, здесь можно воспользоваться функцией `TRIM()`, и она его уберет.

Подробнее этот материал освещается в разделе «Удаление пробелов с помощью функции `TRIM()`» этой главе.

Листинг 5.8. Перечислить имена и фамилии авторов, учитываемых в нашей типовой базе данных, чтобы эти имена и фамилии при распечатке объединились в один производный столбец, который впоследствии должен быть отсортирован в алфавитном порядке по столбцам фамилии и имени (то есть по двум столбцам `au_lname` и `au_fname`). Результат исполнения этого запроса см. на рис. 5.8

```
SELECT au_fname || ' ' || au_lname
       AS "Author name"
FROM authors
ORDER BY au_lname ASC, au_fname ASC;
```

```
Author name
-----
Sarah Buchman
Wendy Heydemark
Hallie Hull
Klee Hull
Christian Kells
    Kellsey
Paddy O'Furniture
```

Рис. 5.8. Результат исполнения запроса, представленного в листинге 5.8

Листинг 5.9. Перечислить в порядке убывания объемы продаж (то есть общее количество когда-либо проданных копий) книг-биографий, учитываемых в нашей типовой базе данных. Обратите внимание, здесь нам понадобилось преобразовать значения объемов продаж из типа данных INTEGER (целые числа) в символьный тип данных CHAR(7). Результат исполнения этого запроса см. на рис. 5.9

```

Листинг
SELECT CAST(sales AS CHAR(7))
       || ' copies sold of title '
       || title_id
       AS "Biography sales"
FROM titles
WHERE type = 'biography'
      AND sales IS NOT NULL
ORDER BY sales DESC;
```

Biography sales

```

-----
1500200 copies sold of title T07
100001  copies sold of title T12
11320   copies sold of title T06
```

Рис. 5.9. Результат исполнения запроса, представленного в листинге 5.9

Листинг 5.10. Перечислить в порядке убывания даты публикации книг, учитываемых в нашей типовой базе данных. Обратите внимание, нам пришлось конвертировать значения pubdate из типа данных даты и времени в некий символьный тип. Результат исполнения этого запроса см. на рис. 5.10

```

Листинг
SELECT 'Title '
       || title_id
       || ' published on '
       || CAST(pubdate AS CHAR(10))
       AS "Biography publication dates"
FROM titles
WHERE type = 'biography'
      AND pubdate IS NOT NULL
ORDER BY pubdate DESC;
```

Biography publication dates

```

-----
Title T12 published on 2000-08-31
Title T06 published on 2000-07-31
Title T07 published on 1999-10-01
```

Рис. 5.10. Результат исполнения запроса, представленного в листинге 5.10



В СУБД Microsoft Access оператором конкатенации является +, а функцией преобразования типов данных — Format(string). Чтобы в этой среде выполнить запросы из листингов 5.8–5.11, внесите следующие изменения в выражения с операторами конкатенации и с функциями конверсии типов:

- au_fname + ' ' + au_lname — для листинга 5.8;
- FORMAT(sales)
→ + ' copies sold of title '
→ + title_id — для листинга 5.9;
- 'Title '
→ + title_id
→ + ' published on '
→ + FORMAT(pubdate) — для листинга 5.10;
- au_fname + ' ' + au_lname
→ = 'Klee Hull'; — для листинга 5.11.

В СУБД Microsoft SQL Server оператором конкатенации является «плюс» (+). Чтобы в этой среде выполнить запросы из листингов 5.8–5.11, внесите следующие изменения в выражения с операторами конкатенации:

- au_fname + ' ' + au_lname — для листинга 5.8;
- CAST(sales AS CHAR(7))
→ + ' copies sold of title '
→ + title_id — для листинга 5.9;
- 'Title '
→ + title_id
→ + ' published on '
→ + CAST(pubdate AS CHAR(10)) — для листинга 5.10;
- au_fname + ' ' + au_lname
→ = 'Klee Hull'; — для листинга 5.11.

В СУБД MySQL оператор || означает логическое OR, а функцией конкатенации является CONCAT(), на вход которой можно передать любое количество аргументов и которая автоматически конвертирует любые данные, не являющиеся строками символов, в строки символов так, что функция CAST()

становится ненужной. Чтобы в среде MySQL выполнить запросы, представленные в листингах 5.8–5.11, внесите следующие изменения в выражения с операторами конкатенации:

- CONCAT(au_fname + ' ' + au_lname)
— для листинга 5.8;
- CONCAT(sales,
→ ' copies sold of title '
→ title_id) — для листинга 5.9;
- CONCAT('Title ',
→ title_id,
→ ' published on ',
→ pubdate) — для листинга 5.10;
- CONCAT(au_fname, ' ', au_lname)
→ = 'Klee Hull'; — для листинга 5.11.

СУБД Oracle трактует любую пустую строку как значение null. Поэтому выражение 'a' || NULL || 'b' в этой среде означает 'ab' (см. примечания с пометкой **DBMS** в разделе «Значение null» главы 3).

При конкатенации СУБД Oracle, MySQL и PostgreSQL автоматически и неявно (то есть не требуя вашего вмешательства) конвертируют все нестроки в строки. Поэтому запросы из листингов 5.9 и 5.10 будут работать в среде этих СУБД, даже если вы опустите в них функцию CAST(). Чтобы разобраться с этим вопросом до конца, организуйте поиск в документации по своей СУБД с использованием ключей «конкатенация» (concatenation) и «преобразование» (conversion).

Листинг 5.11. Перечислить всех авторов, учитываемых в нашей типовой базе данных, чье полное имя – Klee Hull. Результат исполнения запроса см. на рис. 5.11

Листинг		
<pre>SELECT au_id, au_fname, au_lname FROM authors WHERE au_fname ' ' au_lname = 'Klee Hull';</pre>		
au_id	au_fname	au_lname
-----	-----	-----
A04	Klee	Hull

Рис. 5.11. Результат исполнения запроса, представленного в листинге 5.11

Выбор произвольной подстроки с помощью функции SUBSTRING()

Для выбора любой части произвольной строки символов применяют функцию SUBSTRING(). Перечислим ее основные характеристики:

- всякая подстрока – это некая последовательность символов исходной строки, включая пустую и исходную строки целиком, в которых символы расположены подряд;
- функция SUBSTRING() выделяет часть произвольной строки определенной длины, выраженной количеством знаков в подстроке, начиная от указанной позиции в исходной строке;
- любая подстрока пустой строки есть пустая строка;
- если какой-либо аргумент функции SUBSTRING() является значением null, на выходе функции будет тоже значение null (об исключении из этого правила, связанного с СУБД Oracle, говорится в примечании с пометкой DBMS в этом разделе).

Порядок выбора произвольной подстроки

Напечатайте:

```
SUBSTRING(string FROM start [FOR length])
```

Предполагается, что:

- вместо *string* вы подставите явно или укажете на ту исходную строку, из которой хотите выбрать некую подстроку и которая может быть любым строковым выражением, например:
 - значением столбца произвольной таблицы, который должен содержать

строки символов (на такую исходную строку следует указывать заголовком этого столбца);

- произвольным строковым литералом;
 - результатом операции или функции, у которых на выходе должна быть строка символов;
- вместо *start* вы подставите целое число, которое обозначает место первого символа подстроки в исходной строке;
- вместо *length* вы подставите целое число, которое равно общему числу знаков в выделяемой подстроке, имея при этом в виду, что, если опустить предложение *FOR length*, функция *SUBSTRING()* выдаст в качестве подстроки весь хвост исходной строки, начиная с указанного (с помощью *start*) первого символа и заканчивая последним символом исходной строки *string* (см. листинги и рис. 5.12–5.14).

П

Вы можете применять функцию выделения подстроки *SUBSTRING()* везде, где используется какое-либо выражение, в частности в предложениях *SELECT*, *WHERE*, *ORDER BY*.

П

С помощью функции выделения подстроки *SUBSTRING()* можно выделять подстроки из строк битов, например: *SUBSTRING(B'01001011', 5, 4)* даст в результате *B'1011'*.



В СУБД Microsoft Access функцией выделения подстроки является *Mid(string, start[, length])*, а для выполнения конкатенации строки надо использовать оператор *+*. Поэтому для выполнения в этой среде запросов, представленных в листингах 5.12–5.14, вам придется внести в них следующие изменения:

Листинг 5.12. Разбить идентификаторы тех издателей, которые учитываются в нашей типовой базе данных, на буквенную и цифровую составляющие. При этом буквенная часть идентификатора любого из рассматриваемых издателей – это первый символ идентификатора, а все остальные символы (то есть хвост) составляют цифровую часть. Результат исполнения запроса см. на рис. 5.12

```

Листинг
SELECT      pub_id,
            SUBSTRING(pub_id FROM 1 FOR 1)
              AS "Alpha part",
            SUBSTRING(pub_id FROM 2)
              AS "Num part"
FROM publishers;
```

pub_id	Alpha part	Num part
P01	P	01
P02	P	02
P03	P	03
P04	P	04

Рис. 5.12. Результат исполнения запроса, представленного в листинге 5.12

Листинг 5.13. Распечатать из нашей базы данных первую букву (инициал) имени, точку, пробел и фамилию всех авторов, живущих или в штате Нью-Йорк, или в штате Колорадо. Результат исполнения запроса см. на рис. 5.13

Листинг	
<pre>SELECT SUBSTRING(au_fname FROM 1 FOR 1) '. ' au_lname AS "Author name", state FROM authors WHERE state IN ('NY', 'CO');</pre>	
Author name	state
-----	-----
S. Buchman	NY
W. Heydemark	CO
C. Kells	NY

Рис. 5.13. Результат исполнения запроса, представленного в листинге 5.13

Листинг 5.14. Перечислить имена и фамилии учитываемых в нашей типовой базе данных авторов, у которых телефонный номер начинается на 415. Результат исполнения запроса см. на рис. 5.14

```

SELECT au_fname, au_lname, phone
FROM authors
WHERE SUBSTRING(phone FROM 1 FOR
3)='415';

```

Рис. 5.14. Результат исполнения запроса, представленного в листинге 5.14

- Mid(pub_id, 1, 1)
...
Mid(pub_id, 2) – для запроса, представленного в листинге 5.12;
- Mid(au_fname, 1, 1) + '. ' +
→ au_lname – для запроса, представленного в листинге 5.13;
- Mid(phone, 1, 3)='415' – для запроса, представленного в листинге 5.14.

В СУБД Microsoft SQL Server функцией выделения подстроки является SUBSTRING(*string*, *start*, *length*), а для объединения строк надо применять оператор +. Чтобы в этой среде выполнить запросы, представленные в листингах 5.12–5.14, вам придется внести в них следующие изменения:

- SUBSTRING(pub_id, 1, 1)
...
SUBSTRING(pub_id, 2 LEN(pub_id)-1)
–
для запроса, представленного в листинге 5.12;
- SUBSTRING(au_fname, 1, 1)
→ + '. '
→ + au_lname – для запроса, представленного в листинге 5.13;
- SUBSTRING(phone, 1, 3)='415' – для запроса, представленного в листинге 5.14.

В СУБД Oracle функцией выделения подстроки является SUBSTR(*string*, *start*, [*length*]). Поэтому, чтобы в этой среде выполнить запросы из листингов 5.12–5.14, вам придется внести в них следующие изменения:

- SUBSTR(pub_id, 1, 1)
...
SUBSTR(pub_id, 2) – для запроса, представленного в листинге 5.12;
- SUBSTRING(au_fname, 1, 1)
→ a|| '. '
→ a|| au_lname – для запроса, представленного в листинге 5.13;

- `SUBSTR(phone, 1, 3)='415'` — для запроса, представленного в листинге 5.14.

Если вы работаете в среде СУБД MySQL, то для выполнения запроса из листинга 5.13 вам придется применить функцию `CONCAT()` и изменить выражение с оператором конкатенации следующим образом:

```
CONCAT(  
→ SUBSTRING(au_fname, FROM 1 FOR  
1),  
→ ' ',  
→ au_lname)
```

Советуем еще раз обратиться к разделу «Объединение строк с помощью оператора ||» в этой главе.

СУБД Oracle воспринимает любую пустую строку как значение `null`. Например, функция выделения подстроки `SUBSTR(NULL, 1, 2)` выдаст на выходе значение '' (пустая строка). Прочитайте еще раз примечания с пометкой **DBMS** в разделе «Значение null» главы 3.

СУБД может автоматически и неявно (то есть не спрашивая вашего решения) ограничить аргументы *start* и/или *length*, которые являются или слишком маленькими, или слишком большими значениями, и заменить их разумными с точки зрения СУБД значениями. Например, если вместо *start* поставить какое-нибудь отрицательное число, СУБД может заменить значение *start* единицей, а если вместо *length* использовать слишком большое число, выводящее подстроку за рамки исходной строки, СУБД может заменить его длиной всей строки. Чтобы как следует разобраться в этом вопросе, организуйте поиск в документации по вашей СУБД с использованием ключа «substring» или «substr».

Переключение регистра символов строки с использованием функций UPPER() и LOWER()

Функцию UPPER() применяют для конвертации произвольной строки символов нижнего регистра в строку символов верхнего регистра, чтобы поменялся только регистр символов (буквы должны быть те же самые, но заглавные). Функция LOWER() используется для конвертации произвольной строки символов верхнего регистра в строку символов нижнего регистра, чтобы поменялся только регистр символов (буквы должны быть те же самые, но строчные). Основные характеристики функций UPPER() и LOWER() таковы:

- произвольный символ, который может находиться как в верхнем (A), так и в нижнем (a) регистре, называется *регистровым*;
- регистровыми символами являются только буквы, следовательно, переключение регистра затрагивает только буквы, а цифры, знаки пунктуации и пробелы при этом остаются неизменными;
- функции UPPER() и LOWER() обычно применяют для того, чтобы форматировать результаты и производить сравнения без учета регистра символов в каком-нибудь предложении WHERE (функция UPPER()/LOWER() меняет регистр только тех букв, которые были в нижнем/верхнем регистре, а регистр тех букв, которые и так были в верхнем/нижнем регистре, остается прежним);
- на пустые строки изменение регистра не оказывает никакого влияния;
- если аргумент любой из функций UPPER() и LOWER() является значением null,

на выходе этой функции будет тоже значение null (но обратите внимание на исключение из этого правила, которое связано с СУБД Oracle и о котором говорится в примечании DBMS этого раздела).

Переключение регистра произвольной строки символов

Напечатайте:

- UPPER(*string*) – чтобы переключится на верхний регистр;
- LOWER(*string*) – чтобы переключиться на нижний регистр.

Предполагается, что вместо *string* вы подставите явно или укажете на ту исходную строку, у которой хотите переключить регистр и которая может быть любым строковым выражением, например (см. листинги и рис. 5.15–5.16):

- значением произвольного столбца таблицы, которое должно содержать строки символов (на такую исходную строку следует указывать заголовком этого столбца);
- произвольным строковым литералом;
- результатом некой операции или функции, чтобы в итоге получилась строка символов.



Функции переключения регистра UPPER() и LOWER() можно применять везде, где используется какое-либо выражение, в частности в предложениях SELECT, WHERE, ORDER BY.



Функции переключения регистра UPPER() и LOWER() не оказывают никакого воздействия на такие наборы символов (в том числе алфавиты), для которых концепция регистра не предусмотрена (например, алфавит иврита).

П

Функции переключения регистра `UPPER()` и `LOWER()` влияют на символы, находящиеся под каким-нибудь диакритическим знаком (скажем, ударения), например: `UPPER('ó')` дает 'Ó'.



В СУБД Microsoft Access функциями переключения регистра являются `UCase(string)` и `LCase(string)`. Поэтому, чтобы выполнить в этой среде запросы из листингов 5.15 и 5.16, вам придется внести в текст предложений, отвечающих за переключение регистра, следующие изменения:

- `LCase(au_fname)` и `UCase(au_lname)` — для запроса, представленного в листинге 5.15;
- `UCase(title_name) LIKE '%M0%'` — для запроса, представленного в листинге 5.16.

СУБД Oracle воспринимает любую пустую строку как значение `null`. Поэтому и функция `UPPER(NULL)`, и функция `LOWER(NULL)` выдадут на выходе значение '' (пустая строка). Советуем еще раз обратиться к примечаниям с пометкой **DBMS** в разделе «Значение null» главы 3.

СУБД может предоставлять дополнительные возможности форматирования строки, такие как инверсия регистра или конверсия строк в предложения или заголовки. Чтобы как следует разобраться с этим вопросом, проанализируйте документацию по вашей СУБД с использованием ключа «символьные функции» (*character functions*).

Листинг 5.15. Показать имена авторов, учитываемых в нашей типовой базе данных, в нижнем регистре, а их фамилии — в верхнем регистре. Результат исполнения запроса см. на рис. 5.15

Листинг	
<pre>SELECT LOWER(au_fname) AS "Lower", UPPER(au_lname) AS "Upper" FROM authors;</pre>	
Lower	Upper
-----	-----
sarah	BUCHMAN
wendy	HEYDEMARK
hallie	HULL
klee	HULL
christian	KELLS
	KELLSEY
paddy	O' FURNITURE

Рис. 5.15. Результат исполнения запроса, представленного в листинге 5.15

Листинг 5.16. Перечислить названия книг, учитываемых в нашей типовой базе данных, которые содержат сочетание (латинских) букв «МО», независимо от регистра. Чтобы этот запрос сработал, все буквы шаблона `LIKE` должны быть в верхнем регистре. Результат исполнения запроса см. на рис. 5.16

Листинг	
<pre>SELECT title_name FROM titles WHERE UPPER(title_name) LIKE 'M0%';</pre>	
title_name	

200 Years of German Humor	
I Blame My Mother	

Рис. 5.16. Результат исполнения запроса, представленного в листинге 5.16

Удаление пробелов с помощью функции TRIM()

Функцию TRIM() применяют для того, чтобы отсекал нежелательные символы в конце строк. Основные характеристики функции TRIM() таковы:

- с ее помощью можно отсекал символы только с начала или с конца строки или одновременно с начала и с конца, но ни у одной строки нельзя вырезать символы из середины;
- по умолчанию эта функция отсекает пробелы, но вы можете убрать с ее помощью любые нежелательные символы, например нули или звездочки, в начале или/и в конце какой-нибудь строки;
- обычно ее применяют для того, чтобы форматировать результаты запросов и производить сравнения предложением WHERE;
- эта функция особенно полезна в том случае, когда надо удалить длинные цепочки пробелов на хвостах значений типа данных CHAR (в разделе «Типы данных» главы 3 мы отмечали, что СУБД автоматически дополняют значения типа данных CHAR хвостами пробелов, чтобы создать строки символов той длины, которая для них указана);
- на пустые строки она не влияет;
- если какой-либо ее аргумент является значением null, на выходе функции будет тоже значение null (советуем обратить внимание на исключение из этого правила, которое связано с СУБД Oracle, – см. примечание DBMS в этом разделе).

Как удалить пробелы в строке

Напечатайте:

```
TRIM([[LEADING | TRAILING | BOTH]
FROM] string)
```

Предполагается, что:

- вместо *string* вы подставите явно или укажете на ту исходную строку, у которой хотите отсечь пробелы и которая может быть любым строковым выражением, например:
 - значением произвольного столбца произвольной таблицы, который должен содержать строки символов (на такую исходную строку следует указывать заголовком этого столбца);
 - произвольным строковым литералом;
 - результатом некой операции или функции, на выходе которых должна быть строка символов;
- вы укажете опцию LEADING в том случае, если необходимо отсечь головные пробелы;
- вы укажете опцию TRAILING в том случае, если необходимо отсечь хвостовые пробелы;
- вы укажете опцию BOTH в том случае, если необходимо удалить одновременно и хвостовые, и головные пробелы;
- вы можете не указывать опции LEADING, TRAILING и BOTH, и тогда СУБД назначит по умолчанию опцию BOTH (см. листинг 5.17 и рис. 5.17).

Как отделить символы от произвольной строки

Напечатайте:

```
TRIM([LEADING | TRAILING | BOTH]
      'trim_chars' FROM string)
```

Предполагается, что (см. листинги и рис. 5.18 и 5.19):

- вместо *trim_chars* вы подставите один или несколько символов, которые хотите отсечь от строки, обозначенной как *string*;
- вместо *string* подставите явно или укажете на ту исходную строку, у которой хотите удалить символы, обозначенные как *trim_chars*;
- вместо *trim_chars* и *string* вы будете подставлять какие-нибудь строковые выражения по своему выбору, например:
 - значение произвольного столбца произвольной таблицы, который должен содержать строки символов (на такую исходную строку следует указывать заголовком этого столбца);
 - произвольный строковый литерал;
 - результат некой операции или функции, у которых на выходе должна быть строка символов;
- вы укажете опцию **LEADING** в том случае, если необходимо отсечь головные символы;
- вы укажете опцию **TRAILING** в том случае, если необходимо удалить хвостовые символы;
- вы укажете опцию **BOTH** в том случае, если необходимо убрать одновременно и хвостовые, и головные символы;
- вы можете не указывать опции **LEADING**, **TRAILING** и **BOTH**, и тогда СУБД назначит по умолчанию опцию **BOTH**.

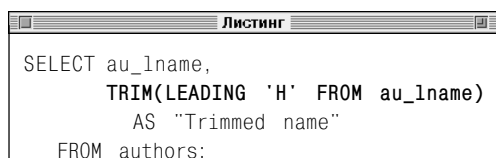
Листинг 5.17. Последовательно отсечь и головные, и хвостовые пробелы у строки ' AAA '.

Символы < (меньше) и > (больше) обозначают длину усеченных строк. Результат исполнения запроса см. на рис 5.17

Листинг			
<pre>SELECT '<' ' AAA ' '>' AS "Untrimmed", '<' TRIM(LEADING FROM ' AAA ') '>' AS "Leading", '<' TRIM(TRAILING FROM ' AAA ') '>' AS "Trailing", '<' TRIM(' AAA ') '>' AS "Both";</pre>			
Untrimmed	Leading	Trailing	Both
-----	-----	-----	-----
< AAA >	<AAA >	< AAA>	<AAA>

Рис. 5.17. Результат исполнения запроса, представленного в листинге 5.17

Листинг 5.18. Удалить первую заглавную латинскую букву «Н» в тех фамилиях авторов, учитываемых в нашей типовой базе данных, которые начинаются именно с нее. Результат исполнения запроса см. на рис. 5.18

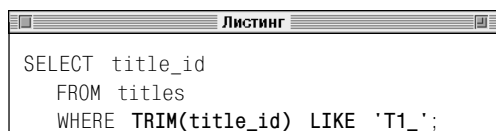


```
SELECT au_lname,
       TRIM(LEADING 'H' FROM au_lname)
       AS "Trimmed name"
FROM authors;
```

au_lname	Trimmed name
-----	-----
Buchman	Buchman
Heydemark	eydemark
Hull	ull
Hull	ull
Kells	Kells
Kellsey	Kellsey
O'Furniture	O'Furniture

Рис. 5.18. Результат исполнения запроса, представленного в листинге 5.18

Листинг 5.19. Игнорируя головные и хвостовые пробелы, перечислить трехзначные идентификаторы названий книг, учитываемых в нашей типовой базе данных, которые начинаются с буквенно-цифрового сочетания T1. Результат исполнения запроса представлен на рис. 5.19



```
SELECT title_id
FROM titles
WHERE TRIM(title_id) LIKE 'T1_';
```

title_id

T10
T11
T12
T13

Рис. 5.19. Результат исполнения запроса, представленного в листинге 5.19

П

Функцию TRIM() можно использовать там, где применяют какое-либо выражение, в частности в предложениях SELECT, WHERE, ORDER BY.

П

В запросе из листинга 5.8 мы объединили в один производный столбец имена и фамилии авторов, учитываемых в нашей типовой базе данных. Результат исполнения этого запроса представлен на рис. 5.8. Мы видим «лишний» пробел перед фамилией Kellsey. Этот пробел оказывается лишним только в строке, соответствующей Келлси. Он должен разделять имена и фамилии авторов, но у Келлси нет имени, и СУБД поставила перед этой фамилией пробел, который и стал лишним. Так вот, чтобы убрать этот пробел, можно воспользоваться функцией TRIM(). С этой целью вам следует изменить текст выражения из листинга 5.8, которое содержит операторы конкатенации, следующим образом:

```
TRIM(au_fname || ' ' || au_lname)
```



В СУБД Microsoft Access функциями отсе- чения (их еще называют функциями трим- мирования) являются:

- LTrim(string) – для отсе- чения голов- ных пробелов;
- RTrim(string) – для отсе- чения хвос- товых пробелов;
- Trim(string) – для удаления и голов- ных, и хвостовых пробелов одновре- менно.

Чтобы в среде этой СУБД от произвольной строки отсечь символы, не являющиеся пробелами, следует использовать функцию Replace(string, find, replacement [,start] [,count] [,compare]), которая заменит указанные символы пус- тыми строками.

Для того чтобы объединить строки в среде СУБД Microsoft Access, применяйте опера- тор +. В частности, если нужно выполнить запросы из листингов 5.17 и 5.18, в текст этих запросов придется внести соответ- ствующие изменения:

- '<<' + ' AAA ' + '>'
- '<<' + LTRIM(' AAA ') + '>'
- '<<' + RTRIM(' AAA ') + '>'
- '<<' + TRIM(' AAA ') + '>' – для запроса, приведенного в листинге 5.17;
- Replace(au_lname, 'H', '', 1, 1) – для запроса, представленного в листинге 5.18.

В СУБД Microsoft SQL Server функциями отсечения (их еще называют функциями триммирования) являются:

- LTRIM(string) – для отсечения головных пробелов;
- RTRIM(string) – для удаления хвостовых пробелов.

Чтобы объединить строки в среде этой СУБД, применяйте оператор +. В частности, если нужно запустить запрос из листинга 5.17, в текст этого запроса придется внести соответствующие изменения:

```
'<' + ' AAA ' + '>'
'<' + LTRIM(' AAA ') + '>'
'<' + RTRIM(' AAA ') + '>'
'<' + LTRIM(RTRIM(' AAA ')) + '>'
```

Функции отсечения LTRIM(string) и RTRIM(string) в среде СУБД Microsoft SQL Server удаляют именно пробелы, а не произвольные символы, обозначенные как trim_chars. Поэтому, чтобы удалить произвольные символы и запустить на выполнение запросы из листингов 5.18 и 5.19, вам придется встраивать в текст этих запросов и комбинировать какие-нибудь символьные функции, в том числе характерные только для СУБД Microsoft SQL Server, например CHARINDEX(), LEN(), PATINDEX(), REPLACE(), STUFF(), SUBSTRING. В частности, в тексты выражений усечения соответствующих запросов можно внести такие изменения:

- REPLACE(
 - SUBSTRING(au_lname, 1, 1), 'H', '')
 - SUBSTRING(au_lname, 2,
 - LEN(au_lname)) – для листинга 5.18;
- LTRIM(RTRIM(title_id)) LIKE 'T1_' – для листинга 5.19.

Чтобы в среде СУБД Oracle исполнить запрос из листинга 5.17, в текст этого запроса придется добавить предложение FROM DUAL. Советуем перечитать примечание с пометкой **DBMS** в разделе «Создание производных столбцов» этой главы. Имейте в виду, что Oracle не разрешает подставлять вместо trim_chars сразу несколько символов.

Чтобы в среде СУБД MySQL исполнить запрос, представленный в листинге 5.17, в текст этого запроса придется встроить функцию конкатенации CONCAT() (см. раздел «Объединение строк с помощью оператора ||» в этой главе). Исправьте выражения конкатенации запроса из листинга 5.17 следующим образом:

```
CONCAT('<', ' AAA ', '>')
CONCAT('<',
→ TRIM(LEADING FROM ' AAA '),
→ '>')
CONCAT('<',
→ TRIM(TRAILING FROM ' AAA '),
→ '>')
CONCAT('<', TRIM(' AAA '), '>')
```

СУБД Oracle воспринимает любую пустую строку как значение null. Поэтому функция TRIM(NULL) выдаст на выходе значение '' (пустая строка). В этой связи посмотрите еще раз примечания с пометкой **DBMS** в разделе «Значение null» главы 3.

Возможно, СУБД предоставляет в ваше распоряжение так называемые функции-заполнители, которые должны вставлять символы пробелов или какие-либо другие символы в свободные места символьных строк. Например, в СУБД Oracle такими функциями-заполнителями являются LPAD() и RPAD(). Для того чтобы детально разобраться с функциями-заполнителями, организуйте поиск в документации по своей СУБД с использованием ключа «символьные функции» (character functions).

Листинг 5.20. Вычислить длину имен авторов, учитываемых в нашей типовой базе данных. Результат исполнения запроса см. на рис. 5.20

```

SELECT au_fname,
       CHARACTER_LENGTH(au_fname) AS "Len"
FROM authors;

```

au_fname	Len
-----	----
Sarah	5
Wendy	5
Hallie	6
Klee	4
Christian	9
	0
Paddy	5

Рис. 5.20. Результат исполнения запроса, представленного в листинге 5.20

Определение длины произвольной строки с помощью функции CHARACTER_LENGTH()

Чтобы вывести длину произвольной строки символов как число символов этой строки, применяют функцию CHARACTER_LENGTH(). Назовем ее основные характеристики:

- на ее выходе всегда будет или какое-нибудь натуральное число, или ноль (тип INTEGER);
- эта функция подсчитывает символы, а не байты: произвольный символ, состоящий из нескольких байтов, или произвольный символ Unicode представляют для этой функции всего один символ (как считать байты, описывается в примечаниях этого раздела);
- длина любой пустой строки равна нулю;
- если аргументом этой функции является значение null, на ее выходе тоже будет значение null (обращаем ваше внимание на исключение из этого правила, которое связано с СУБД Oracle, – см. примечание с пометкой **DBMS** в этом разделе).

Вычисление длины произвольной строки

Напечатайте CHARACTER_LENGTH(*string*).

Предполагается, что вместо *string* вы подставите явно или укажете на ту исходную строку (см. листинги и рис. 5.20 и 5.21), для которой хотите вычислить длину, причем вместо *string* будете подставлять какие-нибудь строковые выражения по своему выбору, например:

- значение произвольного столбца произвольной таблицы, который должен содержать строки символов (на такую исходную строку следует указывать заголовком этого столбца);
- произвольный строковый литерал;
- результат некой операции или функции, у которых на выходе должна быть строка символов.

П Вы можете применять функцию `CHARACTER_LENGTH()` везде, где используется какое-либо выражение, в частности в предложениях `SELECT`, `WHERE`, `ORDER BY`.

П Функции `CHAR_LENGTH()` и `CHARACTER_LENGTH()` являются синонимами.

П Стандарт SQL определяет следующие строковые функции:

- `BIT_LENGTH(expr)` – выводит число битов в произвольном выражении, например `BIT_LENGTH(B'01001011')` равно 8;
- `OCTET_LENGTH(expr)` – выводит число байтов в произвольном выражении, например `OCTET_LENGTH(B'01001011')` равно 1, а `OCTET_LENGTH('ABC')` – 3.

Восьмеричная длина (то есть длина в байтах) равна округленному до ближайшего целого частному от деления двоичной длины (то есть длины в битах) на натуральное число 8. Подробнее о тех функциях, которые предоставляют конкретные СУБД для определения восьмеричной и двоичной длины, рассказывается в примечании с пометкой **DBMS** этого раздела.



В СУБД Microsoft Access и Microsoft SQL Server функцией определения длины строки является `LEN(string)`. Поэтому для запуска в среде этих СУБД запросов из листингов 5.20 и 5.21 вам придется внести в выражения с функциями определения длины строки следующие изменения:

- `LEN(au_fname)` – для листинга 5.20;
- `LEN(title_name)` – для листинга 5.21.

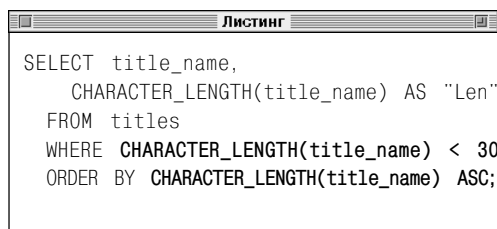
В СУБД Oracle функцией определения длины строки является `LENGTH(string)`. Поэтому для запуска в среде этой СУБД запросов из листингов 5.20 и 5.21 придется внести в выражения с функциями определения длины строки следующие изменения:

- `LENGTH(au_fname)` – для листинга 5.20;
- `LENGTH(title_name)` – для листинга 5.21.

Функции побитового и побайтового подсчета длины строки сильно меняются при переходе от одной СУБД к другой. Так, Microsoft Access предлагает `Len()`, Microsoft SQL Server – `DATALENGTH()`, Oracle – `LENGTHB()`, MySQL – `BIT_COUNT()` и `OCTET_LENGTH()`, наконец, PostgreSQL – `OCTET_LENGTH()`.

СУБД Oracle трактует любую пустую строку как значение `null`. Поэтому `LENGTH(NULL)` дает ''. Однако в предпоследней строке рис. 5.20 мы видим не 0, а 1. Это происходит потому, что вместо имени соответствующего автора в СУБД Oracle стоит пробел ' ', а не пустая строка '' (см. примечание с пометкой **DBMS** в разделе «Значение null» главы 3).

Листинг 5.21. Перечислить в порядке возрастания длины (восходящий порядок) названия книг, учитываемых в нашей типовой базе данных, которые содержат менее 30 символов. Результат исполнения запроса см. на рис. 5.21



```

SELECT title_name,
       CHARACTER_LENGTH(title_name) AS "Len"
FROM titles
WHERE CHARACTER_LENGTH(title_name) < 30
ORDER BY CHARACTER_LENGTH(title_name) ASC;

```

title_name	Len
-----	----
1977!	5
Kiss My Boo-Boo	15
How About Never?	16
I Blame My Mother	17
Exchange of Platitudes	22
200 Years of German Humor	25
Spontaneous, Not Annoying	25
But I Did It Unconsciously	26
Not Without My Faberge Egg	26
Just Wait Until After School	28
Ask Your System Administrator	29

Рис. 5.21. Результат исполнения запроса, представленного в листинге 5.21

Поиск подстроки с использованием функции POSITION()

Чтобы найти любую заданную подстроку внутри произвольной заданной строки, применяют функцию POSITION(). Основные характеристики функции POSITION() таковы:

- она всегда дает на выходе целое неотрицательное число (тип INTEGER), которое указывает на позицию в той строке, в которой производится поиск, первого знака первого вхождения искомой подстроки;
- если та строка, в которой производится поиск, не содержит искомой подстроки, функция POSITION() выдаст ноль;
- строковые сравнения, определяющие работу этой функции, в зависимости от выбора конкретной СУБД могут быть как чувствительны, так и нечувствительны к регистру сравниваемых символов (см. примечания с пометкой **DBMS** в разделе «Фильтрация строк с помощью предложения WHERE» главы 4);
- позиция любой подстроки в любой пустой строке есть ноль;
- если хотя бы одним аргументом этой функции является значение null, на ее выходе тоже будет значение null (обратите внимание на исключение из этого правила, которое связано с СУБД Oracle, – см. далее в этом разделе примечание с пометкой **DBMS**).

Порядок поиска произвольной подстроки

Напечатайте `POSITION(substring IN string)`.

Предполагается, что:

- вместо *string* вы подставите явно или укажете на ту исходную строку, в которой хотите осуществить поиск подстроки;
- вместо *substring* подставите явно или укажете на ту подстроку, которую собираетесь искать в исходной строке;
- вместо *substring* и *string* будете подставлять какие-нибудь строковые выражения по своему выбору, например:
 - значение любого столбца произвольной таблицы, который должен содержать строки символов (на такую строку или подстроку следует указывать заголовком этого столбца);
 - произвольный строковый литерал;
 - результат некой операции или функции, на выходе которых должна быть строка символов.

Еще раз подчеркнем, что функция `POSITION()` всегда выдает минимальное значение позиции (целое неотрицательное число или ноль, тип `INTEGER`), на которой в исходной строке стоит первый символ искомой подстроки. Объясним смысл минимума: если искомая подстрока встречается в исходной строке несколько раз, таких позиций первого символа подстроки будет тоже несколько, и среди этих натуральных чисел, соответствующих позициям первых символов, берется минимальное. Если искомая подстрока не будет найдена в исходной строке, функция `POSITION()` выдаст ноль (см. листинги и рис. 5.22 и 5.23).

Листинг 5.22. Указать для всех авторов, учитываемых в нашей типовой базе данных, позицию подстроки *e* (латинская строчная буква) в строке, являющейся именем автора, и позицию подстроки *ma* (латинские строчные буквы) в строке, являющейся фамилией этого автора. Результат исполнения запроса см. на рис. 5.22

Листинг			
<pre>SELECT au_fname, POSITION('e' IN au_fname) AS "Pos e", au_lname, POSITION('ma' IN au_lname) AS "Pos ma" FROM authors;</pre>			
au_fname	Pos e	au_lname	Pos ma
-----	-----	-----	-----
Sarah	0	Buchman	5
Wendy	2	Heydemark	6
Hallie	6	Hull	0
Klee	3	Hull	0
Christian	0	Kells	0
	0	Kellsey	0
Paddy	0	O'Furniture	0

Рис. 5.22. Результат исполнения запроса, представленного в листинге 5.22

П

Функцию POSITION() можно использовать везде, где применяют какое-либо выражение, в частности в предложениях SELECT, WHERE, ORDER BY.



В СУБД Microsoft Access функцией определения подстроки является InStr(start_position, string, substring). Поэтому для выполнения в среде этой СУБД запросов из листингов 5.22 и 5.23 в выражения с функциями определения позиции подстроки придется внести следующие изменения:

- InStr(1, au_fname, 'e') и InStr(1, au_lname, 'ma') – для листинга 5.22;
- InStr(1, title_name, 'u') – для листинга 5.23.

В СУБД Microsoft SQL Server функцией определения подстроки является CHARINDEX(substring, string). Поэтому для выполнения в среде этой СУБД запросов из листингов 5.22 и 5.23 в выражения с функциями определения позиции подстроки придется внести следующие изменения:

- CHARINDEX('e', au_fname) и CHARINDEX('ma', au_lname) – для листинга 5.22;
- CHARINDEX('u', title_name) – для листинга 5.23.

В СУБД Oracle функцией определения подстроки является INSTR(string, substring). Поэтому для выполнения в среде этой СУБД запросов из листингов 5.22 и 5.23 в соответствующие выражения с функциями определения позиции подстроки придется внести следующие изменения:

- INSTR(au_fname, 'e') и INSTR(au_lname, 'ma') – для листинга 5.22;
- INSTR(title_name, 'u') – для листинга 5.23.

СУБД Oracle трактует любую пустую строку как значение null. Поэтому POSITION(NULL) дает '' (см. примечания с пометкой **DBMS** в разделе «Значение null» главы 3).

Чтобы найти вхождения произвольной подстроки в произвольную строку, отличные от первого, можно комбинировать и встраивать функцию локализации подстроки POSITION(). Однако коммерческие СУБД предоставляют в ваше распоряжение более мощные и удобные инструменты решения этой задачи. Так, Microsoft Access предлагает InStr(), Microsoft SQL Server – CHARINDEX(), Oracle – INSTR(), MySQL – LOCATE().

Листинг 5.23. Перечислить названия книг, учитываемых в нашей типовой базе данных, которые содержат не более 10 символов, считая слева направо, и латинскую строчную букву «u». Сортировка названий осуществляется по уменьшению (нисходящий порядок) величины значения позиции вхождения подстроки (то есть латинской строчной буквы «u»). Результат исполнения запроса см. на рис. 5.23

```

Листинг
SELECT title_name,
       POSITION('u' IN title_name) AS "Pos"
FROM titles
WHERE POSITION('u' IN title_name)
      BETWEEN 1 AND 10
ORDER BY POSITION('u' IN title_name) DESC;

```

title_name	Pos
Not Without My Faberge Egg	10
Spontaneous, Not Annoying	10
How About Never?	8
Ask Your System Administrator	7
But I Did It Unconsciously	2
Just Wait Until After School	2

Рис. 5.23. Результат исполнения запроса, представленного в листинге 5.23

Таблица 5.3. Операторы, операндами которых могут быть интервалы и данные даты и времени

Операция	Результат
(Дата и время) – (Дата и время)	Интервал
(Дата и время) + Интервал	Дата и время
(Дата и время) - Интервал	Дата и время
Интервал + (Дата и время)	Дата и время
Интервал + Интервал	Интервал
Интервал - Интервал	Интервал
Интервал * Число	Интервал
Интервал / Число	Интервал
Число * Интервал	Интервал

Операции с данными даты и времени



Как мы уже отмечали, соответствие коммерческих СУБД, которые рассматриваются в настоящей книге, требованиям, предъявляемым стандартом SQL к операторам и функциям над данными даты и времени и интервалами, весьма относительное. Эти СУБД имеют свои собственные дополнительные операторы и функции для выполнения, в частности, арифметических действий с данными интервальных типов и данными даты и времени. Поэтому настоящий раздел с самого начала задумывался как краткий справочник по соответствию коммерческих СУБД стандарту SQL. Хотим напомнить, что об интервальных типах данных и о типах данных даты и времени мы говорили в разделах «Типы данных» и «Интервальные типы данных» главы 3.

Начнем с того, что для выполнения арифметических операций с данными интервальных типов и типов даты и времени лучше всего применять операторы, о которых мы рассказывали в разделе «Арифметические операции» этой главы. Наиболее ходовыми задачами, сводящимися к операциям с данными даты, времени и интервалами, являются:

- найти разность между двумя датами, чтобы вычислить интервал времени между ними;
- прибавить к некой дате или интервалу или вычесть из некой даты или интервала какой-нибудь интервал или дату, чтобы вычислить дату в будущем или прошлом;
- сложить два произвольных интервала или найти разность между двумя произвольными интервалами, чтобы вычислить новый интервал;

- умножить или разделить произвольный интервал на произвольное число, чтобы вычислить некий новый интервал.

Легко заметить, что некоторые операции над интервалами и данными даты и времени не определены. Например, сложение двух дат не имеет никакого смысла.

В табл. 5.3 перечислены допустимые с точки зрения стандартного SQL операторы, которые могут работать с интервалами и данными даты и времени. Врезка в данном разделе объясняет, как и почему вы можете применять один и тот же оператор, чтобы выполнять принципиально разные операции.

Рассмотрим еще одну функцию SQL — `EXTRACT()`. Она выделяет конкретное поле значения даты и времени или интервала и на выходе выдает его в виде числа. Эту функцию обычно применяют в выражениях сравнения или/и при форматировании результатов запросов.

Выделение части произвольного значения даты и времени или интервала

Напечатайте

```
EXTRACT(field FROM
datetime_or_interval).
```

Предполагается, что:

- вместо `datetime_or_interval` вы подставите явно или укажете на то значение даты и времени либо на интервал, из которого хотите выделить часть, при том что `datetime_or_interval` может быть:
 - выражением даты и времени или интервальным выражением, например: заголовок некоего столбца, который должен содержать значения даты и времени или интервалы;

Переопределение операторов

Напомним, что операторы `+`, `-`, `*`, `/` обычно используются для операций с числами. В некоторых СУБД компании Microsoft оператор `+` применяется еще и для конкатенации строк. *Перегрузкой* произвольного заданного оператора называется такое его состояние, когда он должен выполнять более одной функции в зависимости от задаваемых условий. Чаще всего этими условиями для *перегружаемых* операторов являются их операнды. Так вот, каждый из операторов `+`, `-`, `*` и `/` даже в рамках стандартного SQL ведет себя по-разному, в зависимости от того, являются ли его операндами числа или интервалы и/или значения даты и времени. Имейте в виду, что кроме стандартного SQL перегружать не только операторы, но и функции может ваша собственная СУБД. Здесь под перегрузкой функций понимается назначение произвольной функции более одной задачи в зависимости от ряда привносимых условий. Чаще всего этими условиями являются типы данных аргументов этой функции. Такая функция называется *перегружаемой*. Примером перегружаемой функции служит `CONCAT()` СУБД MySQL, которая в качестве аргументов принимает как строки, так и нестроки. `CONCAT()` вынуждена дополнительно конвертировать нестроки в строки, чего она не делает, когда ее аргументами являются только строки (см. примечание с пометкой **DBMS** в разделе «Объединение строк с помощью оператора `||`» этой главы).

Листинг 5.24. Перечислить идентификаторы и даты публикации книг, учитываемых в нашей типовой базе данных, которые были напечатаны или в первой половине 2002 года, или в первой половине 2001 года. Перечень должен быть упорядочен по возрастанию срока, прошедшего с момента публикации (нисходящий порядок). Результат исполнения запроса см. на рис. 5.24

Листинг

```
SELECT
    title_id,
    pubdate
FROM titles
WHERE EXTRACT(YEAR FROM pubdate)
      BETWEEN 2001 AND 2002
      AND EXTRACT(MONTH FROM pubdate)
          BETWEEN 1 AND 6
ORDER BY pubdate DESC;
```

title_id	pubdate
-----	-----
T09	2002-05-31
T08	2001-06-01
T05	2001-01-01

Рис. 5.24. Результат исполнения запроса, представленного в листинге 5.24

- литералом даты и времени или интервальным литералом;
- результатом какой-нибудь операции или функции, на выходе которых должны быть значения даты и времени или интервалы;

■ вместо *field* вы подставите ту часть *datetime_or_interval*, которую хотите выделить, при том что *field* может быть (см. табл. 3.11) одним из следующих полей:

- YEAR, MONTH, DAY, HOUR, MINUTE, SECOND;
- TIMEZONE_HOUR, TIMEZONE_MINUTE.

Кроме того, если *field* является полем SECOND, функция EXTRACT() на выходе выдаст значение типа NUMERIC, а во всех остальных случаях – значение типа INTEGER (см. листинг 5.24 и рис. 5.24).



Вы можете применять функцию EXTRACT() везде, где используется какое-либо выражение, в частности в предложениях SELECT, WHERE, ORDER BY.



Если какой-либо операнд оператора или аргумент функции EXTRACT() является значением null, результат соответствующей операции или функции тоже будет значением null.



В СУБД Microsoft Access и Microsoft SQL Server функцией выделения является `DATEPART(datepart, date)`. Поэтому, для выполнения в среде этих СУБД запроса из листинга 5.24 в соответствующие выражения с функциями выделения придется внести следующие изменения:

- `DATEPART("yyyy", pubdate);`
- `DATEPART("m", pubdate).`

СУБД Oracle, MySQL и PostgreSQL могут принимать в качестве аргумента `field` функции `EXTRACT()` помимо указанных выше значений этого аргумента еще и другие типы данных.

Коммерческие СУБД предоставляют в ваше распоряжение функции, которые прибавляют интервалы к датам, например:

- `DATEDIFF()` – для Microsoft Access и Microsoft SQL Server;
- `ADD_MONTHS()` – для Oracle;
- `DATE_ADD()` и `DATE_SUB()` – для MySQL.

Сложная арифметика с данными даты и времени и с интервалами вполне обычное дело в программировании на SQL, и все коммерческие СУБД предоставляют в распоряжение программиста, кроме стандартных инструментов SQL, дополнительные функции, работающие на интервалах и датах. В этой связи имеет смысл организовать поиск в документации по вашей СУБД с использованием ключей «date and time functions» и «datetime functions» (то есть функции даты и времени).

Листинг 5.25. Распечатать текущие дату, время, отметку времени. Результат исполнения запроса см. на рис. 5.25

```

SELECT
    CURRENT_DATE AS "Date",
    CURRENT_TIME AS "Time",
    CURRENT_TIMESTAMP AS "Timestamp";

```

Date	Time	Timestamp
2002-05-19	21:02:04	2002-05-19 21:02:04

Рис. 5.25. Результат исполнения запроса, представленного в листинге 5.25

Листинг 5.26. Перечислить книги, учитываемые в нашей типовой базе данных, дата публикации которых или неизвестна совсем, или попадает в интервал 90 дней, считая назад от текущей даты (та дата, которая в данном запросе считается «текущей», получена в запросе из листинга 5.25). Перечень должен быть упорядочен по удалению даты публикации книги от текущей даты (нисходящий порядок). Результат исполнения запроса см. на рис. 5.26

```

SELECT title_id, pubdate
FROM titles
WHERE pubdate
    BETWEEN CURRENT_TIMESTAMP
        - INTERVAL 90 DAY
    AND CURRENT_TIMESTAMP
        + INTERVAL 90 DAY
    OR pubdate IS NULL
ORDER BY pubdate DESC;

```

title_id	pubdate
T09	2002-05-31
T10	NULL

Рис. 5.26. Результат исполнения запроса, представленного в листинге 5.26

Извлечение значений текущих даты и времени

Чтобы извлечь текущие дату и время из системных часов компьютера, на котором работает конкретная СУБД, применяют функции `CURRENT_DATE`, `CURRENT_TIME`, `CURRENT_TIMESTAMP`.

Порядок извлечения текущих даты и времени

Напечатайте:

- `CURRENT_DATE` — чтобы вывести дату;
- `CURRENT_TIME` — чтобы вывести время;
- `CURRENT_TIMESTAMP` — чтобы вывести отметку времени.

На выходе функции в первом случае будет значение, относящееся к типу данных `DATE`, во втором случае — к типу данных `TIME`, а в третьем — к типу данных `TIMESTAMP` (см. раздел «Типы данных» в главе 3, а также листинги и рис. 5.25 и 5.26).

П Вы можете применять функции даты и времени везде, где используется какое-либо выражение, в частности в предложениях `SELECT`, `WHERE`, `ORDER BY`.

П Каждая из функций `CURRENT_TIME()` и `CURRENT_TIMESTAMP()` принимает в качестве аргумента некую *точность*, обозначающую количество десятичных разрядов в записи дробной части секунды, которое следует, по вашему мнению, включить в значение времени. Например, функция `CURRENT_TIME(6)` даст на выходе такое значение текущего времени, что в поле `SECOND` доли секунды будут представлены *точностью* равной 6. Подробнее о смысле и практическом применении понятия *точности* читайте в разделе «Типы данных» главы 3.



В СУБД Microsoft Access есть три системные функции даты/времени: `Date()`, `Time()` и `Now()`. Чтобы выполнить в среде этой СУБД запросы из листингов 5.25 и 5.26, вам следует:

- внести изменения в текст выражений даты и времени запроса, представленного в листинге 5.25:

- заменить `CURRENT_DATE` AS "Date" на `DATE()` AS "Date";
- заменить `CURRENT_TIME` AS "Time" на `Time()` AS "Time";
- заменить `CURRENT_TIMESTAMP` AS "Timestamp" на `Now()` AS "Timestamp";

- внести изменения в текст предложения `BETWEEN` этого запроса, представленного в листинге 5.26:

```
BETWEEN NOW() - 90
      AND NOW() + 90.
```

В СУБД Microsoft SQL Server системной функцией даты/времени является `CURRENT_TIMESTAMP` (или ее синоним – функция `GETDATE()`). Ни функция `CURRENT_DATE()`, ни функция `CURRENT_TIME()` данной СУБД не поддерживаются. Чтобы выполнить запрос из листинга 5.25 в среде СУБД Microsoft SQL Server, вам придется убрать из текста запроса выражения с функциями `CURRENT_DATE()` и `CURRENT_TIME()`. А для выполнения запроса из листинга 5.26 в текст предложения `BETWEEN` этого запроса вам придется внести следующие изменения:

```
BETWEEN CURRENT_TIMESTAMP - 90
      AND CURRENT_TIMESTAMP + 90.
```

В СУБД Oracle системной функцией даты/времени является `SYSDATE`. И хотя Oracle 9i и более поздних версий поддерживает функции `CURRENT_TIMESTAMP()` и

`CURRENT_DATE()`, функция `CURRENT_TIMESTAMP()` в этих версиях все равно не поддерживается. Кроме того, реализация запроса, представленного в листинге 5.25, в среде СУБД Oracle требует предложения `FROM DUAL` (см. примечание **DBMS** в разделе «Создание производных столбцов» этой главы). Таким образом, чтобы запустить запрос, который представлен в листинге 5.25, в среде СУБД Oracle, вам придется изменить предложение `SELECT` следующим образом:

```
SELECT SYSDATE AS "Date"
      FROM DUAL;
```

Однако это еще не все. Дело в том, что функция `SYSDATE` выводит системные дату и время, но не отображает это время, если только не получит соответствующего указания от отформатированной функции `TO_CHAR()`, например:

```
SELECT TO_CHAR(SYSDATE,
→ 'YYYY-MM-DD HH24:MI:SS')
      FROM DUAL;
```

Чтобы в среде СУБД Oracle выполнить запрос из листинга 5.26, в текст предложения `BETWEEN` этого запроса придется внести следующие изменения:

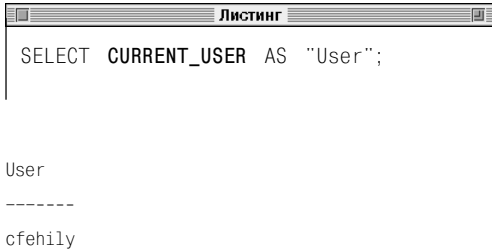
```
BETWEEN SYSDATE - 90
      AND SYSDATE + 90
```

Чтобы в среде СУБД PostgreSQL выполнить запрос из листинга 5.26, в текст предложения `BETWEEN` этого запроса вам придется внести следующие изменения:

```
BETWEEN CURRENT_TIMESTAMP - 90
      AND CURRENT_TIMESTAMP + 90
```

Для получения подробной информации о системных функциях даты и времени вашей СУБД организуйте поиск в ее документации по ключам «date and time functions» (функции даты и времени) и «system functions» (системные функции).

Листинг 5.27. Напечатать имя текущего пользователя



```
SELECT CURRENT_USER AS "User";
```

User

cfehily

Рис. 5.27. Результат выполнения листинга 5.27

Отображение информации о пользователе

Чтобы идентифицировать пользователя, действующего в системе данного базового сервера СУБД, применяют функцию `CURRENT_USER`.

Вывод информации о текущем пользователе

Напечатайте `CURRENT_USER` (см. листинг и рис. 5.27; эта функция выдает на выходе имя текущего пользователя базы данных).

П

Вы можете применять функции получения имени пользователя везде, где присутствует какое-либо выражение, в частности в предложениях `SELECT`, `WHERE`, `ORDER BY`.

П

Кроме функции `CURRENT_USER`, стандарт SQL определяет функции получения имени пользователя `SESSION_USER` и `SYSTEM_USER`. Различия между ними следующие: во-первых, каждая функция понимает под пользователем не тот или не совсем тот объект, по сравнению с остальными двумя функциями; во-вторых, каждая функция подразумевает свой собственный круг полномочий. Так, функция `CURRENT_USER` указывает на регистрационный идентификатор пользователя базы данных, под чьим набором полномочий в текущий момент исполняются запросы (например, этот пользователь может иметь право в рамках своих полномочий запускать только, допустим, команды `SELECT`, и никакие другие). В то же время функция `SESSION_USER` укажет на идентификатор полномочий текущего сеанса работы с базой данных (и эти полномочия могут не совпадать с полномочиями текущего пользователя). Наконец, функция `SYSTEM_USER` укажет на регистрационный номер (учетную запись) пользователя в базовой операционной системе. Очевидно, что эти три пользовательских значения существенно зависят от выбора конкретной СУБД и могут не совпадать. За получением подробной информации о пользователях, сессиях (сеансах) и полномочиях обратитесь к документации по вашей СУБД и организуйте в ней поиск по ключам «authorization», «session», «user», «role» (то есть «полномочия», «сеанс», «пользователь» и «роль»).



Чтобы в среде СУБД Microsoft Access выполнить запрос из листинга 5.27, вам придется изменить текст предложения `SELECT` следующим образом:

```
SELECT CureentUser AS "User";
```

Чтобы в среде СУБД Oracle выполнить запрос, который представлен в листинге 5.27, вам придется изменить текст предложения `SELECT` следующим образом:

```
SELECT USER AS "User" FROM DUAL;
```

Чтобы в среде СУБД MySQL выполнить запрос, который представлен в листинге 5.27, вам придется изменить текст предложения `SELECT` следующим образом:

```
SELECT USER() AS "User";
```

Кроме уже рассмотренной пользовательской функции СУБД Microsoft SQL Server поддерживает `SESSION_USER` и `SYSTEM_USER`.

Кроме уже рассмотренной пользовательской функции СУБД MySQL поддерживает `SESSION_USER()` и `SYSTEM_USER()`.

Еще одна функция СУБД Oracle – `SYS_CONTEXT()` – дает на выходе пользовательские атрибуты сеанса.

СУБД PostgreSQL поддерживает `SESSION_USER`.

За получением подробной информации о системных функциях обратитесь к документации по вашей СУБД и организуйте в ней поиск по ключам «user» или «system function» («пользователь» или «системная функция»).

Преобразование типов данных с помощью функции CAST()

В большинстве случаев ваша СУБД автоматически произведет преобразование (или, как еще говорят, *приведение*) типов данных. Тем самым СУБД позволит, не задумываясь о типах данных, применять, скажем, одновременно числа и даты в символьных выражениях типа конкатенации или, допустим, без проблем использовать разнотипные числа в смешанных арифметических выражениях, потому что все эти числа будут преобразованы в один «старший» или самый сложный тип данных (подробнее об этом см. в примечании с пометкой **DBMS** в разделе «Арифметические операции» этой главы). Но иногда ваша СУБД не будет производить преобразование типов данных автоматически. В этом случае вам следует воспользоваться функцией CAST() и принудительно конвертировать некое выражение из одного типа данных в другой (см. раздел «Типы данных» в главе 3).

Основные характеристики функции CAST() таковы:

- все преобразования типов данных, которые проводит ваша СУБД, делятся на две категории (иногда конверсия типов данных запрещена, например: тип FLOAT нельзя преобразовать в тип TIMESTAMP):
 - *неявные*, или автоматические, происходят без участия функции CAST(), которую в этом случае указывать не надо;
 - *явные*, для совершения которых необходимо указать функцию CAST();

- тип данных, над которым проводится преобразование, называют *исходным*, а тип данных, в который данные переходят в результате преобразования, – *целевым*;
- произвольный числовой тип и тип даты и времени можно преобразовать в любой символьный тип данных;
- вы можете преобразовать любой исходный символьный тип данных в другой тип данных, но только если соответствующая строка символов представляет собой в целевом типе данных какое-нибудь незапрещенное буквенное выражение;
- некоторые преобразования числовых типов в зависимости от конкретной СУБД или округляют, или усекают данные, как это происходит при конверсии из типа DECIMAL в тип INTEGER;
- преобразование из типа VARCHAR в тип CHAR может вызвать усеечение строк;
- иногда преобразование типов может сгенерировать ошибку, если целевой тип не имеет достаточно места для того, чтобы вместить преобразованное значение (например, преобразование из типа FLOAT в тип SMALLINT даст сбой, если преобразуемое число с плавающей десятичной точкой не попадает в интервал, который СУБД отводит для чисел типа SMALLINT, а именно: от –32 768 до +32 768);
- любое преобразование из типа NUMERIC в тип DECIMAL может потребовать явного указания функции CAST() из-за опасности потери точности, которая могла бы произойти при неявном преобразовании;

Листинг 5.28. Этот запрос должен преобразовать цены книг, учитываемых в нашей типовой базе данных, из типа данных DECIMAL в типы данных INTEGER и CHAR(8). При этом символы < и > используются для того, чтобы показать границы строк типа CHAR(8). В результате вы получите или то, что показано на рис. 5.28а, или то, что показано на рис. 5.28б, в зависимости от того, округляет или усекает ваша СУБД целые числа

```

SELECT
    price
    AS "price(DECIMAL)",
    CAST(price AS INTEGER)
    AS "price(INTEGER)",
    '<' || CAST(price AS CHAR(8)) || '>'
    AS "price(CHAR(8))"
FROM titles;

```

price(DECIMAL)	price(INTEGER)	price(CHAR(8))
21.99	21	<21.99 >
19.95	19	<19.95 >
39.95	39	<39.95 >
12.99	12	<12.99 >
6.95	6	<6.95 >
19.95	19	<19.95 >
23.95	23	<23.95 >
10.00	10	<10.00 >
13.95	13	<13.95 >
NULL	NULL	NULL
7.99	7	<7.99 >
12.99	12	<12.99 >
29.99	29	<29.99 >

Рис. 5.28а. Результат исполнения запроса, представленного в листинге 5.28, в случае, если СУБД *усекает* десятичные числа (то есть, строго говоря, Exact Numeric, целые и рациональные числа с фиксированной десятичной точкой) при конвертации их в целые

```
price(DECIMAL)price(INTEGER)price(CHAR(8))
```

price(DECIMAL)	price(INTEGER)	price(CHAR(8))
21.99	22	<21.99 >
19.95	20	<19.95 >
39.95	40	<39.95 >
12.99	13	<12.99 >
6.95	7	<6.95 >
19.95	20	<19.95 >
23.95	24	<23.95 >
10.00	10	<10.00 >
13.95	14	<13.95 >
NULL	NULL	NULL
7.99	8	<7.99 >
12.99	13	<12.99 >
29.99	30	<29.99 >

Рис. 5.286. Результат исполнения запроса, представленного в листинге 5.28, когда СУБД округляет десятичные числа при конвертации их в целые

- при любом преобразовании из типа DATE в тип TIMESTAMP часть результата этого преобразования, которая соответствует времени, будет равна 00:00:00 (это полночь);
- если какой-либо аргумент функции CAST() является значением null, ее результат тоже будет значением null (исключение составляет СУБД Oracle, см. далее в этом разделе примечание с пометкой DBMS).

Преобразование значений одного типа данных к другому

Напечатайте CAST(*expr AS data_type*).

Предполагается, что (см. листинги 5.28, 5.29 и рис. 5.28а, 5.28б, 5.29):

- вместо *expr* вы подставите выражение, которое хотите преобразовать в другой тип данных;
- вместо *data_type* вы подставите целевой тип данных, имея в виду следующее: тип *data_type* должен быть одним из типов данных, представленных в табл. 3.5, 3.6, 3.7, 3.9, 3.10, 3.12, которые могут при необходимости включать аргументы длины, точности и масштаба (множество допустимых значений *data_type* включает, например, CHAR(10), VARCHAR(25), NUMERIC(5, 2), INTEGER, FLOAT, DATE);
- если тип данных, обозначенный как *expr*, окажется несовместимым с типом данных *data_type*, СУБД выдаст ошибку.



Функцию CAST() можно применять там, где используется какое-либо выражение, в частности в предложениях SELECT, WHERE, ORDER BY.

П

Расширяющим называется любое преобразование одного произвольного типа данных в другой тип данных при условии, что потери данных или их искажения не произойдет. Например, преобразование из типа данных `SMALLINT` в тип данных `INTEGER` является расширяющим, так как по своему формату тип `INTEGER` способен вместить любое число, представленное типом `SMALLINT`. Преобразование данных, не являющееся расширяющим, называется *сужающим*. Оно несет в себе риск потери или/и искажения данных. Примером сужающего преобразования может служить преобразование, противоположное только что рассмотренному расширяющему преобразованию. Преобразование из типа данных `INTEGER` в тип данных `SMALLINT` может привести к тому, что некоторые числа, представленные значениями типа данных `INTEGER`, просто не поместятся в формате типа данных `SMALLINT`, следовательно, будут искажены. Все расширяющие преобразования разрешены во всех СУБД, но попытка выполнить какое-нибудь сужающее преобразование может вынудить СУБД выдать предупреждение о возможной ошибке или сообщение об ошибке.



СУБД Microsoft Access вместо одной функции преобразования типов `CAST()` включает целое семейство аналогичных функций. Например, функции `CStr(expr)`, `CInt(expr)` и `CDec(expr)` приведут произвольное выражение, представленное как *expr*, к некоей строке, некоему целому числу или некоему числу с фиксированной десятичной точкой. Кроме того, в этой среде вы можете применять следующие функции:

- `Space(number)` – чтобы добавить в строку пробелы в числе, обозначенном как *number*;
- `Left(string, length)` – чтобы усечь строку *string* до длины *length*.

Учитывая все вышесказанное и то, что оператором конкатенации строк в СУБД Microsoft Access является оператор `+`, при запуске в среде СУБД Microsoft Access запросов из листингов 5.28 и 5.29 вам придется внести в выражения следующие изменения:

- `CInt(price)`
`'<' + CStr(price) + '>'` – для листинга 5.28;
- `CStr(sales)`
`→ + Space(8 - Len(CStr(sales)))`
`→ + ' copies sold of '`
`→ + Left(title_name, 20)` – для листинга 5.29.

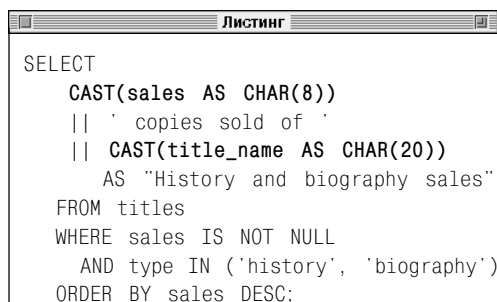
В СУБД Microsoft SQL Server для конкатенации строк следует применять оператор `+`. Поэтому, чтобы запустить в среде этой СУБД запросы из листингов 5.28 и 5.29, вам придется внести соответствующие изменения в текст предложений с оператора-ми конкатенации:

- `'<' + CAST(price AS CHAR(8)) + '>'`
 – для листинга 5.28;
- `CAST(sales AS CHAR(8))`
`→ + ' copies sold of '`
`→ + CAST(title_name AS CHAR(20))` – для листинга 5.29.

СУБД Oracle не разрешает преобразовывать символьную строку произвольного типа в тип `CHAR(length)`, если длина целого типа, указанная как *length*, меньше длины исходной строки. То есть в среде СУБД Oracle не удастся применить функцию `CAST()` для усечения строк. С этой целью воспользуйтесь функцией `SUBSTR()`. О том, как это делается, прочитайте в примечании **DBMS** в разделе «Выбор произвольной подстроки с помощью функции `SUBSTRING()`» этой главы.

Таким образом, если в среде СУБД Oracle запустить запрос из листинга 5.29, вам

Листинг 5.29. Перечислить в порядке убывания (нисходящий порядок) общие объемы продаж вместе с некоторыми частями названий заданной длины тех книг, учитываемых в нашей типовой базе данных, темой которых является или история, или биография. Здесь преобразование в тип CHAR(20) укорачивает эти названия до разумных пределов, делая итоговый перечень более читабельным. Результат исполнения запроса см. на рис. 5.29



```

SELECT
    CAST(sales AS CHAR(8))
    || ' copies sold of '
    || CAST(title_name AS CHAR(20))
        AS "History and biography sales"
FROM titles
WHERE sales IS NOT NULL
    AND type IN ('history', 'biography')
ORDER BY sales DESC;

```

History and biography sales

```

-----
1500200 copies sold of I Blame My Mother
100001  copies sold of Spontaneous, Not Ann
11320   copies sold of How About Never?
10467   copies sold of What Are The Civilia
9566    copies sold of 200 Years of German
566     copies sold of 1977!

```

Рис. 5.29. Результат исполнения запроса, представленного в листинге 5.29

придется внести соответствующие изменения в текст выражения с функцией CAST():

```

CAST(sales AS CHAR(8))
→ || ' copies sold of '
→ || SUBSTR(title_name, 1, 20)

```

В среде СУБД MySQL множество допустимых целевых типов данных функции CAST() ограничено бинарными, целочисленными типами данных и типами данных даты и времени (только при указании целевого типа данных вместо типа INTEGER следует ставить SIGNED). Также в среде MySQL вы можете применять следующие функции:

- RPAD(*string*, *length*, *padstring*) – чтобы добавлять пробелы в строки;
- LEFT(*string*, *length*) – чтобы усекал строки.

Наконец, в среде СУБД MySQL для конкатенации строк следует применять функцию CONCAT().

Чтобы в среде СУБД MySQL выполнить запросы из листингов 5.28 и 5.29, вам придется внести соответствующие изменения в выражения с функцией CAST():

- CAST(*price* AS SIGNED)
CONCAT('<', RPAD(*price*, 8, ' '),
→ '>') – для листинга 5.28;
- CONCAT(
→ RPAD(*sales*, 8, ' ')
→ ' copies sold of ',
→ LEFT(*title_name*, 20)) – для листинга 5.29.

В среде СУБД PostgreSQL для усекания строк следует применять функцию SUBSTRING(), а для преобразования произвольного числа в строку – функцию TO_CHAR(*number*, *format*).

Если в среде СУБД PostgreSQL вы хотите запустить запросы, представленные в листингах 5.28 и 5.29, то в выражения с функцией `CAST()` придется внести соответствующие изменения:

- `CAST(price AS INTEGER)`
`'<' || TO_CHAR(price, '99.99 ')`
`||`
→ `'>'` – для листинга 5.28;
- `SUBSTRING(title_name FROM 1`
→ `FOR 20)` – для листинга 5.29.

СУБД Oracle трактует любую пустую строку как значение `null`. Поэтому `CAST(NULL AS CHAR)` дает на выходе `''` (пустая строка). Обратите внимание на примечание с пометкой **DBMS** в разделе «Значение null» главы 3.

Чтобы в среде СУБД PostgreSQL сравнить любое значение столбца, которому присвоен тип данных `NUMERIC` или `DECIMAL` (то есть целые и рациональные числа с фиксированной десятичной точкой), с любым действительным числом (то есть с числом, которое в программировании принято называть действительным, а на самом деле — с целым или рациональным числом с плавающей точкой, так как действительные числа есть абстракция), необходимо конвертировать это число или в тип `NUMERIC`,

или в тип `DECIMAL`. Например, следующая команда в среде СУБД PostgreSQL не работает, так как столбец `price` относится к типу `DECIMAL(5,2)`:

```
SELECT price
FROM titles
WHERE price < 20.00;
```

Обозначенную проблему можно решить с помощью такого запроса:

```
SELECT price
FROM titles
WHERE price < CAST(20.00 AS
DECIMAL);
```

Кроме рассмотренных функций коммерческие СУБД включают и другие функции преобразования и форматирования:

- `CONVERT()` – в СУБД Microsoft SQL Server и MySQL;
- `TO_CHAR()`, `TO_DATE()`, `TO_TIMESTAMP()` и `TO_NUMBER()` – в СУБД Oracle и PostgreSQL.

Чтобы полностью разобраться в этом вопросе, организуйте поиск в документации по вашей СУБД с использованием ключей «conversion», «cast» или «formatting functions» (преобразование, приведение или функции форматирования).

Вычисление условных значений с помощью выражения CASE

До появления стандарта SQL-92 программисты часто жаловались на то, что недостаток условных конструкций в SQL вынуждает обращаться к базовому языку всякий раз, когда возникает необходимость предпринять некие действия на основе истинностного значения условия (то есть когда надо принять во внимание все значения условия, а именно: `true`, `false`, `unknown`, что означает «истина», «ложь», «неизвестно»). В качестве ответа на это замечание в стандарт SQL-92 были введены выражение `CASE` и его упрощенные эквиваленты `COALESCE()` и `NULLIF()`. В этом разделе мы поговорим о функции `CASE()`, а все остальные аналогичные конструкции будут рассмотрены в конце главы.

Основные характеристики выражения `CASE`:

- если вы когда-либо программировали, то заметите, конечно, что выражение `CASE` представляет собой SQL-эквивалент командных конструкций `if-then-else` или `switch`, активно применяемых в процедурных языках. Разница лишь в том, что `CASE` — это выражение, а не команда;
- по сути, выражение `CASE` вычисляет несколько логических условий подряд до появления первого значения `true` (истина) и выдает одно какое-нибудь значение, соответствующее этому условию, оказавшемуся истинным;
- выражение `CASE` позволяет, не внося никаких фактических изменений в данные таблиц произвольной базы данных,

отобразить некое значение, являющееся альтернативой фактическому значению, которое содержится в произвольно выбранном столбце какой-либо таблицы;

- основное предназначение выражения `CASE` состоит в том, чтобы заменять коды или аббревиатуры (любимые программистами и разработчиками баз данных за то, что их легче хранить и ими легче управлять по сравнению с пространными текстовыми объяснениями) более читабельными значениями (например, если столбец `marital_status` содержит целочисленные коды 1, 2, 3, 4 состояний «холост», «женат», «разведен» и «вдовец», то непрофессионалы в программировании баз данных, кто будет работать с вашей базой, наверное, предпочтут воспользоваться пояснениями, а не кодами, похожими на криптограмму);
- выражение `CASE` имеет два формата:
 - в *простом* формате сравнивает произвольно заданное выражение с простыми выражениями из какого-либо перечня и выдает общий результат сравнений;
 - в *поисковом* формате вычисляет логические выражения определенного набора логических (Булевых) выражений и по результатам этих вычислений выдает общий результат;
- если ни одно из проверяемых условий не окажется истинным (то есть значение `true` так и не появится), в качестве результата по умолчанию выражение `CASE` выдаст необязательное предложение `ELSE`.

Использование выражения CASE в простом формате

Напечатайте:

```
CASE comparison_value
  WHEN value1 THEN result1
  WHEN value2 THEN result2
  ...
  WHEN valueN THEN resultN
  [ELSE default_result]
END
```

Предполагается, что:

- вместо *value1, value2, ..., valueN* вы подставите выражения, не содержащие оператор сравнения;
- вместо *result1, result2, ..., resultN* вы подставите выражения, являющиеся кандидатами на результат всего выражения CASE в простом формате;
- вместо выражения *comparison_value* подставите (явно или через указание на заголовок столбца некой таблицы) выражение, которому должно соответствовать каждое выражение из группы *value1, value2, ..., valueN*, чтобы соответствующий кандидат стал результатом;
- вместо *default_result* вы подставите выражение, которое по вашему решению быть должно результатом выражения CASE в том случае, когда ни одно значение из группы *value1, value2, ..., valueN* не будет соответствовать значению *comparison_value* (если опция *ELSE default_result* будет опущена, СУБД выберет опцию *ELSE NULL*);
- вы выберете все выражения, участвующие в выражении CASE в простом формате, чтобы они обязательно были или одного и того же типа данных или таких типов данных, которые можно неявно преобразовать в один и тот же тип данных.

Выражение CASE в простом формате сравнивает каждое выражение из группы *value1, value2, ..., valueN* с выражением *comparison_value*: сначала *value1*, потом *value2...* и т.д. до тех пор, пока не будет достигнуто соответствие (то есть то тех пор, пока в результате сравнения не будет получено логическое значение *truth*). Когда же соответствие будет достигнуто на некоем условии *valuea* ($1 < a < N$), выражение CASE выдаст в качестве результата значение *resulta*. Если весь набор значений *value1, value2, ..., valueN* будет просмотрен, а соответствие ни на одном значении достигнуто не будет, выражение CASE выдаст в качестве результата *default_result* (или значение *null*; см. листинг 5.30 и рис. 5.30).

Использование выражения CASE в поисковом формате

Напечатайте:

```
CASE
  WHEN condition1 THEN result1
  WHEN condition2 THEN result2
  ...
  WHEN conditionN THEN resultN
  [ELSE default_result]
END
```

Предполагается, что:

- вы замените *condition1, condition2, ..., conditionN* поисковыми условиями, каждое из которых может содержать операторы сравнения и несколько логических выражений, связанных операторами AND и OR, и принимать значения *true, false, unknown* (см. раздел «Фильтрация строк с помощью предложения WHERE» в главе 4);
- вместо *result1, result2, ..., resultN* вы подставите выражения, являющиеся кандидатами на результат всего выражения CASE в поисковом формате;

Листинг 5.30. Перечислить в алфавитном порядке (восходящий порядок) идентификаторы и темы книг, учитываемых в нашей типовой базе данных, вместе с текущими и новыми ценами при том условии, что новая цена исторических книг должна быть больше текущей цены этих книг на 10%, новая цена книг по психологии должна быть больше текущей цены этих книг на 20%, а новая цена любой книги по другой теме должна быть равна ее текущей цене. Результат исполнения запроса см. на рис. 5.30

```

SELECT
    title_id,
    type,
    price,
    CASE type
        WHEN 'history'
            THEN price * 1.10
        WHEN 'psychology'
            THEN price * 1.20
        ELSE price
    END
    AS "New price"
FROM titles
ORDER BY type ASC, title_id ASC;
```

title_id	type	price	New price
-----	-----	-----	-----
T06	biography	19.95	19.95
T07	biography	23.95	23.95
T10	biography	NULL	NULL
T12	biography	12.99	12.99
T08	children	10.00	10.00
T09	children	13.95	13.95
T03	computer	39.95	39.95
T01	history	21.99	24.19
T02	history	19.95	21.95
T13	history	29.99	32.99
T04	psychology	12.99	15.59
T05	psychology	6.95	8.34
T11	psychology	7.99	9.59

Рис. 5.30. Результат исполнения запроса, представленного в листинге 5.30

- вместо *default_result* вы подставите, если захотите, то выражение, которое по вашему решению должно быть результатом выражения CASE в поисковом формате, если ни одно условие из группы *condition1*, *condition2*, ..., *conditionN* не даст значения true (если опция *ELSE default_result* будет опущена, СУБД будет считать выбранной опцию *ELSE NULL*);
- вы выберете все выражения, участвующие в выражении CASE в поисковом формате, чтобы они обязательно были одного и того же типа данных или таких типов данных, которые могут быть неявно преобразованы в один и тот же тип данных.

Выражение CASE в поисковом формате вычисляет каждое из логических выражений *condition1*, *condition2*, ..., *conditionN*: сначала *condition1*, потом *condition2*... и т.д. до тех пор, пока в результате не будет получено логическое значение truth. Когда же это значение будет достигнуто на некотором условии *conditiona*, при котором $1 < a < N$, выражение CASE выдаст в качестве результата значение *resulta*. Если весь набор условий *condition1*, *condition2*, ..., *conditionN* будет просмотрен, а значение truth достигнуто не будет, выражение CASE выдаст в качестве результата *default_result* (или значение null; см. листинг 5.31 и рис. 5.31).



Имейте в виду, что в зависимости от СУБД выражение CASE в поисковом формате может по достижении первого результата как продолжить работу и рассмотреть все остальные предложения WHEN и соответствующие условия, так и остановиться. В связи с этим вам следует предотвратить нежелательные побочные эффекты типа деления на ноль при вычислении выражений, до которых дело не должно было дойти, как вам могло показаться, исходя из предположения, что выражение CASE работает до первого результата.

П

Выражение CASE можно применять там, где используется какое-либо выражение, в частности в предложениях SELECT, WHERE, ORDER BY.

П

Выражение CASE поможет избежать ошибок, связанных с делением на ноль:

```
CASE
  WHEN n <> 0 THEN expr/n
  ELSE 0
END
```

П

Выражение CASE в простом формате можно представлять как упрощенную запись следующего выражения CASE в поисковом формате:

```
CASE
  WHEN comparison_value = value1
  THEN result1
  WHEN comparison_value = value2
  THEN result2
...
  WHEN comparison_value = valueN
  THEN resultN
  [ELSE default_result]
END
```



СУБД Microsoft Access не поддерживает выражение CASE, поэтому для запуска в этой среде запросов, представленных в листингах 5.30 и 5.31, вам придется вместо выражения CASE использовать функцию Switch(*condition1*, *result1*, *condition2*, *result2*, ..., *conditionN*, *resultN*) и, соответственно, заменить в текстах указанных запросов выражения CASE следующими:

```
■ Switch(
  → type IS NULL, NULL,
  → type = 'history', price * 1.10,
  → type = 'psychology', price * 1.20,
  → type IN ('biography',
  → type = 'children', 'computer'),
  price) — для листинга 5.30;
```

```
■ Switch(
  → sales IS NULL,
  → 'Unknown' ,
  → sales <= 1000,
  → 'Not more than 1,000',
  → sales <= 10000,
  → 'Between 1,001 and 10,000',
  → sales <= 100000,
  → 'Between 1,001 and 100,000',
  → sales <= 1000000,
  → 'Between 1,001 and 1,000,000',
  → sales > 1000000,
  → 'Over 1,000,000') — для листинга 5.31.
```

Листинг 5.31. Перечислить в порядке возрастания объемов продаж идентификаторы книг, учитываемых в нашей базе данных, с разбивкой по категориям объемов продаж

```
SELECT
  title_id,
CASE
  WHEN sales IS NULL
  THEN 'Unknown'
  WHEN sales <= 1000
  THEN 'Not more than 1,000'
  WHEN sales <= 10000
  THEN 'Between 1,001 and 10,000'
  WHEN sales <= 100000
  THEN 'Between 10,001 and 100,000'
  WHEN sales <= 1000000
  THEN 'Between 100,001 and
1,000,000'
  ELSE 'Over 1,000,000'
END
AS "Sales category"
FROM titles
ORDER BY sales ASC;
```

title_id	Sales category
-----	-----
T10	Unknown
T01	Not more than 1,000
T08	Between 1,001 and 10,000
T09	Between 1,001 and 10,000
T02	Between 1,001 and 10,000
T13	Between 10,001 and 100,000
T06	Baetween 10,001 and 100,000
T04	Between 10,001 and 100,000
T03	Between 10,001 and 100,000
T11	Between 10,001 and 100,000
T12	Between 100,001 and 1,000,000
T05	Between 100,001 and 1,000,000
T07	Over 1,000,000

Рис. 5.31. Результат исполнения запроса, представленного в листинге 5.31

СУБД Oracle 9i поддерживает выражение CASE в простом формате, поэтому в среде Oracle 9i запрос из листинга 5.30 будет выполнен без проблем в том виде, как он есть. Но если вы работаете в среде СУБД Oracle 8i, то для запуска запроса из листинга 5.30 придется вносить в него изменения: или заменить выражение CASE в простом формате подходящим выражением CASE в поисковом формате (см. выше примечание в этом разделе), или использовать функцию DECODE (*comparison_value*, *value1*, *result1*, *value2*, *result2*, ..., *default_result*) следующим образом:

```
DECODE(type,  
→ NULL,          NULL,  
→ 'history',     price * 1.10,  
→ 'psychology',  price * 1.20,  
→ price)
```

Если вы работаете в среде СУБД PostgreSQL, то для исполнения запроса, представленного в листинге 5.30, придется преобразовать числа с плавающей точкой в тип данных DECIMAL. Подробнее о том, как выполнять подобные преобразования, читайте в разделе «Преобразование типов данных с помощью функции CAST()» этой главы. В текст выражений для вычисления новых цен, относящихся к рассматриваемому запросу, придется внести следующие изменения:

- price * CAST((1.10) AS DECIMAL);
- price * CAST((1.20) AS DECIMAL).

Проверка на значения null с использованием функции COALESCE()

Функция COALESCE() выдает первое выражение, не являющееся значением null, которое она встретит при просмотре своих аргументов слева направо. Функцию COALESCE() чаще всего применяют для того, чтобы в результатах запросов вместо значений null показывать какое-нибудь специальное обозначение. Дело в том, что некоторых пользователей значение null смущает. Интересно отметить, что функция COALESCE(*expr1*, *expr2*, *expr3*) – это не более чем упрощенная форма следующего выражения CASE в поисковом формате:

```
CASE
  WHEN expr1 IS NOT NULL THEN expr1
  WHEN expr2 IS NOT NULL THEN expr2
  ELSE expr3
END
```

Отображение первого найденного значения, не являющегося значением null

Напечатайте COALESCE(*expr1*, *expr2*,...).

Предполагается, что:

- вместо *expr1*, *expr2*,... вы подставите необходимое количество выражений, разделенных запятыми;
- вы выберете все выражения *expr1*, *expr2*,..., чтобы они обязательно были одного и того же типа данных или таких типов данных, которые можно неявно преобразовать в один и тот же тип данных.

Функция COALESCE() выдаст первое выражение, не являющееся значением null, которое она встретит при просмотре и вычислении своих аргументов в порядке следования их слева направо. Если все выражения *expr1*, *expr2*,... окажутся значениями null, функция COALESCE() тоже выдаст значение null (см. листинг 5.32 и рис. 5.32).



Функцию COALESCE() можно использовать там, где применяется какое-либо выражение, в частности в предложениях SELECT, WHERE, ORDER BY.



СУБД Microsoft Access не поддерживает функцию COALESCE(), поэтому для запуска в этой среде запроса из листинга 5.32 вместо функции COALESCE() придется использовать функцию Switch(*condition1*, *result1*, *condition2*, *result2*, ..., *conditionN*, *resultN*) и, соответственно, заменить в тексте указанного запроса функцию COALESCE() следующим выражением:

```
Switch(state IS NOT NULL, state,
→ state IS NULL, 'N/A')
```

СУБД Oracle 9i поддерживает функцию COALESCE(), поэтому в среде Oracle 9i запрос из листинга 5.32 будет выполнен без проблем в том виде, как он есть. Но если вы работаете в среде СУБД Oracle 8i, то для запуска запроса из листинга 5.32 придется вносить в него изменения: вместо функции COALESCE() применить функцию NVL(*expr1*, *expr2*), которая принимает только два аргумента (если надо заменить такое выражение, где у функции COALESCE() больше двух аргументов, в среде СУБД Oracle 8i следует использовать выражение CASE).

В запросе 5.32 можно применить функцию NVL(), например:

```
NVL(state, 'N/A')
```


Листинг 5.32. Распечатать идентификаторы и места нахождения издателей, учитываемых в нашей типовой базе данных, но так, чтобы при наличии у какого-нибудь издателя в столбце state (штат) значения null в распечатке вместо него стояло N/A. Результат исполнения запроса см. на рис. 5.32

```

SELECT
    pub_id,
    city,
    COALESCE(state, 'N/A') AS "state",
    country
FROM publishers;

```

pub_id	city	state	country
P01	New York	NY	USA
P02	San Francisco	CA	USA
P03	Hamburg	N/A	Germany
P04	Berkeley	CA	USA

Рис. 5.32. Результат исполнения запроса, представленного в листинге 5.32

Сравнение выражений с помощью функции NULLIF()

Функция NULLIF() сравнивает два выражения, являющиеся ее аргументами, и выдает на выходе:

- значение null, если эти выражения равны (то есть совпадают в каком-либо заранее определенном смысле);
- первое из этих двух выражений, если они не равны.

Самый распространенный вариант использования функции NULLIF(): значение null вводится в то место, куда должно быть введено другое значение, которое на данный момент или/и неизвестно, или/и недоступно для распознавания, или/и утеряно.

Некоторые программисты не любят применять значение null в качестве слова-заменителя. Поэтому вместо значения null они предпочитают, скажем, число -1 или строку 'N/A' (выражения not applicable или not available переводятся как «не годится», «нет в наличии» или «не определено»). Здесь важно понимать, что все СУБД имеют очень четкие инструкции о том, как обращаться со значениями null. Именно поэтому весьма желательно преобразовать отсутствующие значения, которые должны быть определены пользователем, в значения null. Если бы вам, к примеру, понадобилось вычислить среднее значение какого-нибудь столбца, состоящего из данных, определяемых пользователем, то при условии, что пользователь вводил -1 всякий раз, когда ему было неизвестно верное значение, а вы находили бы среднее среди значений единицы с отрицательным знаком, получился бы неверный результат. Перед вычислением среднего отрицательные числа надо как-то убрать. Вот здесь вам и пригодится функция NULLIF(),

которая преобразует `-1` в значения `null`, пропущенные СУБД при расчете среднего арифметического.

В некотором смысле функция `NULLIF()` есть не более чем сокращенное обозначение одного выражения `CASE` в поисковом формате, а именно: функция `NULLIF(expr1, expr2)` эквивалентна выражению

```
CASE
  WHEN expr1 = expr2 THEN NULL
  ELSE expr1
END
```

Отображение значения null в случае эквивалентности двух выражений

Напечатайте `NULLIF(expr1, expr2)`.

Предполагается, что:

- вместо *expr1* и *expr2* вы подставите выражения, которые хотите сравнить;
- вместо *expr2* вы не будете подставлять литерал `NULL`.

Функция `NULLIF()` сравнивает значения *expr1* и *expr2*, являющиеся ее аргументами, и выдает на выходе (см. листинг 5.33 и рис. 5.33):

- значение `null`, если эти выражения совпадают;
- значение *expr1*, если эти выражения не совпадают.

Листинг 5.33. Столбец `contract` таблицы `titles` нашей типовой базы данных содержит ноль, если для книги, которой соответствует строка, не существует формального контракта. Поменять нули на значения `null`. Ненулевые значения не должны быть изменены этим запросом. Результат исполнения запроса см. на рис. 5.33

Листинг

```
SELECT
  title_id,
  contract,
  NULLIF(contract, 0) AS "Null
contract"
FROM titles;
```

title_id	contract	Null contract
-----	-----	-----
T01	1	1
T02	1	1
T03	1	1
T04	1	1
T05	1	1
T06	1	1
T07	1	1
T08	1	1
T09	1	1
T10	0	NULL
T11	1	1
T12	1	1
T13	1	1

Рис. 5.33. Результат исполнения запроса, представленного в листинге 5.33



Функцию NULLIF() можно использовать везде, где применяется какое-либо выражение, в частности в предложениях SELECT, WHERE, ORDER BY.



СУБД Microsoft Access не поддерживает функцию NULLIF(), поэтому для выполнения в этой среде запроса из листинга 5.33 вам придется вместо функции NULLIF() применить функцию IIf(*expr1* = *expr2*, NULL, *expr1*) и, соответственно, заменить в тексте указанного запроса функцию NULLIF() следующей:

```
IIf(contract = 0, NULL, contract)
```

СУБД Oracle 9i поддерживает функцию NULLIF(), поэтому в этой среде запрос из листинга 5.33 будет выполнен без проблем в том виде, как он есть. Но если вы работаете в среде СУБД Oracle 8i, то для запуска запроса из листинга 5.33 придется вносить в него изменения: вместо функции NULLIF() применить CASE.

В запросе 5.33 можно использовать CASE, например:

```
CASE
  WHEN contract = 0 THEN NULL
  ELSE contract
END
```

СУММИРОВАНИЕ И ГРУППИРОВКА ДАННЫХ

6

Предыдущая глава была посвящена скалярным функциям – таким, которые заданы на значениях одной отдельно взятой строки. Настоящая глава знакомит с функциями агрегатов, или *агрегатными функциями*, или функциями, аргументами которых являются группы строк, называемые *агрегатами*, и на выходе которых всегда одно-единственное резюмирующее значение. В один агрегат вы, по своему усмотрению, можете включить произвольное количество строк, которое может состоять из:

- всех строк произвольной таблицы;
- только строк, заданных неким предложением `WHERE`;
- только строк, созданных неким предложением `GROUP BY`.

Независимо от того, сколько именно строк вы включите в свой агрегат, агрегатная функция выдаст для него только одно значение, например статистическое (сумма, минимум или среднее).

В данной главе помимо собственно агрегатных функций SQL отдельно рассматриваются два предложения команды `SELECT`:

- `GROUP BY` – группирует строки;
 - `HAVING` – фильтрует группы строк.
-

Таблица 6.1. Агрегатные функции

Функция	Результат
MIN(<i>expr</i>)	Минимальное значение в <i>expr</i>
MAX(<i>expr</i>)	Максимальное значение в <i>expr</i>
SUM(<i>expr</i>)	Сумма всех значений в <i>expr</i>
AVG(<i>expr</i>)	Среднее всех значений в <i>expr</i>
COUNT(<i>expr</i>)	Число всех значений, не являющихся значениями null, в <i>expr</i>
COUNT(*)	Число строк в произвольной таблице или произвольном наборе строк

Использование агрегатных функций

Стандартные функции агрегатов языка SQL перечислены в табл. 6.1. Приведем их основные характеристики:

- под *expr* подразумевается не только наименование (заголовок) некоего столбца определенной таблицы, но и какой-нибудь литерал, произвольная функция или комбинация заголовков столбцов, литералов, функций и связывающих все это в единое целое операторов;
- области определения агрегатных функций совпадают не всегда;
- функции SUM() и AVG() определены только для числовых типов данных;
- функции MIN() и MAX() определены для символьных и числовых типов данных, а также для типов данных даты и времени;
- функции COUNT(*expr*) и COUNT(*) определены для всех типов данных;
- все агрегатные функции, за исключением COUNT(*), игнорируют значения null, но вы всегда можете посредством функции COALESCE() подставить в аргумент любой агрегатной функции вместо null произвольное значение (см. раздел «Проверка на значения null с использованием функции COALESCE()» в главе 5);
- функции COUNT(*expr*) и COUNT(*) никогда не дают на выходе значения null, а могут вернуть только натуральное число (положительное целое) или ноль (все остальные агрегатные функции тоже могут дать на выходе значение null, если, скажем, агрегат не содержит ни одной строки или содержит строки, включающие только значения null);

- для того чтобы агрегировать (то есть собрать в одном агрегате) различные значения, следует применить предложение `DISTINCT` (см. раздел «Исключение повторных значений с помощью предложения `DISTINCT`» в этой главе);
- агрегатные функции часто применяют вместе с предложением `GROUP BY` (см. раздел «Группирование строк с использованием предложения `GROUP BY`» в этой главе);
- чтобы ограничить множество строк, которые могут принять участие в агрегатных вычислениях, следует использовать предложение `WHERE` (см. раздел «Фильтрация строк с помощью предложения `WHERE`» в главе 4);
- заголовки по умолчанию для столбцов под агрегатные выражения определяются выбором конкретной СУБД, поэтому, чтобы не полагаться на эти заголовки, следует применить предложение `AS` и назвать результирующий столбец принудительно (см. раздел «Создание псевдонимов столбцов с помощью предложения `AS`» в главе 4);
- никакое агрегатное выражение не может появиться в предложении `WHERE`. Например, если бы вам понадобилось найти среди книг, учитываемых в нашей типовой базе данных, такую, объем продаж которой был бы максимален среди всех книг нашей базы, демонстрируемое применение агрегатной функции `MAX()` было бы неверным:

```
SELECT title_id
FROM titles
WHERE sales = MAX(sales);
```
- любое предложение `SELECT` должно содержать или неагрегатные выражения (скалярные выражения, то есть те, которые работают со своими аргументами строка за строкой), или агрегатные

выражения, но не те и другие одновременно. Например, если бы вам понадобилось найти среди книг, учитываемых в нашей типовой базе данных, такую, объем продаж которой был бы максимальным среди всех книг нашей базы, демонстрируемое применение агрегатной функции `MAX()` было бы неверным:

```
SELECT title_id, MAX(sales)
FROM titles;
```

- единственным исключением из предыдущего правила является то, что в одном предложении `SELECT` можно применять и агрегатные, и неагрегатные выражения, но только для тех столбцов, по которым проводится группировка, причем в этой же самой команде `SELECT` (см. раздел «Группирование строк с использованием предложения `GROUP BY`» в этой же главе). Например, следующий запрос вполне допустим:

```
SELECT type, SUM(sales)
FROM titles
GROUP BY type;
```

- в одном и том же предложении `SELECT` можно применять более одного агрегатного выражения, например:

```
SELECT MIN(sales), MAX(sales)
FROM titles;
```

- агрегатные функции нельзя вкладывать друг в друга. В частности, следующее применение агрегатной функции `AVG()` было бы неверным:

```
SELECT SUM(AVG(sales))
FROM titles;
```

- агрегатные выражения можно применять в подзапросах (см. главу 8). Приведем вполне допустимый запрос, который находит среди книг, учитываемых в нашей

типовой базе данных, ту, объем продаж которой максимален:

```
SELECT title_id, price
FROM titles
WHERE sales =
      (SELECT MAX(sales) FROM titles);
```

- подзапросы нельзя применять в агрегатных выражениях (подзапросам посвящена глава 8), то есть следующее выражение недопустимо:

```
AVG(SELECT price FROM titles)
```



Имейте в виду, что СУБД Oracle позволяет вкладывать агрегатные выражения друг в друга, но только в запросах с группировкой `GROUP BY`. Например, следующий запрос вычисляет средний максимальный объем продаж по всем типам (темам) книг, учитываемых в нашей типовой базе данных. При выполнении этого запроса СУБД Oracle сначала вычисляет внутренний (вложенный) агрегат `MAX(sales)` для групп строк, определенных столбцом `type`, по которому осуществляется группировка, а потом еще раз агрегирует полученные результаты вложенной агрегатной функции:

```
SELECT AVG(MAX(sales))
FROM titles
GROUP BY type;
```

СУБД MySQL 4.0 и более ранних версий не поддерживает подзапросы.

Коммерческие СУБД предоставляют, помимо рассмотренных, не входящие в стандарт SQL агрегатные функции, например для вычисления дополнительных статистических показателей, скажем, среднеквадратичного отклонения (корень квадратный из дисперсии). Если вы хотите как следует разобраться с дополнительными агрегатными функциями вашей СУБД, организуйте поиск в ее документации по ключам «aggregate functions» и «group functions» (агрегатные функции и групповые функции).

Поиск минимума посредством функции MIN()

Чтобы найти минимум по произвольному множеству значений, применяют агрегатную функцию MIN().

Поиск минимума среди произвольного множества значений

Напечатайте MIN(*expr*).

Предполагается, что:

- вместо *expr* вы подставите заголовок столбца, литерал или выражение, для значений которых хотите найти минимум;
- вы получите результат этой функции в виде значения того же типа данных, что и *expr*.

Листинг 6.1 и рис. 6.1 демонстрируют текст команд и результаты сразу нескольких запросов, применяющих агрегатную функцию MIN(). Первый из этих запросов выдает название самой дешевой книги из всех, учитываемых в нашей типовой базе данных. Второй запрос выдает самую раннюю дату публикации по всем книгам, учитываемым в нашей типовой базе данных. Третий запрос находит среди всех книг в базе данных, темой которых является история, ту, чей объем в страницах минимален, и выдает это самое число страниц.

Листинг 6.1. Несколько запросов с применением функции MIN(). Результаты выполнения листинга см. на рис. 6.1

Листинг	
<pre>SELECT MIN(price) AS "Min price" FROM titles; SELECT MIN(pubdate) AS "Earliest pubdate" FROM titles; SELECT MIN(pages) AS "Min history pages" FROM titles WHERE type = 'history';</pre>	
Min price	6.95
Earliest pubdate	1998-04-01
Min history pages	14

Рис. 6.1. Результат исполнения запросов, представленных в листинге 6.1

П

Функцию `MIN()` допустимо применять только к данным, имеющим следующие типы: символьные, числовые и дата и время.

П

Если функцию `MIN()` применить к символьным данным, она найдет среди них такое значение, которое займет самое последнее место в упорядочивающей последовательности (схеме сличения символов) конкретной СУБД (см. раздел «Сортировка строк с помощью предложения `ORDER BY`» в главе 4).

П

Применять предложение `DISTINCT` в сочетании с агрегатной функцией `MIN()` бессмысленно (см. раздел «Исключение повторных значений с помощью предложения `DISTINCT`» в этой главе).



В разных СУБД строковые сравнения могут как зависеть, так и не зависеть от регистра символов (см. примечание с пометкой **DBMS** в разделе «Фильтрация строк с помощью предложения `WHERE`» главы 4).

При сравнении на равенство двух строк, относящихся к типу `VARCHAR`, ваша СУБД может дополнить пробелами справа ту из них, которая короче, и потом сравнить две строки уже равной длины посимвольно. В таком случае, например, строка `'Jack'` и строка `'Jack '` окажутся равными. Чтобы понять, как эти тонкости отразятся на работе функции `MIN()` и ответить на вопрос о том, какую из строк `'Jack'` или `'Jack '` (с двумя пробелами на конце) выберет функция `MIN()`, обратитесь к документации по вашей СУБД или поэкспериментируйте сами.

Поиск максимума с использованием функции MAX()

Чтобы найти максимум по произвольному множеству значений, применяют функцию MAX().

Поиск максимума среди произвольного множества значений

Напечатайте MAX(*expr*).

Предполагается, что:

- вместо *expr* вы подставите заголовок столбца, литерал или выражение, для значений которых хотите найти максимум;
- вы получите результат этой функции в виде значения того же типа данных, что и *expr*.

Листинг 6.2 и рис. 6.2 демонстрируют текст команд и результаты сразу нескольких запросов, применяющих агрегатную функцию MAX(). Первый из этих запросов выдает ту из фамилий авторов, учитываемых в нашей типовой базе данных, которая является последней по алфавиту. Второй запрос выдает цены самой дешевой и самой дорогой книг из всех, учитываемых в нашей типовой базе данных, плюс разброс цен. Третий запрос выводит максимальный валовой доход, полученный на книге, учитываемой в нашей типовой базе данных, тема которой – история (валовой доход равен произведению цены книги и ее объема продаж).

Листинг 6.2. Несколько запросов с применением функции MAX(). Результаты выполнения см. на рис. 6.2

```

Листинг
SELECT MAX(au_lname) AS "Max last name"
FROM authors;

SELECT
    MIN(price) AS "Min price",
    MAX(price) AS "Max price",
    MAX(price) - MIN(price) AS "Range"
FROM titles;

SELECT MAX(price * sales)
      AS "Max history revenue"
FROM titles
WHERE type = 'history';

```

Max last name

O'Furniture

Min price	Max price	Range

6.95	39.95	33.00

Max history revenue

313905.33

Рис. 6.2. Результат исполнения запросов, представленных в листинге 6.2

П

Функцию MAX() допустимо применять только к данным, имеющим следующие типы: символьные, числовые и дата и время.

П

Если функцию MAX() применить к символьным данным, она найдет среди них значение, которое займет самое высокое место в упорядочивающей последовательности (схеме сличения символов) конкретной СУБД (см. раздел «Сортировка строк с помощью предложения ORDER BY» в главе 4).

П

Применять предложение DISTINCT в сочетании с агрегатной функцией MAX() бессмысленно (см. раздел «Исключение повторных значений с помощью предложения DISTINCT» в этой главе).



В разных СУБД строковые сравнения могут как зависеть, так и не зависеть от регистра символов (см. соответствующее примечание с пометкой **DBMS** в разделе «Фильтрация строк с помощью предложения WHERE» в главе 4).

При сравнении на равенство двух строк, относящихся к типу VARCHAR, ваша СУБД может дополнить справа пробелами ту из них, которая короче, и потом сравнить две строки уже равной длины посимвольно. В таком случае, например, строка 'Jack' и строка 'Jack ' (с двумя пробелами на конце) окажутся равными. Чтобы понять, как эти тонкости отразятся на работе функции MAX(), и ответить на вопрос о том, какую из строк 'Jack' или 'Jack ' выберет функция MAX(), обратитесь к документации по вашей СУБД или поэкспериментируйте сами.

Вычисление суммы с помощью функции SUM()

Чтобы найти сумму (итог) произвольно-го множества значений, применяют агрегатную функцию SUM().

Подсчет суммы на произвольном множестве значений

Напечатайте SUM(*expr*).

Предполагается, что вы:

- вместо *expr* подставите заголовок столбца, литерал или выражение, для значений которых вы хотите найти сумму;
- получите результат этой функции в виде значения, чья точность будет не меньше максимальной точности типов данных, примененных в *expr*.

Листинг 6.3 и рис. 6.3 демонстрируют текст команд и результаты сразу нескольких запросов, применяющих агрегатную функцию SUM(). Первый из этих запросов выдает сумму авансовых платежей, выплаченных всем авторам, учитываемым в нашей типовой базе данных. Второй запрос выводит общий объем продаж книг, опубликованных в 2000 году, а третий запрос – итоговую цену (суммарная цена), итоговый объем продаж (сумма объемов по всем книгам) и итоговый валовой доход (сумма по всем книгам произведений цен на объем продаж). Заметьте, между прочим, одну математическую заморочку, которая состоит в том, что сумма произведений вовсе не должна быть равна произведению сумм (мультипликативная и аддитивная операции не коммутативны).

Листинг 6.3. Несколько запросов с применением функции SUM(). Результаты выполнения см. на рис. 6.3

```

Листинг
SELECT SUM(advance) AS "Total advances"
FROM royalties;

SELECT SUM(sales)
      AS "Total sales (2000 books)"
FROM titles
WHERE pubdate
      BETWEEN DATE '2000-01-01'
      AND DATE '2000-12-31';

SELECT
  SUM(price) AS "Total price",
  SUM(sales) AS "Total sales",
  SUM(price * sales) AS "Total revenue"
FROM titles;
```

Total advances		

1336000.00		
	Total sales (2000 books)	

	232677	
Total price	Total sales	Total revenue
-----	-----	-----
220.65	1975446	41428860.77

Рис. 6.3. Результат исполнения запросов, представленных в листинге 6.3



Функция SUM() обрабатывает данные только числовых типов.



Сумма по пустому множеству строк равна значению null, а не нулю, как вы могли бы предположить.



В среде СУБД Microsoft Access при записи литералов даты и времени не употребляется ключевое слово DATE, а сами литералы заключаются не в кавычки, а в знаки решетки (#). Поэтому, чтобы в среде Microsoft Access запустить те запросы, которые представлены в листинге 6.3, вам придется заменить литералы даты и времени во втором запросе на соответственно #2000-01-01# и #2000-12-31#.

В среде СУБД Microsoft SQL Server при записи литералов даты и времени не употребляется ключевое слово DATE. Поэтому, чтобы в среде Microsoft SQL Server запустить запросы, которые представлены в листинге 6.3, вам придется заменить литералы даты и времени во втором запросе на соответственно '2000-01-01' и '2000-12-31'.

Порядок расчета среднего значения с помощью функции AVG()

Чтобы вычислить среднее, или *среднеарифметическое*, значение произвольного множества значений, применяют агрегатную функцию `AVG()`. При этом под среднеарифметическим значением нескольких величин имеется в виду результат деления суммы этих величин на их число.

Для того чтобы вычислить среднее значение произвольного множества значений, напечатайте `SUM(expr)`.

Предполагается, что:

- вместо *expr* вы подставите заголовок столбца, литерал или выражение, для значений которых хотите найти среднее;
- вы получите результат этой функции в виде значения, чья точность будет не меньше максимальной точности типов данных, примененных в *expr*.

Листинг 6.4 и рис. 6.4 демонстрируют текст команд и результаты сразу нескольких запросов, применяющих агрегатную функцию `AVG()`. Первый из этих запросов выдает ту величину, какой равнялась бы средняя цена одной книги, взятая по всему множеству книг, учитываемых в нашей базе, если бы цены этих книг выросли вдвое по сравнению с действующими значениями. Второй запрос выдает средний и общий объемы продаж, вычисленные по множеству книг, учитываемых в нашей базе, и книг, написанных по бизнесу. Обратите внимание, что оба результата равны значению `null`, а не нулю, потому что в нашей базе нет книг по бизнесу, а функция `SUM()` на пустом множестве дает не ноль, а `null`. Третий из этих запросов использует некий субзапрос (см. главу 8)

Листинг 6.4. Несколько запросов с применением функции `AVG()`. Результаты выполнения см. на рис. 6.4

```

Листинг
SELECT AVG(price * 2) AS "AVG(price * 2)"
    FROM titles;

SELECT AVG(sales) AS "AVG(sales)",
       SUM(sales) AS "SUM(sales)"
    FROM titles
   WHERE type = 'business';

SELECT title_id, sales
    FROM titles
   WHERE sales >
         (SELECT AVG(sales) FROM titles)
   ORDER BY sales DESC;
```

```
AVG(price * 2)
```

```
-----
36.775000
```

```
AVG(sales)      SUM(sales)
-----
NULL           NULL
```

```
title_id      sales
-----
T07           1500200
T05           201440
```

Рис. 6.4. Результат исполнения запросов, представленных в листинге 6.4

и с его помощью перечисляет те книги, учитываемые в нашей типовой базе данных, объемы продаж которых превышают средний объем по всей базе.



Функция `AVG()` работает с данными только числовых типов.



Среднее значение по пустому множеству строк равно значению `null`, а вовсе не нулю, как вы могли бы предположить.



Если бы вы применили для обозначения неизвестных или отсутствующих значений не `null`, а, скажем, `-1`, подстановка отрицательных величин в функцию `AVG()` привела бы к неверному значению среднего. Поэтому, если вы заменяете значения `null`, то, прежде чем проводить какие-либо вычисления, примените функцию `NULLIF()`, преобразуйте отсутствующие значения в значения `null`, тем самым исключив их из вычислений (см. раздел «Сравнение выражений с помощью функции `NULLIF()`» в главе 5).



СУБД MySQL 4.0 и более ранних версий не поддерживает подзапросы. Поэтому в среде этой СУБД вы не сможете запустить третий запрос из тех, что представлены в листинге 6.4.

Подсчет строк с помощью функции COUNT()

Чтобы сосчитать строки в произвольном множестве значений, применяют агрегатную функцию COUNT(), которая может быть представлена в двух формах:

- COUNT(*expr*) – выдает число таких строк, в которых аргумент *expr*, называемый *критерием подсчета*, не является значением null;
- COUNT(*) – выдает общее число строк в произвольном множестве, включая значения null и дубликаты.

Подсчет строк, не содержащих значение null

Напечатайте COUNT(*expr*).

Предполагается, что:

- вместо *expr* вы подставите критерий подсчета, которым может быть заголовок столбца, литерал или выражение;
- вы получите результат этой функции в виде натурального числа или нуля (тип INTEGER).

Подсчет всех строк, включая те, которые содержат значение null

Напечатайте COUNT(*). Эта функция выдает натуральное число или ноль (тип INTEGER).

Листинг 6.5 и рис. 6.5 демонстрируют текст команд и результаты сразу трех запросов, применяющих агрегатные функции COUNT(*expr*) и COUNT(*). Все три запроса считают строки таблицы titles и идентичны друг другу во всем, за исключением предложения WHERE. Обратите внимание: счетчики строк в первом запросе различны. Это произошло потому, что столбец price со-

Листинг 6.5. Несколько запросов с применением функций COUNT(*expr*) и COUNT(*). Результаты выполнения см. на рис. 6.5

Листинг

```
SELECT
    COUNT(title_id) AS
"COUNT(title_id)",
    COUNT(price) AS "COUNT(price)",
    COUNT(*) AS "COUNT(*)"
FROM titles;

SELECT
    COUNT(title_id) AS
"COUNT(title_id)",
    COUNT(price) AS "COUNT(price)",
    COUNT(*) AS "COUNT(*)"
FROM titles
WHERE price IS NOT NULL;

SELECT
    COUNT(title_id) AS
"COUNT(title_id)",
    COUNT(price) AS "COUNT(price)",
    COUNT(*) AS "COUNT(*)"
FROM titles
WHERE price IS NULL;
```

COUNT(title_id)	COUNT(price)	COUNT(*)
13	12	13

COUNT(title_id)	COUNT(price)	COUNT(*)
12	12	12

COUNT(title_id)	COUNT(price)	COUNT(*)
1	0	1

Рис. 6.5. Результат исполнения запросов, представленных в листинге 6.5

держит одно значение null. Теперь обратите внимание на то, что во втором запросе все три счетчика строк равны, так как предложение WHERE не рассматривает строку со значением null в столбце price до того, как начнется собственно подсчет строк. Третий запрос показывает разницу между значениями соответствующих счетчиков двух первых запросов.



Функции `COUNT(expr)` и `COUNT(*)` работают на всех типах данных.



Ни функция `COUNT(expr)`, ни функция `COUNT(*)` никогда не дают на выходе значение null.



Использовать предложение `DISTINCT` в сочетании с агрегатной функцией `COUNT(*)` бессмысленно (см. раздел «Исключение повторных значений с помощью предложения `DISTINCT`» в этой главе).

Исключение повторных значений с помощью предложения DISTINCT

Чтобы исключить участие дубликатов строк в вычислении некоторых агрегатных функций, применяют предложение DISTINCT (см. раздел «Удаление повторяющихся строк с помощью ключевого слова DISTINCT» в главе 4).

Начнем с того, что в самом общем виде SQL-синтаксис любой агрегатной функции выглядит так:

```
agg_func([ALL | DISTINCT] expr)
```

Предполагается, что:

- вместо *agg_func* вы подставите какую-нибудь агрегатную функцию (например, MIN(), MAX(), SUM(), AVG() или COUNT());
- вместо выражения *expr*, называемого *критерием*, вы подставите заголовок какого-нибудь столбца, какой-нибудь литерал или какое-нибудь выражение;
- будете иметь в виду, что:
 - опция ALL выбирается СУБД по умолчанию, применяет выбранную агрегатную функцию ко всем строкам агрегата, независимо от значений критерия на них, и поэтому ее почти никогда явно в текст запроса не вставляют;
 - опция DISTINCT применяет выбранную агрегатную функцию только к тем строкам агрегата, на которых критерий принимает различные значения.

С функциями SUM(), AVG() и COUNT(*expr*) предложение DISTINCT работает так: оно исключает дубликаты строк (которые содержатся в агрегате) из процесса вычисления агрегатной функции. Применять это предложение с агрегатными функциями

Листинг 6.6. Несколько агрегатных запросов с применением предложения DISTINCT. Результаты см. на рис. 6.6

Листинг			
<pre> SELECT COUNT(*) AS "COUNT(*)", COUNT(price) AS "COUNT(price)", SUM(price) AS "SUM(price)", AVG(price) AS "AVG(price)" FROM titles; SELECT COUNT(DISTINCT price) AS "COUNT(DISTINCT)", SUM(DISTINCT price) AS "SUM(DISTINCT)", AVG(DISTINCT price) AS "AVG(DISTINCT)" FROM titles; </pre>			
COUNT(*)	COUNT(price)	SUM(price)	AVG(price)
-----	-----	-----	-----
13	12	220.65	18.3875
COUNT(DISTINCT)	SUM(DISTINCT)	AVG(DISTINCT)	
-----	-----	-----	
	10	187.71	18.7710

Рис. 6.6. Результат исполнения запросов, представленных в листинге 6.6

MIN() и MAX() бессмысленно, хотя и не запрещено. Использовать это предложение с агрегатной функцией COUNT(*) недопустимо.

Вычисление суммы на произвольном множестве различных значений

Напечатайте SUM(DISTINCT *expr*).

Предполагается, что:

- *expr* — это заголовок какого-нибудь столбца, литерал или числовое выражение;
- результат выполнения функции будет относиться к числовому типу и иметь точность, равную максимальной точности тех числовых типов данных, значения которых участвуют в критерии *expr*.

Вычисление среднего значения на произвольном множестве различных значений

Напечатайте AVG(DISTINCT *expr*).

Предполагается, что:

- *expr* — это заголовок какого-нибудь столбца, литерал или числовое выражение;
- результат функции будет относиться к числовому типу и иметь точность, равную максимальной точности тех числовых типов данных, значения которых участвуют в критерии *expr*.

Подсчет строк, не содержащих значение null

Напечатайте COUNT(DISTINCT *expr*).

Предполагается, что:

- критерий *expr* — это заголовок какого-нибудь столбца, литерал или числовое выражение;

- результат функции будет относиться к числовому типу INTEGER и будет больше либо равен нулю.

П

Имейте в виду, что предложение DISTINCT в составе предложения SELECT не обязательно даст тот же результат, что и предложение DISTINCT в составе агрегатной функции. В листинге 6.7 представлены три запроса, которые считают идентификаторы авторов, учитываемых в нашей типовой базе данных, в таблице `title_authors`, но каждый по-своему (на рис. 6.7 показаны результаты исполнения этих запросов):

- первый запрос считает все идентификаторы авторов, имеющиеся в наличии;
- второй запрос, несмотря на наличие предложения DISTINCT, выдает тот же результат, что и первый запрос, поскольку еще до того, как предложение DISTINCT начало работать, функция COUNT(*expr*) уже выполнила все вычисления и выдала результат в виде одной-единственной строки, которую это предложение изменить не может;
- в третьем запросе, в отличие от второго, сначала к идентификаторам авторов применяется предложение DISTINCT, и только после этого производится подсчет этих идентификаторов.

П

Имейте в виду, что, если в одном предложении SELECT вы будете одновременно применять и агрегаты, пропущенные через предложение DISTINCT, и агрегаты, не обработанные предложением DISTINCT, результаты всего запроса наверняка будут ошибочны. Запросы, представленные в листинге и на рис. 6.8, как раз демонстрируют все возможные комбинации включения/исключения предложения DISTINCT для сумм и счетчиков:

- первая комбинация — предложение DISTINCT не включено ни в сумму, ни в счетчик;
- вторая комбинация — предложение DISTINCT включено в сумму, но не включено в счетчик;

- третья комбинация – предложение `DI-SINCT` не включено в сумму, но включено в счетчик;
- четвертая комбинация – предложение `DISTINCT` включено и в сумму, и в счетчик.

Так вот, из этих четырех запросов только первый (нет ни одного включения предложения `DISTINCT`) и последний (оба включения) математически осмысленны, потому что, по крайней мере, считают некую сумму и число слагаемых этой самой суммы. Это легко проверить с помощью агрегатных функций `AVG(price)` и `AVG(DISTINCT price)`. Дело в том, что в смешанных третьем и втором случаях вы не сможете определить эти величины делением суммы на счетчик.



СУБД Microsoft Access не поддерживает агрегатные функции с предложением `DISTINCT` в качестве аргумента. Например, следующий запрос в среде Microsoft Access был бы некорректен:

```
SELECT SUM(DISTINCT price)
FROM titles;
```

Однако вы могли бы имитировать такую сумму с предложением `DISTINCT` в качестве аргумента посредством подзапроса (см. примечания в разделе «Принципы работы с подзапросами» главы 8), например:

```
SELECT SUM(price)
FROM (SELECT DISTINCT price
FROM titles);
```

К тому же этот обходной маневр в среде Access не позволит вам запутаться во включениях/исключениях предложения `DISTINCT` в агрегатные функции, как это было в третьем и втором запросах из тех четырех, что представлены в листинге 6.8.

Имейте в виду, что СУБД MySQL поддерживает агрегатную функцию `COUNT(DISTINCT expr)`, но не поддерживает ни `SUM(DISTINCT expr)`, ни `AVG(DISTINCT expr)`. Поэтому запросы, представленные в листингах 6.6 и 6.8, в среде MySQL

работать не будут. Работая в СУБД Microsoft SQL Server, надо иметь в виду, что, если вы применяете предложение `DISTINCT`, вместо *expr* можно подставить только заголовок столбца, а, скажем, арифметическое выражение – нельзя. Например, следующий запрос был бы некорректен в СУБД Microsoft SQL Server:

```
SELECT COUNT(DISTINCT price * sales)
FROM titles;
```

Листинг 6.7. Несколько агрегатных запросов с применением предложения `DISTINCT` показывают, что предложения `DISTINCT` в предложении `SELECT` и в составе некой агрегатной функции имеют разный смысл. Результаты см. на рис. 6.7

Листинг	
<pre>SELECT COUNT(au_id) AS "COUNT(au_id)" FROM title_authors;</pre>	<pre>COUNT(au_id) ----- 17</pre>
<pre>SELECT DISTINCT COUNT(au_id) AS "DISTINCT COUNT(au_id)" FROM title_authors;</pre>	<pre>DISTINCT COUNT(au_id) ----- 17</pre>
<pre>SELECT COUNT(DISTINCT au_id) AS "COUNT(DISTINCT au_id)" FROM title_authors;</pre>	<pre>COUNT(DISTINCT au_id) ----- 6</pre>

Рис. 6.7. Результат исполнения запросов, представленных в листинге 6.7

Листинг 6.8. Примеры запросов со смешанным применением агрегатов, обработанных и не обработанных предложением DISTINCT в одном и том же предложении SELECT. Результаты подобных запросов могут быть ошибочными (см. рис. 6.8)

Листинг

```
SELECT
    COUNT(price)
      AS "COUNT(price)",
    SUM(price)
      AS "SUM(price)"
FROM titles;

SELECT
    COUNT(price)
      AS "COUNT(price)",
    SUM(DISTINCT price)
      AS "SUM(DISTINCT price)"
FROM titles;

SELECT
    COUNT(DISTINCT price)
      AS "COUNT(DISTINCT price)",
    SUM(price)
      AS "SUM(price)"
FROM titles;

SELECT
    COUNT(DISTINCT price)
      AS "COUNT(DISTINCT price)",
    SUM(DISTINCT price)
      AS "SUM(DISTINCT price)"
FROM titles;
```

COUNT(price)SUM(price)	

12	220.65
COUNT(price)SUM(DISTINCT price)	

12	187.71
COUNT(DISTINCT price)SUM(price)	

10	220.65
COUNT(DISTINCT price)	SUM(DISTINCT price)

10	187.71

Рис. 6.8. Результаты запросов, представленных в листинге 6.8. Мы видим, что счетчики и суммы надо вычислять одинаково. В противном случае значения средних, получаемые как частное от деления суммы на счетчик, будут неверными, как во втором (187.71/12) и третьем (220.65/10) запросах. В то же время первый (220.65/12) и четвертый (187.71/10) запросы дают результаты, имеющие определенный смысл

Группирование строк с использованием предложения GROUP BY

До сих пор мы рассматривали применение агрегатных функций ограниченно, то есть всего лишь итожили все значения какого-нибудь столбца или все те значения, которые удовлетворяли какому-нибудь условию поиска предложения WHERE. Однако SQL предоставляет нам возможность разбивать любую таблицу на логические *группы* (категории) и вычислять агрегатные статистические функции на каждой из этих групп. Для этого и служит предложение GROUP BY.

Лучше всего пояснить эту абстрактную идею примером. Например, запрос, представленный в листинге 6.9, применяет предложение GROUP BY для того, чтобы подсчитать те книги, которые написал, в том числе в соавторстве, каждый автор, учитываемый в нашей типовой базе данных. В соответствующем предложении SELECT столбец `au_id` идентифицирует авторов, а производный столбец `num_books`, называемый *агрегатным*, подсчитывает книги каждого автора. Так вот, роль предложения GROUP BY в этом запросе состоит в том, чтобы заставить СУБД вычислить столбец независимых значений `num_books`, по одному на каждого нашего автора (то есть на каждый идентификатор `au_id`), а не одно-единственное значение на всю таблицу. Фактически запрос выполняет разбиение/группировку книг по авторам. Это значит, что, поскольку столбец `au_id`, будучи первичным ключом, однозначно идентифицирует наших авторов, группировка строк таблицы `title_authors` проводится по столбцу `au_id`, который в этом случае называется *группирующим*. Основные

Листинг 6.9. Распечатать идентификаторы авторов, учитываемых в нашей типовой базе данных, и напротив каждого идентификатора указать то количество книг, которое написал соответствующий автор, в том числе в соавторстве. Результат исполнения запроса см. на рис. 6.9

Листинг																	
<pre>SELECT au_id, COUNT(*) AS "num_books" FROM title_authors GROUP BY au_id;</pre>																	
<table> <tr> <th>au_id</th><th>num_books</th></tr> <tr> <td>-----</td><td>-----</td></tr> <tr> <td>A01</td><td>3</td></tr> <tr> <td>A02</td><td>4</td></tr> <tr> <td>A03</td><td>2</td></tr> <tr> <td>A04</td><td>4</td></tr> <tr> <td>A05</td><td>1</td></tr> <tr> <td>A06</td><td>3</td></tr> </table>	au_id	num_books	-----	-----	A01	3	A02	4	A03	2	A04	4	A05	1	A06	3	
au_id	num_books																
-----	-----																
A01	3																
A02	4																
A03	2																
A04	4																
A05	1																
A06	3																

Рис. 6.9. Результат исполнения запроса, представленного в листинге 6.9

характеристики предложения GROUP BY таковы:

- располагается в команде SELECT после предложения WHERE, но перед предложением ORDER BY;
- группирующими столбцами могут быть заголовки (имена) столбцов или заголовки производных столбцов;
- любой столбец, представленный в предложении SELECT, но не являющийся агрегатным, должен содержаться и в предложении GROUP BY. Например, следующий запрос некорректен:

```
SELECT type, pub_id, COUNT(*)
FROM titles
GROUP BY type;
```

потому, что:

- имени столбца `pub_id` нет в предложении GROUP BY;
- группирующий столбец единственный и предложение GROUP BY может выдавать только одну строку на каждое значение столбца `type`, следовательно, не может выдать несколько значений столбца `pub_id`, которые надо связать со значением столбца `type`;
- если предложение SELECT содержит некое сложное выражение, не являющееся агрегатным (сложным в данном случае является любое выражение, отличное от просто заголовка столбца), соответствующее выражение в предложении GROUP BY должно совпадать с этим сложным выражением из предложения SELECT буквально, а не по смыслу;
- чтобы организовать вложенные группы, следует указать несколько группирующих столбцов (при этом реальный подсчет данных будет происходить по той

группе, которая определена последней, то есть по последнему группирующему столбцу);

- если произвольный группирующий столбец содержит значение null, то строка, в которую попадает это значение, после выполнения данного запроса становится отдельной группой;
- если группирующий столбец содержит несколько значений null, все они помещаются в одну группу, что не означает равенства значений null, сколько бы их ни было;
- устранять строки из результата запроса следует предложением WHERE, причем до применения предложения GROUP BY;
- применять псевдонимы столбцов в предложении GROUP BY нельзя, но применять псевдонимы таблиц в качестве квалификаторов можно (см. раздел «Создание псевдонимов столбцов с помощью предложения AS» в главе 4);
- без предложения ORDER BY группы, выдаваемые предложением GROUP BY, не будут упорядочены, и для сортировки результата запроса (в порядке убывания книг), представленного в листинге 6.9, вам придется добавить предложение ORDER BY "num_books" DESC.

Порядок группирования строк

Напечатайте:

```
SELECT columns
FROM table
[WHERE search_condition]
GROUP BY grouping_columns
[HAVING search_condition]
[ORDER BY sort_columns];
```

Предполагается, что:

- вместо `table` вы подставите имя той таблицы, строки которой хотите сгруппировать;

- вместо *columns* и *grouping_columns* вы подставите по одному столбцу или по набору перечисленных через запятую заголовков столбцов таких, чтобы:
 - оба набора столбцов принадлежали таблице *table*, исключая агрегатные столбцы;
 - *columns*, включая агрегатные столбцы, соответствовали тем столбцам, которые вы хотите видеть в результате исполнения запроса;
 - *grouping_columns* соответствовали набору группирующих столбцов;
 - все столбцы набора столбцов *columns*, не являющиеся агрегатными, были представлены в наборе группирующих столбцов *grouping_columns*;
 - порядок следования заголовков столбцов в *grouping_columns* определял уровни группировки от высшего до самого нижнего.

Предложение `GROUP BY` реализуется следующим образом:

- ограничивает число строк результата так, что для каждого уникального значения группирующего столбца или группирующих столбцов (здесь надо понимать, что значением набора столбцов является упорядоченный набор значений составляющих столбцов, то есть вектор) в результате запроса появляется только одна строка;
- каждая строка результата в своих агрегатных столбцах содержит резюмирующую информацию, соответствующую тому уникальному значению упорядочивающих столбцов, которое в результате определило появление этой строки.

Если предложение `SELECT` включает предложение `WHERE`, СУБД приступит к группировке строк только после того, как применит условие поиска *search_condition* ко всем строкам таблицы *table*.

Если предложение `SELECT` включает и предложение `WHERE`, и предложение `ORDER BY`, столбцы условия *sort_columns* должны быть набраны из столбцов *columns* (см. разделы «Фильтрация строк с помощью предложения `WHERE`» и «Сортировка строк с помощью предложения `ORDER BY`» в главе 4).

Наконец, предложение `HAVING`, фильтрующее уже сгруппированные строки, будет рассмотрено в следующем разделе.

Листинг 6.10 и рис. 6.10 в одном и том же запросе, включающем предложение `GROUP BY`, наглядно демонстрируют разницу между агрегатными функциями `COUNT(expr)` и `COUNT(*)`. Обратите внимание, что таблица *publishers* содержит одно значение `null` (для издателя P03 в Германии). Теперь вспомните: в разделе «Подсчет строк с помощью функции `COUNT()`» этой главы мы говорили, что агрегатная функция `COUNT(expr)` считает только те строки, значение критерия *expr* на которых не является значением `null`, и что функция `COUNT(*)` считает все строки агрегата. В ходе исполнения рассматриваемого запроса предложение `GROUP BY` распознает значение `null` и создает предназначенную специально для него группу. После этого счетчик `COUNT(*)` находит значение `null` и вычисляет его для группы, специально предусмотренной для значений `null`. Поэтому для значения `null` столбца *state* счетчик (то есть значение агрегатного столбца) `COUNT(*)` равен 1. В то же время счетчик `COUNT(state)` находит это значение `null` в той самой группе, но не вычисляет его потому, что эта агрегатная функция так устроена. Таким образом, значение

Листинг 6.10. Этот запрос демонстрирует разницу между агрегатными функциями `COUNT(expr)` и `COUNT(*)` в одном и том же предложении `SELECT`, включающем предложение `GROUP BY`. Результат исполнения запроса см. на рис. 6.10

```

SELECT
    state,
    COUNT(state) AS "COUNT(state)",
    COUNT(*) AS "COUNT(*)"
FROM publishers
GROUP BY state;

```

state	COUNT(state)	COUNT(*)
NULL	0	1
CA	2	2
NY	1	1

Рис. 6.10. Результат исполнения запроса, представленного в листинге 6.10

счетчика `COUNT(state)` для значения `null` столбца `state` равно 0.

Имейте в виду, что, если некий столбец, не являющийся агрегатным, содержит значения `null`, применение агрегатной функции `COUNT(*)` вместо агрегатной функции `COUNT(expr)` может привести к неверным результатам. Запрос, представленный в листинге 6.11 и на рис. 6.11, как раз и демонстрирует этот случай – он должен распечатать итоговую статистику продаж с разбивкой по типам книг. Но значение объема продаж для одной из биографий есть `null`. Поэтому, как мы теперь знаем, счетчики `COUNT(sales)` и `COUNT(*)` различаются на 1. И хотя расчет среднего значения в пятом столбце с псевдонимом `SUM/COUNT(sales)` математически состоятелен в том смысле, что равен некой сумме, деленной на число ее слагаемых, расчет среднего значения в шестом столбце с псевдонимом `SUM/COUNT(*)` математически несостоятелен. Эту несостоятельность мы проверили агрегатной функцией `AVG(sales)`, значения которой с разбивкой по типам книг распечатаны в последнем столбце (см. листинг 6.8).

В листинге 6.12 и на рис. 6.12 представлен один простой запрос, включающий предложение `GROUP BY`, который вычисляет и распечатывает общий объем продаж, средний объем продаж и общее число книг по нашей типовой базе данных с разбивкой по типам этих книг. Запрос, представленный в листинге 6.13 и на рис. 6.13, является модификацией предыдущего запроса, суть которой в том, что мы добавили предложение `WHERE`, чтобы еще до группировки исключить из рассмотрения все книги с ценой ниже \$13, и предложение `ORDER BY`, чтобы отсортировать результат в порядке уменьшения общих объемов продаж.

Листинг 6.14 и рис. 6.14 демонстрируют применение нескольких группирующих столбцов для того, чтобы подсчитать число книг каждого типа, которые напечатал каждый издатель.

В листинге 6.15 и на рис. 6.15 показана некая модификация запроса, представленного в листинге 5.31 и на рис. 5.31 (см. главу 5). Здесь мы не распечатывали список книг нашей базы с разбивкой по категориям объемов продаж, а использовали предложение GROUP BY, чтобы показать число книг в каждой из этих категорий.

C

Чтобы исключить из рассмотрения те строки, которые вы не хотите группировать, применяйте предложение WHERE, а чтобы фильтровать строки после группировки – предложение HAVING, которое подробно рассматривается в следующем разделе.

Листинг 6.11. Для получения математически достоверных результатов следует применять не функцию COUNT(*), а функцию COUNT(*expr*). В противном случае ошибка неизбежна, когда *expr* содержит значения null. Результат исполнения запроса см. на рис. 6.11

Листинг

```
SELECT
    type,
    SUM(sales) AS "SUM(sales)",
    COUNT(sales) AS "COUNT(sales)",
    COUNT(*) AS "COUNT(*)",
    SUM(sales)/COUNT(sales) AS "SUM/COUNT(sales)",
    SUM(sales)/COUNT(*) AS "SUM/COUNT(*)",
    AVG(sales) AS "AVG(sales)"
FROM titles
GROUP BY type;
```

type	SUM(sales)	COUNT(sales)	COUNT(*)	SUM/COUNT(sales)	SUM/COUNT(*)	AVG(sales)
biography	1611521	3	4	537173.67	402880.25	537173.67
children	9095	2	2	4547.50	4547.50	4547.50
computer	25667	1	1	25667.00	25667.00	25667.00
history	20599	3	3	6866.33	6866.33	6866.33
psychology	308564	3	3	102854.67	102854.67	102854.67

Рис. 6.11. Результат исполнения запроса, представленного в листинге 6.11

Листинг 6.12. Этот простой запрос с применением предложения GROUP BY вычисляет несколько итоговых статистических показателей для каждого типа книг. Результат исполнения запроса см. на рис. 6.12

```

SELECT
    type,
    SUM(sales)      AS "SUM(sales)",
    AVG(sales)      AS "AVG(sales)",
    COUNT(sales) AS "COUNT(sales)"
FROM titles
GROUP BY type;
    
```

type	SUM(sales)	AVG(sales)	COUNT(sales)
-----	-----	-----	-----
biography	1611521	537173.67	3
children	9095	4547.50	2
computer	25667	25667.00	1
history	25559	6866.33	3
psychology	308564	102854.67	3

Рис. 6.12. Результат исполнения запроса, представленного в листинге 6.12

Листинг 6.13. Здесь в текст запроса из листинга 6.12 добавлены предложения WHERE и ORDER BY, чтобы не учитывать книги дешевле \$13 и упорядочить результат по уменьшению общего объема продаж. Результат исполнения запроса см. на рис. 6.13

```

SELECT
    type,
    SUM(sales)      AS "SUM(sales)",
    AVG(sales)      AS "AVG(sales)",
    COUNT(sales) AS "COUNT(sales)"
FROM titles
WHERE price >= 13
GROUP BY type
ORDER BY "SUM(sales)" DESC;
    
```

type	SUM(sales)	AVG(sales)	COUNT(sales)
-----	-----	-----	-----
biography	1511520	755760.00	2
computer	25667	25667.00	1
history	25559	6866.33	3
psychology	5000	5000.00	1

Рис. 6.13. Результат исполнения запроса, представленного в листинге 6.13

Листинг 6.14. Перечислить книги каждого типа с разбивкой по издателям в убывающем порядке (нижний уровень группировки) внутри возрастающего порядка идентификаторов издателей (верхний уровень группировки). Результат исполнения запроса см. на рис. 6.14

Листинг

```
SELECT
    pub_id,
    type,
    COUNT(*) AS "COUNT(*)"
FROM titles
GROUP BY pub_id, type
ORDER BY pub_id ASC, "COUNT(*)" DESC;
```

pub_id	type	COUNT(*)
-----	-----	-----
P01	biography	3
P01	history	1
P02	computer	1
P03	history	2
P03	biography	1
P04	psychology	3
P04	children	2

Рис. 6.14. Результат исполнения запроса, представленного в листинге 6.14

Листинг 6.15. Перечислить книги в каждой категории объемов продаж в порядке их уменьшения. Результат исполнения запроса см. на рис. 6.15

Листинг

```
SELECT
    CASE
        WHEN sales IS NULL
        THEN 'Unknown'
        WHEN sales <= 1000
        THEN 'Not more than 1,000'
        WHEN sales <= 10000
        THEN 'Between 1,001 and 10,000'
        WHEN sales <= 100000
        THEN 'Between 10,001 and 100,000'
        WHEN sales <= 1000000
        THEN 'Between 100,001 and 1,000,000'
        ELSE 'Over 1,000,000'
    END
    AS "Sales category",
    COUNT(*) AS "Num titles"
FROM titles
GROUP BY
    CASE
        WHEN sales IS NULL
        THEN 'Unknown'
        WHEN sales <= 1000
        THEN 'Not more than 1,000'
        WHEN sales <= 10000
        THEN 'Between 1,001 and 10,000'
        WHEN sales <= 100000
        THEN 'Between 10,001 and 100,000'
        WHEN sales <= 1000000
        THEN 'Between 100,001 and 1,000,000'
        ELSE 'Over 1,000,000'
    END
ORDER BY MIN(sales) ASC;
```

Sales category	Num titles
-----	-----
Unknown	1
Not more than 1,000	1
Between 1,001 and 10,000	3
Between 10,001 and 100,000	5
Between 100,001 and 1,000,000	2
Over 1,000,000	1

Рис. 6.15. Результат исполнения запроса, представленного в листинге 6.15

Листинг 6.16. Обратите внимание, что оба представленных здесь запроса дают один и тот же результат, который показан на рис. 6.16

```

SELECT type
FROM titles
GROUP BY type;

SELECT DISTINCT type
FROM titles;
```

```

type
-----
biography
children
computer
history
psychology
```

Рис. 6.16. Результат исполнения запросов, представленных в листинге 6.16

П

Если в некоем запросе предложение GROUP BY применяется без агрегатной функции, оно обрабатывается именно так, как предложение DISTINCT (см. листинг 6.16 и рис. 6.16). Подробнее об этом предложении читайте в разделе «Удаление повторяющихся строк с помощью ключевого слова DISTINCT» главы 4.

П

Вы можете применять предложения GROUP BY для того, чтобы искать скрытые закономерности и взаимосвязи в данных вашей базы. Например, запросом, представленным в листинге 6.17 и на рис. 6.17, мы пытались найти зависимость между категориями цены и средними продажами.

С

Никогда не полагайтесь на то, что предложение GROUP BY упорядочит строки результата. Мы настоятельно рекомендуем всегда применять предложение ORDER BY вместе с предложением GROUP BY (несмотря на то, что из нескольких примеров мы для краткости исключили предложение ORDER BY). Более того, некоторые коммерческие СУБД требуют применять ORDER BY вместе с GROUP BY.

П

Если некая агрегатная функция благодаря предложению GROUP BY выдает многомерный (то есть состоящий из нескольких компонентов) результат в виде определенного набора значений, этот набор называется *вектором агрегатов*, а сами значения — агрегатами вектора, или *векторными агрегатами* (здесь может возникнуть некоторая путаница в терминах, поскольку до сих пор под агрегатом понималось множество строк как аргумент агрегатной функции). Если же в структуре запроса нет предложения GROUP BY и агрегатная функция этого запроса выдает единственное значение, это значение принято называть *скалярным агрегатом*.



Если вы работаете в среде СУБД Microsoft Access, то для запуска запроса из листинга 6.15 выражение `CASE` придется заменить функцией `SWITCH()` (см. примечание с пометкой **DBMS** в разделе «Вычисление условных значений с помощью выражения `CASE`» главы 5).

СУБД MySQL не поддерживает предложения `CASE` в составе предложения `GROUP BY`. Поэтому запрос, представленный в листинге 6.15, в MySQL работать не будет.

Имейте в виду, что некоторые коммерческие СУБД, например MySQL и PostgreSQL, разрешают использовать псевдонимы столбцов в предложении `GROUP BY`.

Листинг 6.17. Показать средние продажи с разбивкой по цене в порядке ее уменьшения. Результат исполнения запроса см. на рис. 6.17

```

Листинг

SELECT price, AVG(sales) AS "AVG(sales)"
FROM titles
WHERE price IS NOT NULL
GROUP BY price
ORDER BY price ASC;
```

price	AVG(sales)
6.95	201440.0
7.99	94123.0
10.00	4095.0
12.99	56501.0
13.95	5000.0
19.95	10443.0
21.99	566.0
23.95	1500200.0
29.99	10467.0
39.95	25667.0

Рис. 6.17. Результат исполнения запроса, представленного в листинге 6.17. Мы видим, что, если игнорировать явную статистическую аномалию, связанную с ценой \$23,95, очевидна слабая обратная зависимость объема продаж от цены

Фильтрация групп с помощью предложения HAVING

В определенном смысле предложение HAVING накладывает ограничения на предложение GROUP BY подобно тому, как предложение WHERE взаимодействует с командой SELECT. Перечислим основные характеристики предложения HAVING:

- в команде SELECT оно помещается после предложения GROUP BY, но перед ORDER BY;
- именно так, как предложение WHERE ограничивает число строк, показываемых командой SELECT, предложение HAVING ограничивает число групп, показываемых предложением GROUP BY;
- СУБД применяет условие поиска предложения WHERE до того, как осуществляется группировка, а условие поиска предложения HAVING выполняется после этого;
- синтаксис его аналогичен синтаксису предложения WHERE, только HAVING может содержать агрегатные функции;
- может сослаться на любые составляющие списка предложения SELECT.

Очередность, с которой СУБД применяет предложения WHERE, GROUP BY и HAVING, такова:

1. Предложение WHERE фильтрует строки, которые являются результатом работы операций, обозначенных предложениями FROM и JOIN.
2. Предложение GROUP BY группирует выход предложения WHERE.
3. Предложение HAVING фильтрует строки сгруппированного результата.

Фильтрация групп

Вслед за предложением GROUP BY напечатайте HAVING *search_condition*.

Предполагается, что вы по своему выбору вместо *search_condition* подставите реальное условие поиска, которое хотите применить для фильтрации групп и которое должно отвечать следующим требованиям:

- содержать агрегатные функции;
- во всем, кроме включения в свой состав агрегатных функций, быть аналогичным условию поиска предложения WHERE, рассмотренному в разделе «Фильтрация строк с помощью предложения WHERE» главы 4;
- объединять несколько условий поиска логическими операторами AND и OR в сложное условие поиска предложения HAVING, инвертируя, если надо, некоторые из этих составляющих логическим оператором NOT.

Предложение HAVING обрабатывается таким образом, что его условие поиска применяется к строкам выхода, который производит группировка. И только те группы, которые соответствуют условию поиска, проходят в результат. При этом вы можете применять предложение HAVING только к столбцам, появляющимся в предложении GROUP BY или в какой-нибудь агрегатной функции.

Для иллюстрации всего вышесказанного рассмотрим несколько запросов. В запросе из листинга 6.18 мы модифицировали запрос, который был рассмотрен в листинге 6.9. Суть модификации заключается в следующем: вместо того чтобы перечислять с разбивкой по авторам книги, которые были написаны, в том числе

в соавторстве, каждым из авторов, учитываемых в нашей типовой базе данных, мы применили предложение `HAVING` с целью перечислить только тех авторов, кто написал, в том числе в соавторстве, не менее трех книг.

В запросе из листинга 6.19 условие поиска предложения `HAVING` одновременно является одним из агрегатных выражений предложения `SELECT`. Этим мы демонстрируем, что условие поиска предложения `HAVING` может включать агрегатные функции. Как видно из листинга 6.20 и рис. 6.20, запрос, представленный в листинге 6.19, будет выполняться даже тогда, когда вы удалите агрегатную функцию `AVG()` из перечня `SELECT` и оставите функцию, которая входит в состав условия поиска предложения `HAVING`, единственной применяемой в запросе.

Запрос, представленный в листинге 6.21 и на рис. 6.21, с помощью нескольких группирующих столбцов подсчитывает количество книг каждого типа, которые опубликовал каждый издатель. При этом условие поиска предложения `HAVING` исключает те группы второго уровня группировки, в которых данный издатель (первый уровень группировки) имеет не более одной опубликованной книги произвольного заданного типа (второй уровень группировки). Запрос выдает некое подмножество того результата, который был уже получен с помощью запроса, представленного в листинге 6.14.

В запросе из листинга 6.22 предложение `WHERE`, как мы видим, сначала исключает из рассмотрения все книги, кроме тех, что выпущены издателями P03 и P04. После этого предложение `GROUP BY` группирует выход предложения `WHERE` по столбцу `type`. Наконец, предложение `HAVING` фильтрует строки в сформированных группах.

Листинг 6.18. Перечислить авторов, учитываемых в нашей типовой базе данных и написавших, в том числе в соавторстве, не менее трех книг. Результат исполнения запроса представлен на рис. 6.18

Листинг													
<pre>SELECT au_id, COUNT(*) AS "num_books" FROM title_authors GROUP BY au_id HAVING COUNT(*) >= 3;</pre>													
<table> <tr> <th>au_id</th><th>num_books</th></tr> <tr> <td>-----</td><td>-----</td></tr> <tr> <td>A01</td><td>3</td></tr> <tr> <td>A02</td><td>4</td></tr> <tr> <td>A04</td><td>4</td></tr> <tr> <td>A06</td><td>3</td></tr> </table>	au_id	num_books	-----	-----	A01	3	A02	4	A04	4	A06	3	
au_id	num_books												
-----	-----												
A01	3												
A02	4												
A04	4												
A06	3												

Рис. 6.18. Результат исполнения запроса, представленного в листинге 6.18

Листинг 6.19. Перечислить книги, средний валовой доход по которым превышает 1 миллион долларов, и средний валовой доход с разбивкой по их типам. Результат исполнения запроса см. на рис. 6.19

ЛИСТИНГ

```
SELECT
```

```
    type,
```

```
    COUNT(price) AS "COUNT(price)",
```

```
    AVG(price * sales) AS "AVG revenue"
```

```
FROM titles
```

```
GROUP BY type
```

```
HAVING AVG(price * sales) > 1000000;
```

type	COUNT(price)	AVG revenue
-----	-----	-----
biography	3	12484879.00
computer	1	1025396.65

Рис. 6.19. Результат исполнения запроса, представленного в листинге 6.19

Листинг 6.20. Запрос получен удалением из предложения SELECT команды листинга 6.19 агрегатной функции AVG(price*sales). На рис. 6.20 мы видим, что модифицированный запрос работает

Листинг

```
SELECT
    type,
    COUNT(price) AS "COUNT(price)"
FROM titles
GROUP BY type
HAVING AVG(price * sales) > 1000000;
```

type	COUNT(price)
-----	-----
biography	3
computer	1

Рис. 6.20. Результат исполнения запроса, представленного в листинге 6.20

Листинг 6.21. Перечислить с разбивкой по издателям (первый уровень) и типам книг (второй уровень) опубликованные книги, учитываемые в нашей типовой базе данных. Для каждого фиксированного издателя из результата исключаются группы (второй уровень), в которых этот издатель выпустил не более одной книги. Результат исполнения запроса см. на рис. 6.21

Листинг

```
SELECT
    pub_id,
    type,
    COUNT(*) AS "COUNT(*)"
FROM titles
GROUP BY pub_id, type
HAVING COUNT(*) > 1
ORDER BY pub_id ASC, "COUNT(*)" DESC;
```

pub_id	type	COUNT(*)
-----	-----	-----
P01	biography	3
P03	history	2
P04	psychology	3
P04	children	2

Рис. 6.21. Результат исполнения запроса, представленного в листинге 6.21

П

Предложения `HAVING` должны включать только агрегаты. Только те и никакие другие условия, которые необходимо применить *после* того, как проведена группировка, следует определить в предложении `HAVING`. Дело в том, что уточнить условия, которые необходимо применить *до* того, как будет проведена группировка, гораздо эффективнее в предложении `WHERE`. Поясним эту мысль. В следующем примере первая из двух эквивалентных по результату команд работает быстрее, что предпочтительнее, так как она сокращает количество строк, которые необходимо сгруппировать:

```
SELECT pub_id, SUM(sales)
FROM titles
WHERE pub_id IN ('P03', 'P04')
GROUP BY pub_id
HAVING SUM(sales) > 1000;
```

```
SELECT pub_id, SUM(sales)
FROM titles
GROUP BY pub_id
HAVING SUM(sales) > 1000
AND pub_id IN ('P03', 'P04');
```

Листинг 6.22. Перечислить для издателей P03 и P04 общий объем продаж выпущенных книг с разбивкой по их типам. Учесть только те книги, итоговый объем продаж которых превышает 10 000 копий при средней цене одной копии менее \$20. Результат исполнения запроса см. на рис. 6.22

```
SELECT
    type,
    SUM(sales) AS "SUM(sales)",
    AVG(price) AS "AVG(price)"
FROM titles
WHERE pub_id IN ('P03', 'P04')
GROUP BY type
HAVING SUM(sales) > 10000
AND AVG(price) < 20;
```

type	SUM(sales)	AVG(price)
-----	-----	-----
psychology	308564	9.31

Рис. 6.22. Результат исполнения запроса, представленного в листинге 6.22

ВЫБОР ДАННЫХ ИЗ НЕСКОЛЬКИХ ТАБЛИЦ

7

Все запросы, которые мы рассматривали до сих пор, обладали одной общей чертой. Каждый из них выбирал данные из какой-нибудь одной таблицы. В этой главе мы объясним, как с помощью объединений выбирать строки одновременно из нескольких таблиц. Эта задача очень важна в практической работе с базами данных. Для начала припомните, что в разделе «Связи» главы 2 мы определили, что связь – это объединение, устанавливаемое между общими столбцами двух разных таблиц. Так вот, объединение – это такая табличная операция, на вход которой подаются две разные таблицы и которая применяет связанные столбцы этих таблиц для того, чтобы объединить их строки в некую одну общую таблицу. В принципе с помощью цепочки объединений вы можете сгруппировать любое число таблиц.

В чем же заключается важность объединений? Дело в том, что основная информация любой базы данных хранится не в ее

таблицах, а, образно выражаясь, между ними. Конечно, имеются в виду связи между множествами строк. Если взять, к примеру, нашу типовую базу данных, таблицы `authors`, `publishers` и `titles` содержат важную информацию, спору нет. Но именно связи между этими таблицами дают возможность отвечать на вопросы о том, кто и что написал, кто и что опубликовал, кому были посланы чеки в счет выплаты авторского гонорара и т.д. Следовательно, именно связи позволяют анализировать данные как систему. И только анализируя данные о произвольном объекте как единое целое (то есть работая с моделью данных), мы можем получить о нем адекватное представление.

Учитывая все вышесказанное, в этой главе мы рассмотрим различные типы допустимых в рамках стандарта SQL объединений, объясним их назначение и то, как следует строить команды `SELECT`, которые используют эти объединения.

Уточнение имен столбцов

Вспомните, в разделе «Таблицы, столбцы и строки» главы 2 мы отмечали, что имена столбцов в пределах одной таблицы должны быть уникальными для любой таблицы, но могут повторяться в других таблицах. Например, таблицы нашей типовой базы данных `authors` и `publishers` содержат столбцы `city`. Поэтому, если при построении запроса рассматривается более одной таблицы, имена столбцов могут перестать быть их однозначными идентификаторами.

Для того чтобы определить те столбцы, имена которых перестали быть их однозначными идентификаторами из-за перехода к нескольким таблицам, используют так называемые уточненные имена. Любое *уточненное имя* является составным. В простейшем случае для получения составного имени произвольного столбца надо напечатать имя его таблицы, точку (без пробела) и имя этого столбца в этой таблице. Очевидно, что в любой прикладной системе, применяющей только одну базу данных, такие уточненные имена будут однозначно идентифицировать свои столбцы просто потому, что таблицы должны иметь в рамках одной базы уникальные имена. Более сложные уточненные имена применяются в сложных сетевых иерархических СУБД (см. примечание DBMS в этом разделе).

Уточнение имени произвольного столбца

Напечатайте `table.column`.

Предполагается, что:

- вместо `column` вы подставите имя столбца, которое хотите уточнить;
- вместо `table` вы подставите имя таблицы, где находится столбец, имя которого вы хотите уточнить (см. листинг 7.1 и рис. 7.1).

П Допускается применять в одной и той же команде как уточненные, так и не уточненные имена столбцов.

П Имейте в виду, что уточненные имена улучшают работу любой системы, основанной на базах данных, не только тем, что устраняют неоднозначность идентификации. Дело в том, что производительность ваших систем возрастет, если вы в *любом* запросе будете уточнять *все* имена столбцов. Мы рекомендуем поступать именно так, хотя очевидно, что в уточненных именах нет жесткой необходимости, если нет риска потери однозначности идентификации столбцов, то есть если во всей базе нет такой пары таблиц, в которой совпадут не уточненные имена пары столбцов.

П Еще одна веская причина применять только уточненные имена состоит в том, чтобы обезопасить себя от того, что какие-либо будущие коррекции структуры какой-нибудь из таблиц вашей базы внесут неоднозначность идентификации. Например, если бы, корректируя таблицу `publishers` нашей типовой базы, кто-нибудь добавил в нее столбец `zip` (почтовый индекс), то любая не уточненная ссылка (то есть ссылка на не уточненное имя столбца) на столбец `zip` в произвольном запросе, который выбирал бы строки из таблиц `authors` (эта таблица содержит столбец `zip` с самого начала) и `publishers`, не была бы однозначной.

Листинг 7.1. В этом запросе уточненные имена определяют, какой из столбцов `city`, присутствующих в таблицах `authors` и `publishers`, используется в запросе. Результат исполнения запроса см. на рис. 7.1

Листинг											
<pre>SELECT au_id, authors.city FROM authors INNER JOIN publishers ON authors.city = publishers.city;</pre>											
<table> <tr> <th>au_id</th><th>city</th></tr> <tr> <td>-----</td><td>-----</td></tr> <tr> <td>A03</td><td>San Francisco</td></tr> <tr> <td>A04</td><td>San Francisco</td></tr> <tr> <td>A05</td><td>New York</td></tr> </table>	au_id	city	-----	-----	A03	San Francisco	A04	San Francisco	A05	New York	
au_id	city										
-----	-----										
A03	San Francisco										
A04	San Francisco										
A05	New York										

Рис. 7.1. Результат исполнения запроса, представленного в листинге 7.1. Мы видим распечатку идентификаторов тех авторов, учитываемых в нашей типовой базе данных, для каждого из которых найдется хотя бы один издатель, живущий в том же городе, что и автор



Отдельные коммерческие СУБД могут потребовать дополнительных префиксов-уточнителей в зависимости от того, в каком месте иерархической структуры данной СУБД хранится ваш запрос. Может так случиться, что вам придется обозначать какую-нибудь таблицу именем сервера, именем базы и именем владельца. Поэтому в тех запросах SQL, которые требуют длинных уточненных имен, полезно применять псевдонимы таблиц (об этом подробно рассказывается в следующем разделе). Например, любое полностью уточненное имя таблицы в среде СУБД Microsoft SQL Server выглядит так:

```
server_name.db_name.owner_name.
→ table_name
```

Предполагается, что:

- вместо *server_name* вы подставите имя вашего сервера;
- вместо *db_name* вы подставите имя той базы данных, к которой относится таблица;
- вместо *owner_name* вы подставите имя владельца базы данных;
- вместо *table_name* вы подставите имя таблицы.

Oracle 8i требует, чтобы объединения строились по синтаксису предложения `WHERE` (см. раздел «Создание объединений с помощью синтаксиса `JOIN` и `WHERE`» в этой главе). Oracle 9i и более поздних версий поддерживает синтаксис `JOIN`, следовательно, запрос, который представлен в листинге 7.1, будет функционировать без изменений. Но для того, чтобы запустить этот запрос в Oracle 8i, вам придется его модифицировать следующим образом:

```
SELECT au_id, authors.city
FROM authors, publishers
WHERE authors.city =
→ publishers.city;
```

П Уточнение имен столбцов не принесет никакого вреда и в тех запросах, которые работают с одной таблицей. На самом деле все СУБД устроены таким образом, что каждый столбец всегда неявно имеет свой префикс-уточнитель (его иногда называют ранговой переменной). Поэтому два следующих запроса эквивалентны:

```
SELECT
    au_fname,
    au_lname
FROM authors;

SELECT
    authors.au_fname,
    authors.au_lname
FROM authors;
```

Создание псевдонимов таблиц с помощью предложения AS

Создавать псевдонимы таблиц с использованием предложения AS следует так же, как псевдонимы столбцов (см. раздел «Создание псевдонимов столбцов с помощью предложения AS» в главе 4). Полезность псевдонимов таблиц заключается в том, что они:

- сокращают объем ввода с клавиатуры;
- придают больше порядка и стройности командам;
- существуют только в тот момент, пока выполняется та команда, где они созданы;
- никогда не появляются в результате (в отличие от псевдонимов столбцов);
- не меняют имен таблиц в базе данных;
- в контексте подзапросов называются корреляционными именами (о подзапросах читайте в главе 8).

Создание произвольного псевдонима произвольной таблицы

В предложении FROM или JOIN напечатайте следующее:

```
table [AS] alias
```

Предполагается, что:

- вы замените *table* именем той таблицы, для которой намерены создать псевдоним;
- вы замените *alias* собственно псевдонимом, который должен быть одним словом, содержащим только или/и буквы, или/и цифры, или/и знаки подчеркивания (пробелы, знаки пунктуации или специальные знаки не допускаются).

- Хотя предложение повышает читаемость текстов программ, ключевое слово AS печатать не обязательно (см. листинг 7.2 и рис. 7.2).

П При создании псевдонимов таблиц мы опускаем ключевое слово AS. Дело в том, что это необходимо для максимальной совместимости представленных в книге запросов с рассматриваемыми в ней коммерческими СУБД (см. примечание **DBMS** в этом разделе).

П Как показывает практика, псевдонимы таблиц обычно выбирают очень короткими, один-два знака, не более. Однако никаких ограничений на их длину нет.

С Если хотите применить фактическое имя любой таблицы, просто опустите ее псевдоним.

П Имейте в виду, что, если псевдоним какой-нибудь таблицы создан, он заменяет фактическое имя этой таблицы в своем запросе. Это значит, что, если некоей таблице присвоен псевдоним, надо использовать его во всех уточненных ссылках (то есть в уточненных именах столбцов). Например, следующий запрос недопустим:

```
SELECT authors.au_id
FROM authors a;
```

П Каждый псевдоним таблицы должен быть уникальным в своей команде SQL.

П В каждом самообъединении от псевдонимов таблиц требуется указать на одну и ту же таблицу более одного раза. О самообъединениях читайте далее в разделе «Создание самообъединения».

Листинг 7.2. Псевдонимы таблиц делают тексты запросов короче и читабельнее. Обратите внимание, что применять псевдонимы запросов в предложении `SELECT` можно *до того*, как они фактически описаны в этом предложении. Результат исполнения запроса показан на рис. 7.2.

```

Листинг
SELECT au_fname, au_lname, a.city
FROM authors a
INNER JOIN publishers p
ON a.city = p.city;

```

au_fname	au_lname	city
-----	-----	-----
Hallie	Hull	San Francisco
Klee	Hull	San Francisco
Christian	Kells	New York

Рис. 7.2. Результат исполнения запроса, представленного в листинге 7.2.

П Чтобы приписать псевдонимы представлениям, тоже можно применить предложение `AS` (о представлениях читайте в главе 12).

П Применять ключевые и зарезервированные слова в качестве псевдонимов таблиц нельзя (см. раздел «Синтаксис SQL» в главе 3).



В среде СУБД Oracle, создавая псевдоним любой таблицы, необходимо опускать ключевое слово `AS`.

Oracle 8i требует, чтобы объединения строились по синтаксису предложения `WHERE` (см. раздел «Создание объединений с помощью синтаксиса `JOIN` и `WHERE`» в этой главе). Oracle 9i и более поздних версий поддерживает синтаксис `JOIN`, следовательно, запрос, который представлен в листинге 7.2, сработает так же, без изменений. Но для того, чтобы проверить выполнение этого запроса в Oracle 8i, вам придется его модифицировать следующим образом:

```

SELECT
    a.au_fname, a.au_lname, a.city
FROM authors a, publishers p
WHERE a.city = p.city;

```

Использование объединений

Любой запрос, который выбирает данные из более чем одной таблицы, выполняет какое-либо объединение. В следующих разделах мы подробно расскажем о различных типах объединений (см. табл. 7.1), о том, зачем нужен каждый из этих типов, и о том, как на практике создавать команды SELECT, которые будут применять объединения.

Перечислим основные характеристики объединений:

- операнды объединения (то есть те две таблицы, которые подаются на вход) обычно называют первой таблицей и второй таблицей, но, имея дело с внешними объединениями, где порядок следования входных таблиц важен, их называют левой таблицей и правой таблицей соответственно;
- таблицы всегда объединяются построчно при выполнении всевозможных условий, определенных в вашем запросе;
- те строки, которые не соответствуют заданным условиям, могут быть как включены в объединение, так и исключены из него, в зависимости от типа этого объединения;
- значения связанных столбцов обычно проверяют на равенство (=), но вы также можете сравнивать эти значения, применяя другие операторы сравнения (<>, <, <=, >, >=);
- *объединением по равенству* называется такое объединение, в условии которого применяется оператор равенства (=), чтобы группировать строки, имеющие равные значения в определенных столбцах, называемых связанными;
- объединение по равенству – это частный случай более общего *тэта-объединения*, при котором значения связанных столбцов сравнивают с применением не только оператора равенства, но и, возможно, любого другого оператора сравнения;
- связанные (или связывающие, как их еще называют) столбцы любого объединения чаще всего оказываются связанными ключевыми столбцами, но вы сами, строя свое объединение, можете связать любые столбцы, если их типы данных совместимы (все это не имеет отношения к так называемому перекрестному объединению, при котором не должно быть никаких выделенных связанных столбцов);
- чтобы обезопасить себя от построения бессмысленных объединений, например объединения на столбцах `titles.price` и `royalties.advance`, выбирайте связывающие столбцы так, чтобы они были определены на одном домене (одно часто встречающееся условие объединения включает какой-нибудь внешний ключ одной дочерней таблицы и связанный с этим внешним ключом первичный ключ другой материнской таблицы; см. разделы «Первичные ключи» и «Внешние ключи» в главе 2);
- если какой-нибудь ключ является составным (то есть содержит несколько столбцов), можете сделать связывающими все столбцы этого ключа;
- связанные (связывающие) столбцы не обязательно должны иметь одно и то же имя (за исключением случая естественного объединения);
- чтобы выполнить слияние более двух таблиц, объединения таблиц можно вкладывать друг в друга, выстраивать в цепочку и комбинировать, но при

Таблица 7.1. Типы объединений

Тип объединения	Комментарий
Полное или перекрестное объединение (CROSS JOIN)	Группирует строки таблиц по правилу «каждая с каждой». Первая строка первой таблицы объединяется с первой строкой второй таблицы (правый бок/стык с левым боком/стыком), потом первая строка первой таблицы объединяется со второй строкой второй таблицы, и т.д. до тех пор, пока в первой таблице не закончатся строки
Естественное объединение (NATURAL JOIN)	В этом объединении по равенству связанными столбцами считаются те, которые в двух таблицах имеют одинаковые имена. Объединяются те строки, в которых все значения связанных столбцов одной таблицы попарно совпадают с соответствующими значениями связанных столбцов другой таблицы. Остальные строки из объединения исключаются
Внутреннее объединение (INNER JOIN)	В этом тэта-объединении связанными столбцами считаются те, которые заданы условиями сравнения (чаще всего они в двух таблицах имеют одинаковые имена). Объединяются те строки, в которых все значения связанных столбцов одной таблицы попарно находятся в заданном оператором сравнения отношении с соответствующими значениями связанных столбцов другой таблицы. Остальные строки из объединения исключаются. Этот вид объединения используется чаще всего
Левое внешнее объединение (LEFT OUTER JOIN)	В этом объединении сначала проводится сравнение значений связанных столбцов. Но в результате объединения включаются <i>все строки левой таблицы</i> , а не только те, для которых в правой таблице нашлись строки с удовлетворяющими сравнению значениями связанных столбцов. Если некой строке слева не нашлось ни одной строки справа, СУБД объединяет ее с искусственной строкой, состоящей из значений null
Правое внешнее объединение (RIGHT OUTER JOIN)	Является зеркальным отображением левого внешнего объединения, и в нем тоже сначала проводится сравнение значений связанных столбцов. Но в результате объединения теперь включаются <i>все строки правой таблицы</i> . Если же некой строке справа не нашлось ни одной строки слева, СУБД объединяет ее с искусственной строкой, состоящей из значений null
Полное внешнее объединение (FULL OUTER JOIN)	Включает и все строки левой таблицы, и все строки правой таблицы. Если у какой-либо строки (слева или справа) нет пары по связанным столбцам, СУБД объединяет ее с искусственной строкой, состоящей из значений null
Тривиальное объединение, самообъединение (SELF JOIN)	Объединение произвольной таблицы с самой собой

этом надо понимать, что СУБД проходит через ваш запрос шаг за шагом, выполняя объединения так, что за один раз всегда обрабатываются только две таблицы (но в каждом конкретном объединении этими двумя таблицами могут быть две таблицы базы данных, одна таблица базы данных и одна таблица, являющаяся результатом предыдущего объединения, две таблицы, являющиеся результатами предыдущих объединений, и т.д.);

- хотя SQL не ограничивает количество тех таблиц (или объединений), которые могут появиться в одном запросе, на практике или ваша СУБД будет иметь встроенные ограничения, или администратор базы данных наложит такие ограничения, которые, как правило, окажутся гораздо ниже встроенных ограничений СУБД (маловероятно, что вам когда-нибудь понадобится объединить в одном запросе пять или шесть таблиц, скорее, их будет две или три);
- если связывающие столбцы содержат значения null, те строки, в которых эти значения оказываются, ни в какое объединение никогда не попадут просто потому, что значения null не могут быть равны и не равны никакому иному значению, включая другие значения null, следовательно, никакое условие объединения, зависящее от связывающего столбца со значением null в некой строке, для этой строки не может быть выполнено;
- значения null, содержащиеся в каком-нибудь столбце объединяемой таблицы, можно вывести в результат объединения,

только если этим объединением будет перекрестное или какое-нибудь внешнее объединение, за исключением того случая, когда предложение WHERE явно исключает значения null из рассмотрения (подробнее о значениях null читайте в разделе «Значение null» в главе 3);

- объединения существуют только в момент выполнения своего запроса и не являются составными элементами ни баз данных, ни СУБД;
- так как типы данных связывающих столбцов должны быть совместимы в том смысле, что СУБД могла бы при необходимости преобразовать данные этих столбцов в какой-нибудь общий тип и сравнить, вам следует иметь в виду следующее (см. раздел «Типы данных» в главе 3):
 - для большинства коммерческих СУБД совместимыми являются числовые типы данных (INTEGER, NUMERIC и FLOAT), символьные типы данных (CHAR, VARCHAR) и типы данных даты и времени (DATE, TIMESTAMP);
 - поскольку преобразования данных требуют значительной дополнительной вычислительной работы и сильно замедляют процесс исполнения запроса, для максимальной производительности необходимо назначать связывающими столбцами в двух таблицах те столбцы, типы данных которых не просто попарно совместимы, а идентичны, включая одинаковое решение вопроса о допустимости/недопустимости значений null;

Таблица 7.2. Операции по обработке наборов строк

Операция	Результат
UNION	Все строки результатов обоих запросов. Дубликаты удалены
INTERSECT	Все строки, являющиеся общими как для результата первого запроса, так и для результата второго запроса. Дубликаты удалены
EXCEPT	Все строки результата первого запроса без тех строк результата первого запроса, которые одновременно являются строками результата второго запроса. Дубликаты удалены

- чтобы запросы исполнялись быстрее, следует индексировать связывающие столбцы (см. главу 11);
- представления можно объединять с таблицами и с другими представлениями;
- чтобы создать любое объединение, можете применить или синтаксис JOIN, или синтаксис WHERE (см. следующий раздел);
- в конце этой главы рассказывается об операциях SQL на наборах строк (см. табл. 7.2), которые, не являясь объединениями, позволяют объединять результаты двух разных запросов в один общий результат.

Создание объединений с помощью синтаксиса JOIN и WHERE

Для создания произвольного объединения предусмотрены два способа:

- применить синтаксис JOIN;
- применить синтаксис WHERE.

Стандарт ANSI SQL-92 предписывает применять синтаксис JOIN, но более старые стандарты SQL рекомендуют применять синтаксис WHERE. Отсюда следует, что оба синтаксиса законны для большинства коммерческих СУБД.

Поскольку оба синтаксиса широко применяются на практике, приведем примеры, в которых можно использовать и JOIN, и WHERE. При этом запросы, применяющие синтаксис JOIN, будут представлены в основном тексте, а соответствующие им запросы, применяющие синтаксис WHERE, — в советах.

В данном разделе рассматриваются оба синтаксиса в приложении к объединениям двух таблиц. Отметим, что тот синтаксис, который вы будете применять, строя реальные запросы, будет меняться в зависимости от типа объединения, числа связываемых столбцов, числа объединяемых таблиц и ограничений, налагаемых на синтаксис конкретной СУБД.

Создание произвольного объединения с использованием синтаксиса JOIN

Напечатайте:

```
SELECT columns
FROM table1 join_type table2
ON join_conditions
[WHERE search_condition]
[GROUP BY grouping_conditions]
[HAVING search_condition]
[ORDER BY sort_columns];
```

Предполагается, что:

- вместо *columns* вы подставите одно или несколько выражений, разделенных запятыми, с именами столбцов или/и самих имен (заголовков) столбцов объединяемых таблиц *table1* и *table2*;
- во избежание неоднозначной идентификации столбцов вы уточните соответствующими префиксами все ссылки на те имена столбцов объединяемых таблиц, которые являются в них общими (см. раздел «Уточнение имен столбцов» в этой главе);
- вместо *table1* и *table2* вы подставите имена объединяемых таблиц (можете присвоить этим именам псевдонимы так, как это описано в разделе «Создание псевдонимов таблиц с помощью предложения AS»);
- вместо *join_type* вы подставите идентификатор того типа объединения, который вам нужен:
 - CROSS JOIN — для перекрестного объединения;
 - NATURAL JOIN — для естественного объединения;
 - INNER JOIN — для внутреннего объединения;
 - LEFT OUTER JOIN — для левого внешнего объединения;
 - RIGHT OUTER JOIN — для правого внешнего объединения;
 - FULL OUTER JOIN — для полного внешнего объединения;
- вместо *join_conditions* вы подставите одно или несколько условий объединения, которые следует вычислять для каждой пары объединяемых строк, и будете иметь в виду, что:
 - предложение ON не допускается для перекрестных и естественных объединений;

- любое условие объединения всегда принимает следующую форму:

`[table1.]column op [table2.]column`,
где:

`[table1.]column` — имя, возможно уточненное, связывающего столбца первой таблицы;

`[table2.]column` — имя, возможно уточненное, связывающего столбца второй таблицы, соотносящегося со столбцом `[table1.]column`;

`op` — оператор сравнения, которым чаще всего является оператор равенства (=), но может быть и любой оператор =, <>, <, <=, > или >= (подробнее об операторах сравнения см. в табл. 4.2);

- вы можете комбинировать несколько условий объединения с помощью логических операторов AND и OR (см. раздел «Комбинирование условий с помощью операторов AND, OR и NOT» в главе 4);
- предложения WHERE, ORDER BY описаны в главе 4, а предложения GROUP BY и HAVING — в главе 6.

Создание произвольного объединения с использованием синтаксиса WHERE

Напечатайте:

```
SELECT columns
FROM table1, table2
WHERE join_conditions
[GROUP BY grouping_conditions]
[HAVING search_condition]
[ORDER BY sort_columns];
```

Предполагается, что:

- `columns`, `table1`, `table2` означают то же самое, что и в подразделе «Создание произвольного объединения с использованием синтаксиса JOIN»;

- `join_conditions` означает то же самое, что и в подразделе «Создание произвольного объединения с использованием синтаксиса JOIN», только теперь вместо `op` можно подставлять специальный символ, означающий тип объединения;

- предложение WHERE может включать еще и условия поиска (не имеющие прямого отношения к объединению таблиц), необходимые для фильтрации строк (см. раздел «Фильтрация строк с помощью предложения WHERE» в главе 4);

- вы уже умеете строить предложения ORDER BY (см. главу 4), GROUP BY и HAVING (см. главу 6).

Далее рассмотрим листинги, в которых представлены запросы, одинаково решающие одну и ту же задачу, но с использованием синтаксиса JOIN и WHERE соответственно. Общий результат этих запросов показан на рис. 7.3.

П

На первый взгляд может показаться странным, что предложение WHERE, по идее, являющееся инструментом фильтрования, применяется для того, чтобы указать условия объединения. Однако ничего странного здесь нет. Дело в том, что всякое условие объединения *работает* как фильтр. Когда вы объединяете две таблицы, СУБД неявно и незаметно для вас начинает с того, что спаривает каждую строку первой таблицы с каждой строкой второй таблицы, то есть строит перекрестное объединение (см. следующий раздел). И только после этого СУБД фильтрует строки перекрестного объединения условиями объединения (очевидно, что на практике слишком дорого строить огромное и бесполезное перекрестное объединение для каждого объединения и что никакая СУБД так делать не будет, для этого существуют оптимизаторы).

П

Одно очевидное преимущество синтаксиса JOIN перед синтаксисом WHERE состоит в том, что первый явно объявляет тип объединения. Конечно, A LEFT OUTER JOIN B гораздо понятнее, чем, скажем, A *= B. Однако в отношении самых ходовых объединений, то есть для простых внутренних объединений, синтаксис WHERE, наоборот, проще синтаксиса JOIN. В любом случае оба синтаксиса одинаково широко распространены среди программистов. Поэтому, если вы хотите читать и понимать запросы, написанные не только вами, но и другими специалистами, придется выучить оба синтаксиса.

П

В любом трехтабличном объединении только одну таблицу можно использовать в качестве моста от двух первых таблиц к третьей. В последующих разделах приводятся специальные примеры, поясняющие это правило.

П

Перечень столбцов в предложении SELECT для любого объединения может относиться ко всему множеству столбцов объединяемых таблиц или к любому подмножеству этого множества. Значит, в указанный перечень вовсе не обязательно включать представителей от всех таблиц, участвующих в объединении. Например, нет такого правила, чтобы в любом трехтабличном объединении в этот перечень обязательно были включены столбцы из средней таблицы.

П

Связывающие столбцы могут иметь разные имена.

Листинг 7.3а. Запрос, который применяет синтаксис JOIN. Результат выполнения запроса см. на рис. 7.3

```

Листинг
SELECT au_fname, au_lname, a.city
FROM authors a
INNER JOIN publishers p
ON a.city = p.city;

```

Листинг 7.3б. Тот же самый запрос, но применяющий синтаксис WHERE. Результат выполнения запроса см. на рис. 7.3

```

Листинг
SELECT au_fname, au_lname, a.city
FROM authors a, publishers p
WHERE a.city = p.city;

```

au_fname	au_lname	city
Hallie	Hull	San Francisco
Klee	Hull	San Francisco
Christian	Kells	New York

Рис. 7.3. Результат выполнения листингов 7.3а и 7.3б

Последовательность исполнения запроса

Когда СУБД обрабатывает объединение, она подчиняется определенной последовательности шагов, которая не только относится к обработке объединений, но и определяет алгоритм выполнения всего запроса. Вот эта последовательность:

1. Применить условия объединения, заданные предложением JOIN.
2. Применить условия объединения и поиска, заданные предложением WHERE.
3. Сгруппировать строки в соответствии с предложением GROUP BY.
4. Применить к группам условия поиска, заданные предложением HAVING.
5. Отсортировать результат в соответствии с предложением ORDER BY.



Если вы применяете синтаксис WHERE с двумя или большим количеством условий объединения, то почти наверняка захотите объединить все эти условия логическим оператором AND. Здесь мы имеем в виду следующее: хотя объединять условия логическим оператором OR вполне законно, смысл результирующего условия обычно трудно понять. Поэтому применяйте лучше оператор AND. Подробнее об операторах AND и OR читайте в разделе «Комбинирование условий с помощью операторов AND, OR и NOT» главы 4.



Большинство запросов, применяющих объединения, можно переписать с помощью подзапросов (то есть запросов, вложенных в другие запросы). Но верно и то, что большинство подзапросов можно переписать как объединения. Подробнее о подзапросах читайте в главе 8.



СУБД Oracle 8i и более ранних версий не поддерживает синтаксис JOIN. Если у вас именно такая СУБД, применяйте синтаксис WHERE. СУБД Oracle 9i поддерживает синтаксис JOIN.

Ваша СУБД может запретить объединения по связывающим столбцам определенных типов данных (почти наверняка под запрет попадут бинарные типы). Например, в среде Microsoft SQL Server под такой запрет попадают столбцы типов ntext, text и image, а в среде СУБД Oracle – столбцы типа BLOB. Чтобы полностью разобраться в этом вопросе, вам следует в документации по своей СУБД организовать поиск по ключу «joins» (объединения).



Связывающие столбцы могут относиться к разным типам данных. Но, если их типы не идентичны, они должны быть совместимыми или такими, чтобы ваша СУБД при необходимости смогла конвертировать их друг в друга неявно. Если же эти типы данных СУБД не может конвертировать неявно, вы сами должны конвертировать явно с использованием CAST() в тексте запроса, то есть в условиях объединения. Подробнее о явных и неявных преобразованиях типов говорится в разделе «Преобразование типов данных с помощью функции CAST()» главы 5.

Предложение USING

Стандартный SQL для синтаксиса JOIN допускает, чтобы вместо предложения ON применялось предложение USING, но только в том случае, когда все связывающие столбцы имеют попарно одинаковые имена (то есть на концах каждой связи должны стоять столбцы с одинаковыми именами) и сравниваются строго на равенство. Тогда синтаксис JOIN будет таким:

```
FROM table1 joint_type table2
    USING (columns)
```

Здесь под *columns* подразумевается одно или несколько разделенных запятыми имен столбцов, что означает одну или несколько пар одинаковых имен связывающих столбцов. Круглые скобки обязательны. Соответствующий запрос выполнит объединение по равенству поименованной пары или поименованных пар столбцов. Такой тип объединения принято называть *объединением поименованных столбцов*. С предложением USING запрос, который представлен в листинге 7.3а, будет переписан следующим образом:

```
SELECT au_fname, au_lname, a.city
    INNER JOIN publishers p
    FROM authors a
    USING (city);
```

Очевидно, что предложение USING работает так же, как естественное объединение, но если вы не хотите заносить в связывающие столбцы *все* столбцы с общими именами, а хотите выбрать из всех столбцов с общими именами *только некоторые* и сделать их связывающими, то можете воспользоваться предложением USING. Обратите внимание на то обстоятельство, что в нашем примере в качестве связывающего выбран только столбец *city*, хотя он вовсе не единственный столбец с общим именем в двух таблицах, и естественное объединение включило бы автоматически в состав связывающих помимо него еще и столбец *state* (см. раздел «Создание произвольного естественного объединения с использованием предложения NATURAL JOIN» в этой главе).

Необходимо отметить, что предложение USING не создает в рамках SQL никакой дополнительной функциональности. Абсолютно все, что можно сделать с помощью предложения USING, можно сделать с помощью предложения ON в рамках синтаксиса JOIN или с помощью предложения WHERE в рамках синтаксиса WHERE.



СУБД Microsoft Access и Microsoft SQL Server не поддерживают предложение USING.

СУБД Oracle 9i не допускает уточненных имен столбцов в списке SELECT для тех запросов с объединениями, в которых применяется предложение USING. Поэтому, чтобы реализовать в среде Oracle 9i только что приведенный пример с предложением USING, в предложении SELECT вам нужно поменять *a.city* на *city*.

Создание произвольного перекрестного объединения с использованием предложения CROSS JOIN

Перечислим основные характеристики перекрестного объединения:

- любое перекрестное объединение всегда выдает все возможные комбинации строк двух таблиц так, что *каждая* строка первой таблицы объединяется с *каждой* строкой второй таблицы;
- никакое перекрестное объединение никогда не применяет никаких условий объединения. Поэтому, если вы используете синтаксис JOIN и хотите создать некое перекрестное объединение, просто опустите предложение ON, а если вы применяете синтаксис WHERE и хотите создать некое перекрестное объединение, опустите все предложение WHERE;
- перекрестные объединения редко применяют на практике, потому что их результаты трудно интерпретировать;
- перекрестные объединения могут выдавать огромные результаты даже при обработке маленьких таблиц, просто потому, что, если первая таблица содержит m строк, а вторая таблица содержит n строк, перекрестное объединение этих таблиц будет по определению состоять из $m \times n$ строк;
- перекрестное объединение требует слишком больших затрат на любой запрос (имеются в виду вычислительные ресурсы и время);
- перекрестное объединение еще называют *Декартовым*, или *прямым произведением*.

Создание произвольного перекрестного объединения

Напечатайте:

```
SELECT columns
FROM table1
CROSS JOIN table2
```

Предполагается, что (см. листинг 7.4 и рис. 7.4):

- вместо *columns* вы подставите одно или несколько разделенных запятыми выражений с именами столбцов или/и самих имен (заголовков) столбцов объединяемых таблиц *table1* и *table2*;
- во избежание неоднозначной идентификации столбцов вы уточните соответствующими префиксами (то есть именами таблиц) те имена столбцов объединяемых таблиц, которые являются в них общими;
- вместо *table1* и *table2* вы подставите имена объединяемых таблиц.

П

В рамках синтаксиса WHERE тот запрос, который представлен в листинге 7.4, будет выглядеть так:

```
SELECT
    au_id, pub_id,
    a.state AS "au_state",
    p.state AS "pub_state"
FROM authors a, publishers p;
```

С

Чтобы показать все столбцы обеих таблиц в перекрестном объединении, используйте предложение SELECT * (см. раздел «Выбор столбцов с помощью предложений SELECT и FROM» в главе 4). Например, следующие запросы выводят все столбцы таблиц *authors* и *publishers*:

- для синтаксиса JOIN:

```
SELECT *
FROM authors
CROSS JOIN publishers;
```

- для синтаксиса WHERE:

```
SELECT *
FROM authors, publishers;
```

С Чтобы показать все столбцы только одной какой-либо таблицы в перекрестном объединении, используйте предложение `SELECT table.*`. Например, следующие запросы выводят все столбцы таблицы `authors` и только один столбец `pub_id` из таблицы `publishers`:

- для синтаксиса JOIN:

```
SELECT authors.*, p.pub_id
FROM authors
CROSS JOIN publishers p;
```

- для синтаксиса WHERE:

```
SELECT authors.*, p.pub_id
FROM authors, publishers p;
```

С Чтобы найти прямое произведение N таблиц, напечатайте:

- для синтаксиса JOIN:

```
SELECT columns
FROM table1
CROSS JOIN table2
...
CROSS JOIN tableN;
```

- для синтаксиса JWHERE:

```
SELECT columns
FROM table1, table2, ..., tableN
```

П Имейте в виду, что прямые произведения очень часто получаются по ошибке. Поэтому, если результат вашего запроса содержит неожиданно большое число строк, это может быть следствием того, что вы где-то забыли напечатать условия объединения.

Листинг 7.4. Создать одно перекрестное объединение и распечатать все возможные комбинации строк двух таблиц нашей типовой базы данных. Результат исполнения запроса показан на рис. 7.4

Листинг

```
SELECT
    au_id,
    pub_id,
    a.state AS "au_state",
    p.state AS "pub_state"
FROM authors a
CROSS JOIN publishers p;
```

au_id	pub_id	au_state	pub_state
-----	-----	-----	-----
A01	P01	NY	NY
A02	P01	CO	NY
A03	P01	CA	NY
A04	P01	CA	NY
A05	P01	NY	NY
A06	P01	CA	NY
A07	P01	FL	NY
A01	P02	NY	CA
A02	P02	CO	CA
A03	P02	CA	CA
A04	P02	CA	CA
A05	P02	NY	CA
A06	P02	CA	CA
A07	P02	FL	CA
A01	P03	NY	NULL
A02	P03	CO	NULL
A03	P03	CA	NULL
A04	P03	CA	NULL
A05	P03	NY	NULL
A06	P03	CA	NULL
A07	P03	FL	NULL
A01	P04	NY	CA
A02	P04	CO	CA
A03	P04	CA	CA
A04	P04	CA	CA
A05	P04	NY	CA
A06	P04	CA	CA
A07	P04	FL	CA

Рис. 7.4. Результат исполнения запроса, представленного в листинге 7.4

П

Хотя прямое произведение таблиц почти никогда не является тем, чего вы добиваетесь, ваша СУБД (теоретически) применяет его часто. Считается, что СУБД, приступая к обработке любого объединения, сначала неявно создает перекрестное объединение соответствующих таблиц, а потом применяет предложение `SELECT`, чтобы удалить из прямого произведения столбцы, которых нет в перечне этого предложения, и условия объединения и поиска, чтобы удалить из прямого произведения лишние строки.

П

Объединение `t1 CROSS JOIN t2` эквивалентно любому из следующих объединений:

- `t1 INNER JOIN t2 ON 1 = 1;`
- `t1 LEFT OUTER JOIN t2 ON 1 = 1;`
- `t1 RIGHT OUTER JOIN t2 ON 1 = 1;`
- `t1 FULL OUTER JOIN t2 ON 1 = 1.`

Здесь `t1` и `t2` – таблицы, а `1 = 1` – любое условие, которое всегда выполняется. О внутренних и внешних объединениях рассказывается в следующих разделах.

П

Одно практическое применение перекрестного объединения состоит в том, чтобы производить наборы данных для тестирования программного обеспечения. Предположим, что у вас есть некая функция от n аргументов и для каждого из них существует m эталонных контрольных значений. Так вот, вам понадобится таблица значений $n \times m$, каждый столбец которой будет представлять собой один контрольный набор всех аргументов вашей функции. Эту таблицу вы легко сможете получить, организовав прямое произведение из n таблиц (по одной таблице на каждый аргумент), в каждой из которых есть только один столбец и лишь m строк (в каждой строке – одно контрольное значение для соответствующего аргумента). Этот метод будет работать, даже если m меняется от аргумента к аргументу.



СУБД Microsoft Access поддерживает для перекрестных объединений только синтаксис `WHERE`. Поэтому, чтобы запустить в этой СУБД запрос, который представлен в листинге 7.4, воспользуйтесь командой из самого первого примечания настоящего раздела.

СУБД Oracle 8i и более ранних версий не поддерживает синтаксис `JOIN`. Поэтому, если у вас именно такая СУБД, применяйте синтаксис `WHERE`. СУБД Oracle 9i и более поздних версий поддерживает синтаксис `JOIN`.

Создание произвольного естественного объединения с использованием предложения NATURAL JOIN

Любое естественное, или натуральное, объединение обладает следующими характеристиками:

- является частным случаем объединения по равенству, и в нем связанными столбцами считаются те, которые в двух таблицах имеют одинаковые имена. Объединяются те строки, в которых все значения связанных столбцов одной таблицы попарно совпадают с соответствующими значениями связанных столбцов другой таблицы. Остальные строки из объединения исключаются;
- работает только в том случае, если во входных (то есть объединяемых) таблицах найдется хотя бы одна пара столбцов с одинаковыми именами, чтобы их типы данных были совместимы в разумных пределах;
- выполняет объединения неявно. Вам не надо определять для естественного запроса ни предложение ON, ни предложение USING;
- все, что может делать это объединение, выполняется явно или с помощью предложения ON в синтаксисе JOIN, или с помощью предложением WHERE в синтаксисе WHERE.

Создание произвольного естественного объединения

Напечатайте:

```
SELECT columns
FROM table1
NATURAL JOIN table2
```

Предполагается, что:

- вместо *columns* вы подставите одно или несколько разделенных запятыми выражений с именами столбцов или/и самих имен (заголовков) столбцов объединяемых таблиц *table1* и *table2*;
- если этого потребует СУБД и для того, чтобы избежать неоднозначной идентификации столбцов, вы уточните соответствующими префиксами (то есть именами таблиц) имена столбцов объединяемых таблиц, которые являются в них общими (см. примечание **DBMS** в этом разделе);
- вместо *table1* и *table2* подставите имена объединяемых таблиц.

Естественное объединение выполняется таким образом, что связующими объявляются те столбцы, у которых имеется пара с таким же именем, но в другой таблице. В каждой паре объединяемых строк значения этих пар столбцов сравниваются на равенство и, если оно по всем парам столбцов с одинаковыми именами достигается, строки объединяются.

Предложение NATURAL JOIN создает естественные внутренние объединения. О том, как создавать естественные внешние объединения, упоминается в одном из советов этого раздела.

Когда СУБД будет исполнять тот запрос, который представлен в листинге 7.5, она

Листинг 7.5. Распечатать перечень книг, учитываемых в нашей типовой базе данных, и издателей этих книг так, чтобы рядом с идентификатором книги были показаны в одной строке идентификатор и наименование издателя, опубликовавшего эту книгу. Результат исполнения запроса показан на рис. 7.5

```

SELECT
    t.title_id,
    t.pub_id,
    p.pub_name
FROM publishers p
NATURAL JOIN titles t;

```

title_id	pub_id	pub_name
T01	P01	Abatis Publishers
T02	P03	Schadenfreude Press
T03	P02	Core Dump Books
T04	P04	Tenterhooks Press
T05	P04	Tenterhooks Press
T06	P01	Abatis Publishers
T07	P03	Schadenfreude Press
T08	P04	Tenterhooks Press
T09	P04	Tenterhooks Press
T10	P01	Abatis Publishers
T11	P04	Tenterhooks Press
T12	P01	Abatis Publishers
T13	P03	Schadenfreude Press

Рис. 7.5. Результат исполнения запроса, представленного в листинге 7.5

объединит строки таблицы *publishers* со строками таблицы *titles*, причем произвольная пара строк объединяется тогда и только тогда, когда значения столбцов *publishers.pub_id* и *titles.pub_id*, то есть единственной пары столбцов, имеющих в обеих таблицах одно и то же имя (не уточненное; уточненные имена всегда различны!), на этой паре строк равны. Результат исполнения этого запроса показан на рис. 7.5.

Листинг 7.6 представляет модифицированный запрос из листинга 7.5. Суть этой модификации состоит в том, что в запрос из листинга 7.5 мы добавили еще одно объединение, чтобы сопроводить упоминание каждой книги данными по сумме аванса, выплаченного за нее автору, и предложение *WHERE*, которое разрешает выбирать только те книги, сумма аванса за которые меньше \$20 000. Когда ваша СУБД будет исполнять этот модифицированный запрос, она сначала выполнит объединение таблиц *publishers* и *titles*, причем связывающими столбцами здесь будут *publishers.pub_id* и *titles.pub_id*, и получит виртуальную таблицу, представленную на рис. 7.5. После этого она объединит эту самую виртуальную таблицу с таблицей *royalties*, причем связывающими столбцами будут *titles.title_id* и *royalties.title_id*. Потом СУБД удалит из результата все строки, в которых сумма аванса составляет не меньше \$20 000 (см. рис. 7.6).

П Чтобы заменить любое естественное объединение каким-либо эквивалентным объединением, но в синтаксисе *WHERE*, следует воспользоваться объединением по равенству с соответствующим предложением *WHERE*, которое применяет оператор *AND*

для комбинации нескольких (по числу пар столбцов с одинаковыми именами) условий равенства значений связывающих столбцов в единое условие. Например, для запросов из листингов 7.5 и 7.6 эквивалентные представления в синтаксисе WHERE таковы:

- ```
SELECT t.title_id, t.pub_id,
 p.pub_name
FROM publishers p, titles t
WHERE p.pub_id = t.pub_id; -- для
запроса, представленного в листин-
ге 7.5;
```
- ```
SELECT t.title_id, t.pub_id,
      p.pub_name, r.advance
FROM publishers p, titles t,
      royalties r
WHERE p.pub_id = t.pub_id
      AND t.title_id = r.title_id
      AND r.advance < 20000; -- для
запроса, представленного в листин-
ге 7.6.
```

С

Чтобы заменить любое естественное объ-единение каким-либо эквивалентным внут-ренним или внешним объединением, но в синтаксисе JOIN, воспользуйтесь объе-динением по равенству с соответствующим предложением ON, которое применя-ет оператор AND для комбинации несколь-ких (по числу пар столбцов с одинаковы-ми именами) условий равенства значений связывающих столбцов в единое условие. Например, для запросов из листингов 7.5 и 7.6 эквивалентные представления в син-таксисе JOIN таковы:

- ```
SELECT t.title_id, t.pub_id,
 p.pub_name
FROM publishers p
INNER JOIN titles t
 ON p.pub_id = t.pub_id; -- для за-
проса, представленного в листинге 7.5;
```

**Листинг 7.6.** Перечислить идентификаторы для книг, учитываемых в нашей типовой базе данных, авторы которых получили аванс меньше \$20 000, в сочетании с идентификатором издателя, напечатавшего каждую книгу, именем этого издателя и суммой аванса. Результат исполнения запроса представлен на рис. 7.6

Листинг

```
SELECT
 t.title_id,
 t.pub_id,
 p.pub_name,
 r.advance
FROM publishers p
NATURAL JOIN titles t
NATURAL JOIN royalties r
WHERE r.advance < 20000;
```

| title_id | pub_id | pub_name            | advance |
|----------|--------|---------------------|---------|
| T01      | P01    | Abatis Publishers   | 10000   |
| T02      | P03    | Schadenfreude Press | 1000    |
| T03      | P02    | Core Dump Books     | 15000   |
| T08      | P04    | Tenterhooks Press   | 0       |
| T09      | P04    | Tenterhooks Press   | 0       |

**Рис. 7.6.** Результат исполнения листинга 7.6

```

■ SELECT t.title_id, t.pub_id,
 p.pub_name, r.advance
FROM publishers p
INNER JOIN titles t
 ON p.pub_id = t.pub_id
INNER JOIN royalties r
 ON t.title_id = r.title_id
WHERE r.advance < 20000; — для
запроса, представленного в листинге 7.6.

```

**П**

Вы можете заменить любое естественное объединение еще каким-нибудь эквивалентным объединением в синтаксисе JOIN, которое применяет предложение USING (см. врезку в разделе «Создание объединений с помощью синтаксиса JOIN и WHERE»). При этом нужно понимать, что NATURAL JOIN — это сокращенная и упрощенная форма предложения USING, предложение NATURAL JOIN тоже формирует список типа USING, но в него включаются только те столбцы, у которых есть пара во второй таблице (то есть столбец с таким же именем). Например, запросы из листингов 7.5 и 7.6 с использованием предложения USING можно переписать в следующем виде:

```

■ SELECT t.title_id, t.pub_id,
 p.pub_name
FROM publishers p
INNER JOIN titles t
 USING (pub_id); — для запроса,
представленного в листинге 7.5;

■ SELECT t.title_id, t.pub_id,
 p.pub_name, r.advance
FROM publishers p
INNER JOIN titles t
 USING (pub_id)
INNER JOIN royalties r
 USING (title_id)
WHERE r.advance < 20000; — для за-
проса, представленного в листинге 7.6.

```

**П**

Имейте в виду, что синтаксис NATURAL JOIN фактически создает некое внутреннее объединение, и именно поэтому предложения NATURAL JOIN и NATURAL INNER JOIN полностью эквивалентны, а ключевое слово INNER всегда опускают при построении соответствующего объединения. Но вы можете создавать не только естественные внутренние, но и естественные внешние объединения с использованием следующих предложений:

```

■ NATURAL LEFT [OUTER] JOIN;
■ NATURAL RIGHT [OUTER] JOIN;
■ NATURAL FULL [OUTER] JOIN.

```

**С**

Прежде чем применять всякое естественное объединение, проверьте, что все связывающие столбцы имеют в двух таблицах одни и те же неуточненные имена и все столбцы объединяемых таблиц, не являющиеся связывающими, имеют уникальные неуточненные имена.

**П**

Обратите внимание, что смысл естественного объединения в реляционной модели (см. главу 2) и в стандарте SQL разный. Если в реляционной модели естественным объединением считается такое объединение дочерней и материнской таблиц, когда единственной парой связывающих столбцов является внешний ключ дочерней таблицы и первичный ключ материнской таблицы, то в SQL под естественным объединением понимается такое объединение двух произвольных таблиц, когда связывающими столбцами являются *все* столбцы двух таблиц, имеющие в двух таблицах одинаковые неуточненные имена. В качестве примера естественного объединения, в состав связывающих столбцов которого ни один ключ не входит, рассмотрим запрос из листинга 7.9. Отсюда следует:

чтобы соответствующие определения реляционной модели и SQL совпали, все внешние ключи в базе должны иметь те же имена, что и родительские первичные ключи, а все остальные столбцы – только уникальные имена.



СУБД Microsoft Access и Microsoft SQL Server не поддерживают синтаксис NATURAL JOIN. Поэтому, чтобы запустить в этих СУБД запросы, которые представлены в листингах 7.5 и 7.6, используйте синтаксис WHERE (см. самое первое примечание этого раздела).

СУБД Oracle 8i и более ранних версий вообще не поддерживает синтаксис JOIN. Поэтому вам придется полностью перейти на синтаксис WHERE.

СУБД Oracle 9i не разрешает применять уточненные имена столбцов в предложениях SELECT, включающих естественные объединения. Поэтому, чтобы запустить в этой СУБД запросы, представленные в листингах 7.5 и 7.6, придется убрать все префиксы-уточнители следующим образом:

- ```
SELECT title_id, pub_id, pub_name
FROM publishers
NATURAL JOIN titles; – для за-
проса, представленного в листинге 7.5;
```
- ```
SELECT title_id, pub_id,
pub_name, advance
FROM publishers
NATURAL JOIN titles
NATURAL JOIN royalties
WHERE advance < 20000; – для за-
проса, представленного в листинге 7.6.
```

СУБД Oracle 9i не разрешает применять уточненные имена столбцов в предложениях SELECT тех запросов, где в объединениях применяется предложение USING. Поэтому, чтобы запустить в среде Oracle 9i запросы, которые мы привели как примеры использования предложения USING, замените `t.title_id` на `title_id` и `t.pub_id` на `pub_id` и т.д.

В среде СУБД PostgreSQL применение префиксов-уточнителей в ситуации неоднозначного именования столбцов в естественных объединениях допускается, но не является обязательным.



## Создание внутреннего объединения с помощью команды INNER JOIN

Внутреннее объединение отличается следующими характеристиками:

- использует оператор сравнения (=, <>, <, <=, > или >=), чтобы сопоставить строки в двух таблицах на основании значений в общих столбцах каждой таблицы. Например, вы можете считать все строки, для которых идентификатор автора (столбец *au\_id*) в таблицах *authors* и *title\_authors* совпадает;
- считывает результат, который включает только объединенные строки, соответствующие условиям объединения;
- является самым распространенным типом объединения.

### Создание внутреннего объединения

Введите:

```
SELECT columns
FROM table1
INNER JOIN table2
ON join_conditions
```

Здесь *columns* – это несколько выражений или названий столбцов и *table1* или *table2*, разделенных запятыми. *table1* и *table2* – это названия объединенных таблиц. Если таблицы имеют столбцы с одинаковыми названиями, обозначьте их с помощью имен таблиц.

*join\_conditions* указывает условия объединения, которые должны быть рассчитаны для каждой пары объединенных строк. Условие объединения приобретает следующую форму:

```
[table1.column op [table2]column
```

*op* обычно имеет значение =, но может быть любым оператором сравнения (=, <>, <, <=, > или >=; см. табл. 4.2). Вы можете комбинировать несколько условий объединения

с помощью AND или OR (см. раздел «Комбинирование условий с помощью операторов AND, OR и NOT» в главе 4).

**П**

Чтобы создать внутреннее объединение для трех или более таблиц с помощью команды JOIN, введите:

```
SELECT columns
FROM table1
INNER JOIN table2
ON join_condition1
INNER JOIN table3
ON join_condition2
...
```

Если вы желаете использовать синтаксис WHERE, введите:

```
SELECT columns
FROM table1, table2, ...
WHERE join_condition1
AND join_condition2
...
```

**П**

По умолчанию команда JOIN (без указания CROSS, NATURAL, OUTER или любых других) эквивалентна INNER JOIN.



Oracle 8i и более ранних версий не поддерживает команду JOIN; вместо нее используйте команду WHERE. Oracle 9i и более поздних версий поддерживает команду JOIN.

В Microsoft Access можно использовать синтаксис WHERE или JOIN, но, если вы будете использовать JOIN для объединений, которые включают три таблицы или более, вам придется размещать объединения на разных уровнях с помощью следующей общей структуры:

```
SELECT columns
FROM table1
INNER JOIN (table2
INNER JOIN (table3
INNER JOIN (table4
INNER JOIN ...
ON table3.column3 op table4.column4)
ON table2.column2 op table3.column3)
ON table1.column1 op table2.column2;
```

(в других базах данных вы тоже можете размещать объединения на разных уровнях с помощью подобной структуры, но в Access вам необходимо ее использовать).

Листинг 7.7 объединяет две таблицы в столбце `au_id`, чтобы отобразить список книг, которые написали все авторы (в том числе в соавторстве). Информация о каждом авторе в столбце `au_id` соответствует строкам в таблице `title_authors`. Результат исполнения листинга показан на рис. 7.7. Обратите внимание, что автор A07 (Paddy O’Furniture) не включен в результат, потому что он не написал ни одной книги и для него нет ответов в строках `title_authors`.

**П** При использовании синтаксиса `WHERE` листинг 7.7 будет выглядеть следующим образом:

```
SELECT a.au_id, a.au_fname,
 a.au_lname, ta.title_id
FROM authors a, title_authors ta
WHERE a.au_id = ta.au_id
ORDER BY a.au_id ASC,
 ta.title_id = ASC;
```

**Листинг 7.7.** Отобразить список книг, которые написали все авторы (в том числе в соавторстве). Результат исполнения см. на рис. 7.7

Листинг

```
SELECT
 a.au_id,
 a.au_fname,
 a.au_lname,
 ta.title_id
FROM authors a
INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
ORDER BY a.au_id ASC, ta.title_id ASC;
```

| au_id | au_fname  | au_lname  | title_id |
|-------|-----------|-----------|----------|
| ----- | -----     | -----     | -----    |
| A01   | Sarah     | Buchman   | T01      |
| A01   | Sarah     | Buchman   | T02      |
| A01   | Sarah     | Buchman   | T13      |
| A02   | Wendy     | Heydemark | T06      |
| A02   | Wendy     | Heydemark | T07      |
| A02   | Wendy     | Heydemark | T10      |
| A02   | Wendy     | Heydemark | T12      |
| A03   | Hallie    | Hull      | T04      |
| A03   | Hallie    | Hull      | T11      |
| A04   | Klee      | Hull      | T04      |
| A04   | Klee      | Hull      | T05      |
| A04   | Klee      | Hull      | T07      |
| A04   | Klee      | Hull      | T11      |
| A05   | Christian | Kells     | T03      |
| A06   |           | Kellsey   | T08      |
| A06   |           | Kellsey   | T09      |
| A06   |           | Kellsey   | T11      |

**Рис. 7.7.** Результат выполнения листинга 7.7

**Листинг 7.8.** Отобразить заголовки и информацию о каждой книге, а также информацию обо всех книгах других издательств. Результат исполнения см. на рис. 7.8

Листинг

```
SELECT
 t.title_id,
 t.title_name,
 t.pub_id,
 p.pub_name
FROM titles t
INNER JOIN publishers p
 ON p.pub_id = t.pub_id
ORDER BY t.title_name ASC;
```

Листинг 7.8 объединяет две таблицы в столбце pub\_id, чтобы отобразить заголовки и информацию о каждой книге, а также информацию о всех книгах других издательств. Обратите внимание, что объединение требуется только для того, чтобы считать название издательства (четвертый столбец в результате), все другие столбцы есть в таблице titles. Результат исполнения листинга показан на рис. 7.8.



При использовании синтаксиса WHERE листинг 7.8 будет выглядеть так:

```
SELECT t.title_id, t.title_name,
 t.pub_id, p.pub_name
FROM titles t, publishers p
WHERE p.pub_id = t.pub_id
ORDER BY t.title_name ASC;
```

| title_id | title_name                          |
|----------|-------------------------------------|
| T01      | 1997!                               |
| T02      | 200 Years of German Humor           |
| T03      | Ask Your System Administrator       |
| T04      | But I Did It Unconsciously          |
| T05      | Exchange of Platitudes              |
| T06      | How About Never?                    |
| T07      | I Blame My Mother                   |
| T08      | Just Wait Until After School        |
| T09      | Kiss My Boo-Boo                     |
| T10      | Not Without My Faberge Egg          |
| T11      | Perhaps It's a Glandular Problem    |
| T12      | Spontaneouse, But Not Annoing       |
| T13      | What Are The Civilian Applications? |

| pub_id | pub_name            |
|--------|---------------------|
| P01    | Abatis Publishers   |
| P03    | Schadenfreude Press |
| P02    | Core Dump Books     |
| P04    | Tenterhooks Press   |
| P04    | Tenterhooks Press   |
| P01    | Abatis Publishers   |
| P03    | Schadenfreude Press |
| P04    | Tenterhooks Press   |
| P04    | Tenterhooks Press   |
| P01    | Abatis Publishers   |
| P04    | Tenterhooks Press   |
| P01    | Abatis Publishers   |
| P03    | Schadenfreude Press |

**Рис. 7.8.** Результат выполнения листинга 7.8

Листинг 7.9 использует два объединения, чтобы отобразить список авторов, которые живут во всех городах, где расположены издательства. Результат исполнения листинга показан на рис. 7.9. Обратите внимание, что этот запрос представляет собой объединение столбцов `city` и `state` в двух таблицах, причем эти столбцы имеют одинаковое название и не являются ключами (см. раздел «Создание произвольного естественного объединения с использованием предложения `NATURAL JOIN`» в этой главе). Приведем эквивалентный запрос:

```
SELECT a.au_id, a.au_fname, a.au_lname,
 a.city, a.state
FROM authors a
NATURAL JOIN publishers p
ORDER BY a.au_id ASC;
```

**П**

При использовании синтаксиса `WHERE` листинг 7.9 будет выглядеть следующим образом:

```
SELECT a.au_id, a.au_fname,
 a.au_lname, a.city, a.state
FROM authors a, publishers p
WHERE a.city = p.city
 AND a.state = p.state
ORDER BY a.au_id ASC;
```

**Листинг 7.9.** Отобразить список авторов, которые живут во всех городах, где расположены издательства. Результат исполнения листинга см. на рис. 7.9

Листинг

```
SELECT
 a.au_id,
 a.au_fname,
 a.au_lname,
 a.city,
 a.state
FROM authors a
INNER JOIN publishers p
 ON a.city = p.city
 AND a.state = p.state
ORDER BY a.au_id ASC;
```

| au_id | au_fname  | au_lname | city          | state |
|-------|-----------|----------|---------------|-------|
| ----  | -----     | -----    | -----         | ----- |
| A03   | Hallie    | Hull     | San Francisco | CA    |
| A04   | Klee      | Hull     | San Francisco | CA    |
| A05   | Christian | Kells    | New York      | NY    |

**Рис. 7.9.** Результат выполнения листинга 7.9

**Листинг 7.10.** Отобразить список книг, опубликованных в штате Калифорния или вне крупных стран Северной Америки. Результат исполнения см. на рис. 7.10

Листинг

```
SELECT
 t.title_id,
 t.title_name,
 p.state,
 p.country
FROM titles t
INNER JOIN publishers p
 ON t.pub_id = p.pub_id
WHERE p.state = 'CA'
 OR p.country NOT IN
 ('USA', 'Canada', 'Mexico')
ORDER BY t.title_id ASC;
```

| title_id | title_name                          |
|----------|-------------------------------------|
| T02      | 200 Years of German Humor           |
| T03      | Ask Your System Administrator       |
| T04      | But I Did It Unconsciously          |
| T05      | Exchange of Platitudes              |
| T07      | I Blame My Mother                   |
| T08      | Just Wait Until After School        |
| T09      | Kiss My Boo-Boo                     |
| T11      | Perhaps It's a Glandular Problem    |
| T13      | What Are The Civilian Applications? |

Листинг 7.10 комбинирует внутреннее объединение с условиями WHERE, чтобы отобразить список книг, опубликованных в штате Калифорния или вне крупных стран Северной Америки (см. раздел «Фильтрация строк с помощью предложения WHERE» в главе 4). Результат исполнения листинга представлен на рис. 7.10.

П

При использовании синтаксиса WHERE листинг 7.10 будет выглядеть следующим образом:

```
SELECT t.title_id, t.title_name,
 p.state, p.country
FROM titles t, publishers p
WHERE t.pub_id = p.pub_id
 AND (p.state = 'CA'
 OR p.country NOT IN
 ('USA', 'Canada', 'Mexico'))
ORDER BY t.title_id ASC;
```

| state | country |
|-------|---------|
| NULL  | Germany |
| CA    | USA     |
| CA    | USA     |
| CA    | USA     |
| NULL  | Germany |
| CA    | USA     |
| CA    | USA     |
| CA    | USA     |
| NULL  | Germany |

**Рис. 7.10.** Результат выполнения листинга 7.10

Листинг 7.11 комбинирует внутреннее объединение с функцией COUNT() и предложением GROUP BY, чтобы отобразить список книг, написанных всеми авторами (в том числе в соавторстве). За информацией о возможностях и синтаксисе предложения GROUP BY обращайтесь к главе 6. Результат исполнения листинга показан на рис. 7.11. Обратите внимание, что, как и на рис. 7.7, автор A07 (Paddy O’Furniture) не включен в результат, поскольку он не написал ни одной книги, и для него нет записей в таблице соответствий title\_authors. В листинге 7.30 приводится пример того, как отобразить список авторов, которые не написали ни одной книги.

**П** При использовании синтаксиса WHERE листинг 7.11 будет выглядеть следующим образом:

```
SELECT a.au_id,
 COUNT (ta.title_id)
 AS "Num books"
FROM authors a, title_authors ta
WHERE a.au_id = ta.au_id
GROUP BY a.au_id
ORDER BY a.au_id ASC;
```

**Листинг 7.11.** Отобразить список книг, написанных всеми авторами (в том числе в соавторстве). Результат исполнения см. на рис. 7.11

```
SELECT
 a.au_id,
 COUNT(ta.title_id) AS "Num books"
FROM authors a
INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
GROUP BY a.au_id
ORDER BY a.au_id ASC;
```

| au_id | Num books |
|-------|-----------|
| A01   | 3         |
| A02   | 4         |
| A03   | 2         |
| A04   | 4         |
| A05   | 1         |
| A06   | 3         |

**Рис. 7.11.** Результат выполнения листинга 7.11

**Листинг 7.12.** Отобразить авансы, выплаченные за книги-биографии. Результат исполнения листинга см. на рис. 7.12

```
LISTING
SELECT
 t.title_id,
 t.title_name,
 r.advance
FROM royalties r
INNER JOIN titles t
 ON r.title_id = t.title_id
WHERE t.type = 'biography'
 AND r.advance IS NOT NULL
ORDER BY r.advance DESC;
```

Листинг 7.12 использует условия WHERE, чтобы отобразить авансы, выплаченные за книги-биографии. Результат исполнения листинга показан на рис. 7.12.



При использовании синтаксиса WHERE листинг 7.12 будет выглядеть следующим образом:

```
SELECT t.title_id, t.title_name,
 r.advance
FROM royalties r, titles_t
WHERE r.title_id = t.title_id
 AND t.type = 'biography'
 AND r.advance IS NOT NULL
ORDER BY r.advance DESC;
```

| title_id | title_name                 | advance    |
|----------|----------------------------|------------|
| T07      | I Blame My Mother          | 1000000.00 |
| T12      | Spontaneouse, Not Annoying | 50000.00   |
| T06      | How About Never?           | 20000.00   |

**Рис. 7.12.** Результат выполнения листинга 7.12

Листинг 7.13 использует агрегатные функции и предложение GROUP BY, чтобы отобразить количество выплаченных авансов и их общую сумму за каждый тип книг. Результат исполнения листинга представлен на рис. 7.13.

**П** При использовании синтаксиса WHERE листинг 7.13 будет выглядеть так:

```
SELECT t.type,
 COUNT (r.advance)
 AS "COUNT(r.advance)",
 SUM(r.advance)
 AS "SUM(r.advance)"
FROM royalties r, titles_t
WHERE r.title_id = t.title_id
 AND r.advance IS NOT NULL
GROUP BY t.type
ORDER BY t.type DESC;
```

**Листинг 7.13.** Отобразить счет и аванс, выплаченный за каждый тип книг. Результат исполнения листинга см. на рис. 7.13

```
Листинг



SELECT
 t.type,
 COUNT(r.advance)
 AS "COUNT(r.advance)",
 SUM(r.advance)
 AS "SUM(r.advance)"
FROM royalties r
INNER JOIN titles t
 ON r.title_id = t.title_id
WHERE r.advance IS NOT NULL
GROUP BY t.type
ORDER BY t.type ASC;
```

| type       | COUNT(r.advance) | SUM(r.advance) |
|------------|------------------|----------------|
| -----      | -----            | -----          |
| biography  | 3                | 1070000.00     |
| children   | 2                | 0.00           |
| computer   | 1                | 15000.00       |
| history    | 3                | 31000.00       |
| psychology | 3                | 220000.00      |

**Рис. 7.13.** Результат выполнения листинга 7.13



**Листинг 7.14.** Отображение количества выплаченных авансов и их общей суммы за каждый тип книг с разбивкой по издательствам. Результат исполнения листинга см. на рис. 7.14

 Листинг 

```
SELECT
 t.type,
 t.pub_id,
 COUNT(r.advance) AS
"COUNT(r.advance)",
 SUM(r.advance) AS "SUM(r.advance)"
FROM royalties r
INNER JOIN titles t
 ON r.title_id = t.title_id
WHERE r.advance IS NOT NULL
GROUP BY t.type, t.pub_id
ORDER BY t.type ASC, t.pub_id ASC;
```

Листинг 7.14 является аналогом листинга 7.13, только он использует дополнительный столбец, чтобы отобразить счет и аванс, выплаченный за каждый тип книг издательством. Результат исполнения листинга показан на рис. 7.14.



При использовании синтаксиса WHERE листинг 7.14 будет выглядеть так:

```
SELECT t.type, t.pub_id,
 COUNT (r.advance)
 AS "COUNT(r.advance)",
 SUM(r.advance)
 AS "SUM(r.advance)"
FROM royalties r, titles_t
WHERE r.title_id = t.title_id
 AND r.advance IS NOT NULL
GROUP BY t.type, t.pub_id
ORDER BY t.type ASC, t.pub_id ASC;
```

| type       | pub_id | COUNT(r.advance) | SUM(r.advance) |
|------------|--------|------------------|----------------|
| -----      | -----  | -----            | -----          |
| biography  | P01    | 2                | 70000.00       |
| biography  | P03    | 1                | 1000000.00     |
| children   | P04    | 2                | 0.00           |
| computer   | P02    | 1                | 15000.00       |
| history    | P01    | 1                | 10000.00       |
| history    | P03    | 2                | 21000.00       |
| psychology | P04    | 3                | 220000.00      |

**Рис. 7.14.** Результат выполнения листинга 7.14

Листинг 7.15 использует предложение `HAVING`, чтобы отобразить список соавторов для всех книг, у которых они есть. За информацией о синтаксисе предложения `HAVING` обратитесь к разделу «Фильтрация групп с помощью предложения `HAVING`» в главе 6. Результат исполнения листинга показан на рис. 7.15.

**П**

При использовании синтаксиса `WHERE` листинг 7.15 будет выглядеть следующим образом:

```
SELECT ta.title_id,
 COUNT (ta.au_id) AS "Num authors"
FROM authors a, title_authors ta
WHERE a.au_id = ta.au_id
GROUP BY ta.title_id
HAVING COUNT (ta.au_id) > 1
ORDER BY ta.title_id ASC;
```

**Листинг 7.15.** Отобразить список соавторов для всех книг, у которых они есть. Результат исполнения листинга см. на рис. 7.15

| Листинг                                                                                                                                                                                                                   |             |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| <pre>SELECT     ta.title_id,     COUNT(ta.au_id) AS "Num authors" FROM authors a INNER JOIN title_authors ta     ON a.au_id = ta.au_id GROUP BY ta.title_id HAVING COUNT(ta.au_id) &gt; 1 ORDER BY ta.title_id ASC;</pre> |             |
| title_id                                                                                                                                                                                                                  | Num authors |
| -----                                                                                                                                                                                                                     | -----       |
| T04                                                                                                                                                                                                                       | 2           |
| T07                                                                                                                                                                                                                       | 2           |
| T11                                                                                                                                                                                                                       | 3           |

**Рис. 7.15.** Результат выполнения листинга 7.15

**Листинг 7.16.** Отобразить все книги, прибыль от которых (= цена × продажи) превышает сумму аванса, уплаченного автору, не меньше чем в 10 раз. Результат исполнения листинга см. на рис. 7.16

Листинг

```
SELECT
 t.title_id,
 t.title_name,
 r.advance,
 t.price * t.sales AS "Revenue"
FROM titles t
INNER JOIN royalties r
 ON t.price * t.sales > r.advance * 10
 AND t.title_id = r.title_id
ORDER BY t.price * t.sales DESC;
```

Также вы можете объединять таблицы по столбцам, содержащим значения, которые не являются равными. Листинг 7.16 использует оператор «больше чем» (>), чтобы найти все книги, прибыль от которых (= цена × продажи) превышает сумму аванса, выплаченного автору, не меньше чем в 10 раз. Результат исполнения листинга показан на рис. 7.16. Операторы <, <=, > и >= используются при объединении таблиц довольно часто, а оператор неравенства (<>) – редко. Как правило, объединения по неравенству полезны только при использовании с самообъединениями (см. раздел «Создание самообъединения» далее в этой главе).

П

При использовании синтаксиса WHERE листинг 7.16 будет выглядеть следующим образом:

```
SELECT t.title_id, t.title_name,
 r.advance,
 t.price * t.sales AS "Revenue"
FROM titles t, royalties r
WHERE t.price * t.sales >
 r.advance * 10
 AND t.title_id = r.title_id
ORDER BY t.price * t.sales DESC;
```

| title_id | title_name                          | advance    | Revenue     |
|----------|-------------------------------------|------------|-------------|
| T07      | I Blame My Mother                   | 1000000.00 | 35929790.00 |
| T05      | Exchange of Platitudes              | 100000.00  | 1400008.00  |
| T12      | Spontaneouse, But Not Annoing       | 50000.00   | 1299012.99  |
| T03      | Ask Your System Administrator       | 15000.00   | 1025396.65  |
| T13      | What Are The Civilian Applications? | 20000.00   | 313905.33   |
| T06      | How About Never?                    | 20000.00   | 225834.00   |
| T02      | 200 Years of German Humor           | 1000.00    | 190841.70   |
| T09      | Kiss My Boo-Boo                     | .00        | 69750.00    |
| T08      | Just Wait Until After School        | .00        | 40950.00    |

**Рис. 7.16.** Результат выполнения листинга 7.16

Сложные запросы могут быть следствием простых вопросов. В листинге 7.17 нам необходимо объединить три таблицы, чтобы отобразить список авторов и названий книг, которые написали все авторы (в том числе в соавторстве). Результат исполнения листинга показан на рис. 7.17.


**П** При использовании синтаксиса WHERE листинг 7.17 будет выглядеть так:

```
SELECT a.au_fname, a.au_lname,
 t.title_name,
FROM authors a, title_authors ta,
 titles t
WHERE a.au_id = ta.au_id
 AND t.title_id = ta.title_id
ORDER BY a.au_lname ASC,
 a.au_fname ASC,
 t.title_name ASC;
```

**Листинг 7.17.** Отобразить список авторов и названий книг, которые написали все авторы (в том числе в соавторстве). Результат исполнения листинга см. на рис. 7.17

Листинг

```
SELECT
 a.au_fname,
 a.au_lname,
 t.title_name
FROM authors a
INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
INNER JOIN titles t
 ON t.title_id = ta.title_id
ORDER BY a.au_lname ASC, a.au_fname
ASC,
 t.title_name ASC;
```

 Чтобы запустить листинг 7.17 в Microsoft Access, введите:

```
SELECT a.au_fname, a.au_lname,
 t.title_name
FROM titles AS t
INNER JOIN (authors AS a
 INNER JOIN title_authors AS ta
 ON a.au_id = ta.au_id)
 ON t.title_id = ta.title_id
ORDER BY a.au_lname ASC,
 a.au_fname ASC,
 t.title_name ASC;
```

| au_fname  | au_lname  | title_name                          |
|-----------|-----------|-------------------------------------|
| -----     | -----     | -----                               |
| Sarah     | Buchman   | 1977!                               |
| Sarah     | Buchman   | 200 Years of German Humor           |
| Sarah     | Buchman   | What Are The Civilian Applications? |
| Wendy     | Heydemark | How About Never?                    |
| Wendy     | Heydemark | I Blame My Mother                   |
| Wendy     | Heydemark | Not Without My Faberge Egg          |
| Wendy     | Heydemark | Spontaneous, Not Annoying           |
| Hallie    | Hull      | But I Did It Unconsciously          |
| Hallie    | Hull      | Perhaps It's a Glandular Problem    |
| Klee      | Hull      | But I Did It Unconsciously          |
| Klee      | Hull      | Exchange of Platitudes              |
| Klee      | Hull      | I Blame My Mother                   |
| Klee      | Hull      | Perhaps It's a Glandular Problem    |
| Christian | Kells     | Ask Your System Administrator       |
|           | Kellsey   | Just Wait Until After School        |
|           | Kellsey   | Kiss My Boo-Boo                     |
|           | Kellsey   | Perhaps It's a Glandular Problem    |

**Рис. 7.17.** Результат выполнения листинга 7.17

**Листинг 7.18.** Отобразить имена авторов, названия всех книг, а также названия издательств. Результат выполнения листинга см. на рис. 7.18

Листинг

```
SELECT
 a.au_fname,
 a.au_lname,
 t.title_name,
 p.pub_name
FROM authors a
INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
INNER JOIN titles t
 ON t.title_id = ta.title_id
INNER JOIN publishers p
 ON p.pub_id = t.pub_id
ORDER BY a.au_lname ASC, a.au_fname
ASC,
 t.title_name ASC;
```

Листинг 7.18 является усложненным вариантом листинга 7.17. Он использует объединение четырех таблиц, чтобы отобразить названия издательств, а также имена авторов и названия книг. Результат выполнения листинга показан на рис. 7.18.

| au_fname  | au_lname  | title_name                          | pub_name            |
|-----------|-----------|-------------------------------------|---------------------|
| -----     | -----     | -----                               | -----               |
| Sarah     | Buchman   | 1977!                               | Abatis Publishers   |
| Sarah     | Buchman   | 200 Years of German Humor           | Schadenfreude Press |
| Sarah     | Buchman   | What Are The Civilian Applications? | Schadenfreude Press |
| Wendy     | Heydemark | How About Never?                    | Abatis Publishers   |
| Wendy     | Heydemark | I Blame My Mother                   | Schadenfreude Press |
| Wendy     | Heydemark | Not Without My Faberge Egg          | Abatis Publishers   |
| Wendy     | Heydemark | Spontaneous, Not Annoying           | Abatis Publishers   |
| Hallie    | Hull      | But I Did It Unconsciously          | Tenterhooks Press   |
| Hallie    | Hull      | Perhaps It's a Glandular Problem    | Tenterhooks Press   |
| Klee      | Hull      | But I Did It Unconsciously          | Tenterhooks Press   |
| Klee      | Hull      | Exchange of Platitudes              | Tenterhooks Press   |
| Klee      | Hull      | I Blame My Mother                   | Schadenfreude Press |
| Klee      | Hull      | Perhaps It's a Glandular Problem    | Tenterhooks Press   |
| Christian | Kells     | Ask Your System Administrator       | Core Dump Books     |
|           | Kellsey   | Just Wait Until After School        | Tenterhooks Press   |
|           | Kellsey   | Kiss My Boo-Boo                     | Tenterhooks Press   |
|           | Kellsey   | Perhaps It's a Glandular Problem    | Tenterhooks Press   |

**Рис. 7.18.** Результат выполнения листинга 7.18

**П**

При использовании синтаксиса WHERE листинг 7.18 будет выглядеть так:

```
SELECT a.au_fname, a.au_lname,
 t.title_name, p.pub_name
FROM authors a, title_authors ta,
 titles t, publishers p
WHERE a.au_id = ta.au_id
 AND t.title_id = ta.title_id
 AND p.pub_id = t.pub_id
ORDER BY a.au_lname ASC,
 a.au_fname ASC,
 t.title_name ASC;
```



Чтобы запустить листинг 7.18 в Microsoft Access, введите:

```
SELECT a.au_fname, a.au_lname,
 t.title_name, p.pub_name
FROM (publishers AS p
 INNER JOIN titles AS t
 ON p.pub_id = t.pub_id)
INNER JOIN (authors AS a)
 INNER JOIN (authors AS ta
 ON a.au_id = ta.au_id)
 ON t.title_id = ta.title_id
ORDER BY a.au_lname ASC,
 a.au_fname ASC,
 t.title_name ASC;
```

Листинг 7.19 рассчитывает гонорары за все книги. Гонорар за книгу – это прибыль от ее продажи ( $= \text{цена} \times \text{продажи}$ ), умноженная на процент гонорара (процент от прибыли, выплачиваемый автору). В большинстве случаев автор получает аванс как часть гонорара. Чтобы выплатить остаток гонорара автору, издатель вычитает аванс из общей суммы гонорара. Если общая прибыль больше нуля, издатель должен заплатить автору; если прибыль равна нулю или меньше нуля, автор не получает гонорар, так как он не «отработал» даже свой аванс. Результат исполнения листинга показан на рис. 7.19. Общие гонорары помечены как "Total royalties", общие авансы – как "Total advances", а гонорары за вычетом авансов – как "Total due to authors".

Листинг 7.19 рассчитывает общие гонорары за все книги; в других примерах мы покажем, как распределить гонорары по авторам, книгам, издательствам и другим группам.



При использовании синтаксиса WHERE листинг 7.19 будет выглядеть так:

```
SELECT
 SUM (t.sales * t.price *
 → r.royalty_rate)
 AS "Total royalties",
 SUM (r.advance)
 AS "Total advances",
 SUM (t.sales * t.price *
 → r.royalry_rate) - r.advance)
 AS "Total due to authors"
FROM titles t, royalties r
WHERE r.title_id = t.title_id
AND t.sales IS NOT NULL;
```

**Листинг 7.19.** Рассчитать гонорары за все книги. Результат исполнения листинга см. на рис. 7.19

Листинг

```
SELECT
 SUM(t.sales * t.price * r.royalty_rate) AS "Total royalties",
 SUM(r.advance) AS "Total advances",
 SUM((t.sales * t.price * r.royalty_rate) - r.advance) AS "Total due to authors"
FROM titles t
INNER JOIN royalties r
 ON r.title_id = t.title_id
WHERE t.sales IS NOT NULL;
```

| Total royalties | Total advances | Total due to authors |
|-----------------|----------------|----------------------|
| -----           | -----          | -----                |
| 4387219.55      | 1336000.00     | 3051219.55           |

**Рис. 7.19.** Результат выполнения листинга 7.19

В листинге 7.20 используется объединение трех таблиц, чтобы рассчитать гонорар, полученный каждым автором за все книги, которые он написал (в том числе в соавторстве). Так как у книги может быть несколько авторов, при расчете гонорара учитывается процент гонорара за книгу, полученный каждым автором (а также аванс). Процент гонорара за книгу, полученный автором, приводится в таблице `title_authors` в столбце `royalty_share`. Если у книги один автор, процент `royalty_share` равен 1.0 (100%). Если у книги несколько авторов, значение `royalty_share` для каждого автора колеблется между 0 и 1; все значения `royalty_share` для одной книги в сумме должны равняться 1.0 (100%). Результат исполнения листинга показан на рис. 7.20. Сумма всех значений последних трех столбцов результата соответствует общему значению на рис. 7.19.



При использовании синтаксиса `WHERE` листинг 7.20 будет выглядеть следующим образом:

```
SELECT ta.au_id, t.title_id,
 t.pub_id,
 t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share
 AS "Royalty share",
 r.advance * ta.royalty_share
 AS "Advance share",
 (t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance *
 → ta.royalty_share)
 AS "Due to author"
FROM title_authors ta,
 titles t, royalties r
WHERE t.title_id = ta.title_id
 AND r.title_id = t.title_id
 AND t.sales IS NOT NULL
ORDER BY ta.au_id ASC,
 t.title_id ASC;
```

**Листинг 7.20.** Рассчитать гонорар, полученный каждым автором за все книги, которые он написал (в том числе в соавторстве). Результат исполнения листинга см. на рис. 7.20

Листинг

```
SELECT
 ta.au_id,
 t.title_id,
 t.pub_id,
 t.sales * t.price * r.royalty_rate * ta.royalty_share AS "Royalty share",
 r.advance * ta.royalty_share AS "Advance share",
 (t.sales * t.price * r.royalty_rate * ta.royalty_share) -
 → (r.advance * ta.royalty_share) AS "Due to author"
FROM title_authors ta
INNER JOIN titles t
 ON t.title_id = ta.title_id
INNER JOIN royalties r
 ON r.title_id = t.title_id
WHERE t.sales IS NOT NULL
ORDER BY ta.au_id ASC, t.title_id ASC;
```





Чтобы запустить листинг 7.20 в Microsoft Access, введите:

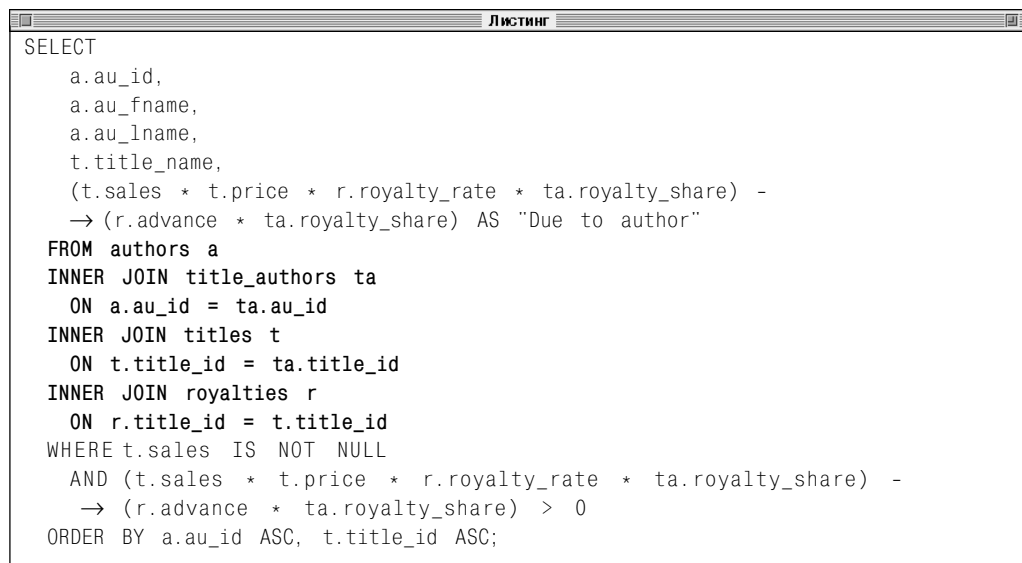
```
SELECT ta.au_id, t.title_id,
 t.pub_id,
 t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share
 AS "Royalty share",
 r.advance * ta.royalty_share
 AS "Advance share",
 (t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance * ta.royalty_share)
 AS "Due to author"
FROM (titles AS t
 INNER JOIN royalties AS r
 ON t.title_id = r.title_id)
 INNER JOIN title_authors AS ta
 ON t.title_id = ta.title_id
WHERE t.sales IS NOT NULL
ORDER BY ta.au_id ASC,
 t.title_id ASC;
```

| au_id | title_id | pub_id | Royalty share | Advance share | Due to author |
|-------|----------|--------|---------------|---------------|---------------|
| A01   | T01      | P01    | 622.32        | 10000.00      | -9377.68      |
| A01   | T02      | P03    | 11450.50      | 1000.00       | 10450.50      |
| A01   | T13      | P03    | 18834.32      | 20000.00      | -1165.68      |
| A02   | T06      | P01    | 18066.72      | 20000.00      | -1933.28      |
| A02   | T07      | P03    | 1976138.45    | 500000.00     | 1476138.45    |
| A02   | T12      | P01    | 116911.17     | 50000.00      | 66911.17      |
| A03   | T04      | P04    | 8106.38       | 12000.00      | -3893.62      |
| A03   | T11      | P04    | 15792.90      | 30000.00      | -14207.10     |
| A04   | T04      | P04    | 5404.26       | 8000.00       | -2595.74      |
| A04   | T05      | P04    | 126000.72     | 100000.00     | 26000.72      |
| A04   | T07      | P03    | 1976138.45    | 500000.00     | 1476138.45    |
| A04   | T11      | P04    | 15792.90      | 30000.00      | -14207.10     |
| A05   | T03      | P02    | 71777.77      | 15000.00      | 56777.77      |
| A06   | T08      | P04    | 1638.00       | .00           | 1638.00       |
| A06   | T09      | P04    | 3487.50       | .00           | 3487.50       |
| A06   | T11      | P04    | 21057.20      | 40000.00      | -18942.80     |

Рис. 7.20. Результат выполнения листинга 7.20

Листинг 7.21 является аналогом листинга 7.20, только он добавляет еще таблицу `authors`, чтобы распечатать имена авторов, а также предложение `WHERE`, чтобы извлечь строки только с теми гонорарами, которые больше нуля. Результат исполнения листинга показан на рис. 7.21.

**Листинг 7.21.** Отобразить только те гонорары за книги, которые больше нуля. Результат исполнения листинга см. на рис. 7.21



```
SELECT
 a.au_id,
 a.au_fname,
 a.au_lname,
 t.title_name,
 (t.sales * t.price * r.royalty_rate * ta.royalty_share) -
 → (r.advance * ta.royalty_share) AS "Due to author"
FROM authors a
INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
INNER JOIN titles t
 ON t.title_id = ta.title_id
INNER JOIN royalties r
 ON r.title_id = t.title_id
WHERE t.sales IS NOT NULL
 AND (t.sales * t.price * r.royalty_rate * ta.royalty_share) -
 → (r.advance * ta.royalty_share) > 0
ORDER BY a.au_id ASC, t.title_id ASC;
```



При использовании синтаксиса WHERE листинг 7.21 будет выглядеть так:

```
SELECT a.au_id, a.au_fname,
 a.au_lname, t.title_name,
 (t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance * ta.royalty_share)
 AS "Due to author"
FROM authors a, title_authors ta,
 titles t, royalties r
WHERE a.au_id = ta.au_id
 AND t.title_id = ta.title_id
 AND r.title_id = t.title_id
 AND t.sales IS NOT NULL
 AND (t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance *
 → ta.royalty_share) > 0
ORDER BY a.au_id ASC,
 t.title_id ASC;
```



Чтобы запустить листинг 7.21 в Microsoft Access, введите:

```
SELECT a.au_id, a.au_fname,
 a.au_lname, t.title_name,
 → (t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance * ta.royalty_share)
 AS "Due to author"
FROM (titles AS t
 INNER JOIN royalties AS r
 ON t.title_id = r.title_id)
INNER JOIN (authors AS a
 INNER JOIN title_authors AS ta
 ON a.au_id = ta.au_id
 ON t.title_id = ta.title_id)
WHERE t.sales IS NOT NULL
 AND (t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance * ta.royalty_share)
 → > 0
ORDER BY a.au_id ASC,
 t.title_id ASC;
```

| au_id | au_fname  | au_lname  | title_name                 | Due to author |
|-------|-----------|-----------|----------------------------|---------------|
| ----- | -----     | -----     | -----                      | -----         |
| A01   | Sarah     | Buchman   | 200YearsofGermanHumor      | 10450.50      |
| A02   | Wendy     | Heydemark | IBlameMyMother             | 1476138.45    |
| A02   | Wendy     | Heydemark | Spontaneous,Notannoying    | 66911.17      |
| A04   | Klee      | Hull      | ExchangeofPlatitudes       | 26000.72      |
| A04   | Klee      | Hull      | IBlameMyMother             | 1476138.45    |
| A05   | Christian | Kells     | AskYourSystemAdministrator | 56777.77      |
| A06   |           | Kellsey   | JustWaitUntilAfterSchool   | 1638.00       |
| A06   |           | Kellsey   | KissMyBoo-Boo              | 3487.50       |

**Рис. 7.21.** Результат выполнения листинга 7.21

Листинг 7.22 использует предложение GROUP BY, чтобы рассчитать все гонорары, выплаченные каждым издателем. Функция COUNT() определяет количество книг, за которое каждое издательство выплачивает гонорары. Обратите внимание, что здесь не нужно вводить процент от гонорара, полученный каждым автором, так как расчеты по каждому автору не производятся. Результат исполнения листинга показан на рис. 7.22. Сумма всех значений последних трех столбцов результата соответствует общему значению на рис. 7.19.

П

При использовании синтаксиса WHERE листинг 7.22 будет выглядеть так:

```
SELECT t.pub_id,
 COUNT(t.sales) AS "Num books",
 SUM(t.sales * t.price *
 → r.royalty_rate)
 AS "Total royalties",
 SUM(r.advance)
 AS "Total advances",
 SUM((t.sales * t.price *
 → r.royalty_rate) -
 → r.advance)
 AS "Total due to authors"
FROM titles t, royalties r
WHERE r.title_id = t.title_id
AND t.sales IS NOT NULL
GROUP BY t.pub_id
ORDER BY t.pub_id ASC;
```

**Листинг 7.22.** Рассчитать гонорары, выплаченные каждым издателем. Результат исполнения листинга см. на рис. 7.22

Листинг

```
SELECT
 t.pub_id,
 COUNT(t.sales) AS "Num books",
 SUM(t.sales * t.price * r.royalty_rate) AS "Total royalties",
 SUM(r.advance) AS "Total advances",
 SUM((t.sales * t.price * r.royalty_rate) - r.advance) AS "Total due to authors"
FROM titles t
INNER JOIN royalties r
 ON r.title_id = t.title_id
WHERE t.sales IS NOT NULL
GROUP BY t.pub_id
ORDER BY t.pub_id ASC;
```

| pub_id | Num books | Total royalties | Total advances | Total due to authors |
|--------|-----------|-----------------|----------------|----------------------|
| P01    | 3         | 135600.21       | 80000.00       | 55600.21             |
| P02    | 1         | 71777.77        | 15000.00       | 56777.77             |
| P03    | 3         | 3982561.72      | 1021000.00     | 2961561.72           |
| P04    | 5         | 197279.85       | 220000         | -22720.15            |

**Рис. 7.22.** Результат выполнения листинга 7.22

Листинг 7.23 аналогичен листингу 7.22, но рассчитывает все гонорары, полученные каждым автором за все книги. Результат исполнения листинга представлен на рис. 7.23. Сумма всех значений последних трех столбцов результата соответствует общему значению на рис. 7.19.

**Листинг 7.23.** Рассчитать гонорары, полученные каждым автором за все книги. Результат исполнения листинга см. на рис. 7.23

Листинг

```
SELECT
 ta.au_id,
 COUNT(sales) AS "Num books",
 SUM(t.sales * t.price * r.royalty_rate * ta.royalty_share) AS "Total royalties
share",
 SUM(r.advance * ta.royalty_share) AS "Total advances share",
 SUM((t.sales * t.price * r.royalty_rate * ta.royalty_share) -
 → (r.advance * ta.royalty_share)) AS "Total due to author"
FROM title_authors ta
INNER JOIN titles t
 ON t.title_id = ta.title_id
INNER JOIN royalties r
 ON r.title_id = t.title_id
WHERE t.sales IS NOT NULL
GROUP BY ta.au_id
ORDER BY ta.au_id ASC;
```

| au_id | Num books | Total royalties share | Total advances share | Total due to authors |
|-------|-----------|-----------------------|----------------------|----------------------|
| A01   | 3         | 30907.14              | 31000.00             | -92.86               |
| A02   | 3         | 2111116.34            | 570000.00            | 1541116.34           |
| A03   | 2         | 23899.28              | 42000.00             | -18100.72            |
| A04   | 4         | 2123336.32            | 638000.00            | 1485336.32           |
| A05   | 1         | 71777.77              | 15000.00             | 56777.77             |
| A06   | 3         | 26182.70              | 40000.00             | -13817.30            |

**Рис. 7.23.** Результат выполнения листинга 7.23

**П**

При использовании синтаксиса WHERE листинг 7.23 будет выглядеть следующим образом:

```
SELECT ta.au_id,
 COUNT(sales) AS "Num books",
 SUM(t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share)
 AS "Total royalties share",
 SUM(r.advance *
 → ta.royalty_share)
 AS "Total advances share",
 SUM((t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance *
 → ta.royalty_share))
 AS "Total due to author"
FROM title_authors ta, titles t,
 royalties r
WHERE t.title_id = ta.title_id
 AND r.title_id = t.title_id
 AND t.sales IS NOT NULL
GROUP BY ta.au_id
ORDER BY ta.au_id ASC;
```



Чтобы запустить листинг 7.23 в Microsoft Access, введите:

```
SELECT ta.au_id,
 COUNT(sales) AS "Num books",
 SUM(t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share)
 AS "Total royalties share",
 SUM(r.advance *
 → ta.royalty_share)
 AS "Total advances share",
 SUM((t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance *
 → ta.royalty_share))
 AS "Total due to author"
FROM (title_authors AS ta
 INNER JOIN titles AS t
 ON t.title_id = ta.title_id)
 INNER JOIN royalties AS r
 ON r.title_id = t.title_id
WHERE t.sales IS NOT NULL
GROUP BY ta.au_id
ORDER BY ta.au_id ASC;
```

Листинг 7.24 использует два сгруппированных столбца, чтобы рассчитать гонорары, которые будут выплачены каждым издательством, расположенным в США, каждому автору за все его книги. Предложение HAVING отфильтровывает строки только с положительными значениями гонораров, а предложение WHERE – только те издательства, которые находятся в США. Результат исполнения листинга продемонстрирован на рис. 7.24.

**Листинг 7.24.** Рассчитать гонорары, которые будут выплачены каждым издательством, расположенным в США, каждому автору за все его книги. Результат исполнения листинга см. на рис. 7.24

Листинг

```
SELECT
 t.pub_id,
 ta.au_id,
 COUNT(*) AS "Num books",
 SUM(t.sales * t.price * r.royalty_rate * ta.royalty_share) AS "Total royalties share",
 SUM(r.advance * ta.royalty_share) AS "Total advances share",
 SUM((t.sales * t.price * r.royalty_rate * ta.royalty_share) -
 → (r.advance * ta.royalty_share)) AS "Total due to author"
FROM title_authors ta
INNER JOIN titles t
 ON t.title_id = ta.title_id
INNER JOIN royalties r
 ON r.title_id = t.title_id
INNER JOIN publishers p
 ON p.pub_id = t.pub_id
WHERE t.sales IS NOT NULL
 AND p.country IN ('USA')
GROUP BY t.pub_id, ta.au_id
HAVING SUM((t.sales * t.price * r.royalty_rate * ta.royalty_share) -
→ (r.advance * ta.royalty_share)) > 0
ORDER BY t.pub_id ASC, ta.au_id ASC;
```

| pub_id | au_id | Num books | Total royalties share | Total advances share | Total due to author |
|--------|-------|-----------|-----------------------|----------------------|---------------------|
| P01    | A02   | 2         | 134977.89             | 70000.00             | 62977.89            |
| P02    | A05   | 1         | 71777.77              | 15000.00             | 56777.77            |
| P04    | A04   | 3         | 147197.87             | 138000.00            | 9197.87             |

**Рис. 7.24.** Результат выполнения листинга 7.24

**П**

При использовании синтаксиса WHERE листинг 7.24 будет выглядеть так:

```
SELECT t.pub_id, ta.au_id,
 COUNT(*) AS "Num books",
 SUM(t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share)
 AS "Total royalties share",
 SUM(r.advance *
 → ta.royalty_share)
 AS "Total advances share",
 SUM((t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance *
 → ta.royalty_share))
 AS "Total due to author"
FROM title_authors ta, titles t,
 → royalties r, publishers p
WHERE t.title_id = ta.title_id
 AND r.title_id = t.title_id
 AND p.pub_id = t.pub_id
 AND t.sales IS NOT NULL
 AND p.country IN ('USA')
GROUP BY t.pub_id, ta.au_id
HAVING SUM((t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance * ta.royalty_share))
 → > 0
ORDER BY t.pub_id ASC,
 ta.au_id ASC;
```



Чтобы запустить листинг 7.24 в Microsoft Access, введите:

```
SELECT t.pub_id,
 ta.au_id,
 COUNT(*) AS "Num books",
 SUM(t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share)
 AS "Total royalties share",
 SUM(r.advance *
 → ta.royalty_share)
 AS "Total advances share",
 SUM((t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance *
 → ta.royalty_share))
 AS "Total due to author"
FROM ((publishers AS p
 INNER JOIN titles AS t
 ON p.pub_id = t.pub_id)
 INNER JOIN royalties AS r
 ON t.title_id =
 r.title_id)
 INNER JOIN title_authors AS ta
 ON t.title_id = ta.title_id
WHERE t.sales IS NOT NULL
 AND p.country IN ('USA')
GROUP BY t.pub_id, ta.au_id
HAVING SUM((t.sales * t.price *
 → r.royalty_rate *
 → ta.royalty_share) -
 → (r.advance * ta.royalty_share))
 → > 0
ORDER BY t.pub_id ASC,
 ta.au_id ASC;
```



## Создание внешних объединений с помощью команды OUTER JOIN

В предыдущем разделе вы узнали, что внутренние объединения считывают строки только в том случае, если хотя бы одна строка в двух таблицах соответствует условиям объединения. Внутреннее объединение удаляет строки, которые не совпадают со строкой в другой таблице. *Внешнее объединение* считывает все строки хотя бы из одной таблицы (если эти строки соответствуют любому условию поиска WHERE или HAVING).

Внешние объединения полезны для ответов на вопросы, в которых есть пропущенные значения: например, авторы, не написавшие ни одной книги, или курсы, на которые не записался ни один студент. Кроме того, внешние объединения помогают создавать отчеты, в которых вы желаете отобразить все строки в одной таблице, а также совпадающие строки из другой таблицы. Например, чтобы отобразить всех авторов книг, копий которых было продано больше, чем указано, либо чтобы отобразить заказы на все товары и включить те товары, которые никто не заказал.

В отличие от других объединений, порядок, в котором вы задаете таблицы для внешнего объединения, имеет значение. Поэтому рабочие таблицы внешнего объединения называются *левой таблицей* и *правой таблицей*. Внешние объединения бывают трех видов.

*Левое внешнее объединение* (LEFT OUTER JOIN). Результат исполнения левого внешнего объединения включает все строки из левой

таблицы, заданные в пункте LEFT OUTER JOIN, а не только строки, которые совпадают для связанных столбцов. Если для строки в левой таблице нет соответствия в правой таблице, то в результате строка будет содержать NULL для всех столбцов в списке SELECT, которые были считаны из правой таблицы.

*Правое внешнее объединение* (RIGHT OUTER JOIN) является прямой противоположностью левому. Считываются все строки из правой таблицы. Если для строки из правой таблицы нет соответствия в правой таблице, то для левой таблицы считываются NULL.

*Полное внешнее объединение* (FULL OUTER JOIN) является комбинацией левого и правого внешних объединений. Считывает все строки как из левой, так и из правой таблиц. Если для строки нет соответствия в другой таблице, столбцы в списке SELECT другой таблицы будут содержать NULL. Если в таблицах есть соответствие, то строка будет содержать значения из двух таблиц.

Если левая таблица задается в левом внешнем объединении, считываются все строки из нее, если в правом внешнем объединении задается правая таблица, соответственно, считываются все строки из нее. Полное внешнее объединение считывает все строки из двух таблиц. В любом случае строки, для которых нет соответствий, заполняются значениями null. Таким образом, вы не сможете отличить значения null, которые изначально были в таблице, от тех, что добавлены при исполнении объединения. Помните, что условие NULL = NULL является неизвестным и не имеет совпадений (см. раздел «Значение null» в главе 3).

## Создание левого внешнего объединения

Введите:

```
SELECT columns
FROM left_table
LEFT [OUTER] JOIN right_table
ON join_conditions
```

Здесь *columns* – это несколько разделенных запятыми выражений или названий столбцов из *left\_table* или *right\_table*. *left\_table* и *right\_table* – это названия связанных таблиц. Если в таблицах есть столбцы с одинаковыми названиями, обозначьте эти столбцы с помощью названий таблиц.

*join\_conditions* задает одно или несколько условий объединения, которые должны рассчитываться для каждой пары связанных строк. Условие объединения имеет следующую форму:

```
[left_table.] column op
→ [right_table.] column
```

*op* обычно принимает значение `=`, но может быть и любым оператором сравнения: `=`, `<>`, `<`, `<=`, `>` или `>=` (см. табл. 4.2). Вы можете комбинировать условия с помощью `AND` или `OR` (см. раздел «Комбинирование условий с помощью операторов `AND`, `OR` и `NOT`» в главе 4). Ключевое слово `OUTER` является опциональным.

## Создание правого внешнего объединения

Введите:

```
SELECT columns
FROM left_table
RIGHT [OUTER] JOIN right_table
ON join_conditions
```

*columns*, *left\_table*, *right\_table* и *join\_conditions* имеют такое же значение, как в подразделе «Создание левого внешнего объединения». Ключевое слово `OUTER` является опциональным.

## Создание полного внешнего объединения

Введите:

```
SELECT columns
FROM left_table
FULL [OUTER] JOIN right_table
ON join_conditions
```

*columns*, *left\_table*, *right\_table* и *join\_conditions* имеют такое же значение, как в подразделе «Создание левого внешнего объединения». Ключевое слово `OUTER` является опциональным.



При работе с внешними объединениями вместо предложения `WHERE` следует использовать команду `JOIN`, поскольку она более точная. `SQL` не имеет заданного стандарта предложения `WHERE` для функционирования с внешними объединениями, поэтому работа с командой различается в разных СУБД. Система управления базами данных также может запрещать использование внешних объединений, построенных с помощью предложения `WHERE`, которые не имеют соответствующих внешних объединений `JOIN`. За дополнительной информацией обращайтесь к примечанию **DBMS** далее в этом разделе.



Будьте внимательны при изменении порядка ввода таблиц для внешнего объединения. В отличие от других объединений, внешние объединения не являются ассоциативными. Это значит, что результат запроса, использующего внешнее объединение, зависит от порядка, в котором таблицы были сгруппированы и связаны. Два последующих внутренних объединения, включающих три таблицы, эквивалентны (за исключением порядка столбцов в результате):

```
SELECT * FROM table1
INNER JOIN table2
INNER JOIN table3
```

**Таблица 7.3.** Результат выполнения команды`t1 UNION JOIN t2`

|                      |                      |
|----------------------|----------------------|
| Все строки <i>t1</i> | NULL                 |
| NULL                 | Все строки <i>t2</i> |

и

```
SELECT * FROM table2
INNER JOIN table3
INNER JOIN table1
```

Но два следующих внешних объединения, включающих три таблицы, выдают разные результаты:

```
SELECT * FROM table1
LEFT OUTER JOIN table2
LEFT OUTER JOIN table3
```

и

```
SELECT * FROM table2
LEFT OUTER JOIN table3
LEFT OUTER JOIN table1
```



В стандарте SQL предусмотрено *объединение union*, которое не сопоставляет строки в двух таблицах, а генерирует полное внешнее объединение и удаляет совпадающие строки. Каждая строка в таком объединении дополняет столбцы одной связанной таблицы значениями null для столбцов в другой таблице. Результат выполнения команды `t1 UNION JOIN t2` показан в табл. 7.3.

Мы не описываем команду `UNION JOIN` в специальном разделе, поскольку ее практическое применение ограничено и многие СУБД ее не поддерживают. Вы можете создать аналог объединения *union* с помощью полного внешнего объединения:

```
t1 UNION JOIN t2
```

которое является эквивалентом

```
t1 FULL OUTER JOIN t2 ON 1 = 2
```

*t1* и *t2* — это таблицы, а `1 = 2` — условие, которое всегда ложно. Обратите внимание, что `UNION JOIN` отличается от `UNION`, которое является командой слияния, а не объединением (см. раздел «Комбинирование строк с помощью оператора `UNION`» далее в этой главе).



Чтобы создать внешние объединения, Microsoft SQL Server использует в предложении `WHERE` оператор внешнего объединения `*`. Чтобы создать левое или правое внешнее объединение, добавьте оператор `*` слева или справа от оператора сравнения. При работе с внешними объединениями предложение `WHERE` менее точно, чем `OUTER JOIN`, поэтому его использование может привести к возникновению некорректных запросов. Следующая версия SQL Server может не поддерживать операторы `*=` и `=*`, поэтому мы приведем всего несколько примеров.

Oracle 8i и более ранних версий не поддерживает предложение `JOIN`; вместо него вы должны использовать `WHERE`. Oracle 9i и более поздних версий поддерживает предложение `JOIN`. Для создания внешних объединений Oracle использует оператор внешнего объединения `(+)` в предложении `WHERE`. Добавьте оператор `(+)` после таблицы, которую вам нужно расширить (заполнить нулями). См. примеры в этом разделе.

**Листинг 7.25.** Показать список городов, в которых живут авторы и расположены издательства. Результат исполнения листинга см. на рис. 7.25

```
SELECT a.au_fname, a.au_lname, a.city
FROM authors a;

SELECT p.pub_name, p.city
FROM publishers p;
```

| au_fname  | au_lname    | city          |
|-----------|-------------|---------------|
| Sarah     | Buchman     | Bronx         |
| Wendy     | Heydemark   | Boulder       |
| Hallie    | Hull        | San Francisco |
| Klee      | Hull        | San Francisco |
| Christian | Kells       | New York      |
|           | Kellsey     | Palo Alto     |
| Paddy     | O'Furniture | Sarasota      |

| pub_name            | city          |
|---------------------|---------------|
| Abatis Publishers   | New York      |
| Core Dump Books     | San Francisco |
| Schadenfreude Press | Hamburg       |
| Tenterhooks Press   | Berkley       |

**Рис. 7.25.** Результат выполнения листинга 7.25

Листинг 7.25 и рис. 7.25 будут использоваться в четырех последующих примерах. Они отражают список городов, в которых живут авторы и расположены издательства.

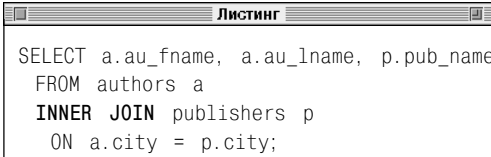
Листинг 7.26 создает внутреннее объединение для столбцов city в таблицах authors и publishers. Результат исполнения показан на рис. 7.26. Здесь отображен список только тех авторов, которые живут в городах, где расположены издательства. Вы можете сравнить результат этого внутреннего объединения с результатами внутренних объединений в трех последующих примерах.

**П**

При использовании синтаксиса WHERE листинг 7.26 будет выглядеть так:

```
SELECT a.au_fname, a.au_lname,
 p.pub_name
FROM authors a, publishers p
WHERE a.city = p.city;
```

**Листинг 7.26.** Отобразить список авторов, которые живут в городах, где расположены издательства. Результат показан на рис. 7.26



```
SELECT a.au_fname, a.au_lname, p.pub_name
FROM authors a
INNER JOIN publishers p
ON a.city = p.city;
```

| au_fname  | au_lname | pub_name          |
|-----------|----------|-------------------|
| -----     | -----    | -----             |
| Hallie    | Hull     | Core Dump Books   |
| Klee      | Hull     | Core Dump Books   |
| Christian | Kells    | Abatis Publishers |

**Рис. 7.26.** Результат выполнения листинга 7.26

**Листинг 7.27.** Это левое внешнее объединение включает в результат все строки, независимо от того, есть ли соответствие в столбце `city` таблицы `publishers`. Результат исполнения см. на рис. 7.27

```

SELECT a.au_fname, a.au_lname, p.pub_name
FROM authors a
LEFT OUTER JOIN publishers p
ON a.city = p.city
ORDER BY p.pub_name ASC,
a.au_lname ASC, a.au_fname ASC;
```

| au_fname  | au_lname    | pub_name          |
|-----------|-------------|-------------------|
| Sarah     | Buchman     | NULL              |
| Wendy     | Heydemark   | NULL              |
|           | Kellsey     | NULL              |
| Paddy     | O'Furniture | NULL              |
| Christian | Kells       | Abatis Publishers |
| Hollie    | Hull        | Core Dump Books   |
| Klee      | Hull        | Core Dump Books   |

**Рис. 7.27.** Результат выполнения листинга 7.27. Обратите внимание, что для четырех авторов в списке нет соответствий, поэтому строки с этими авторами в столбце `pub_name` содержат значение `null`

Листинг 7.27 использует левое внешнее объединение, чтобы включить в результат всех авторов, независимо от того, находятся ли издательства в городах их проживания. Результат исполнения см. на рис. 7.27.



Чтобы запустить листинг 7.27 в Microsoft SQL Server с использованием предложения `WHERE`, введите:

```

SELECT a.au_fname, a.au_lname,
p.pub_name
FROM authors a, publishers p
WHERE a.city *= p.city
ORDER BY p.pub_name ASC,
a.au_lname ASC, a.au_fname ASC;
```

Чтобы запустить листинг 7.27 в Oracle 8i, введите:

```

SELECT a.au_fname, a.au_lname,
p.pub_name
FROM authors a, publishers p
WHERE a.city = p.city (+)
ORDER BY p.pub_name ASC,
a.au_lname ASC, a.au_fname ASC;
```

Листинг 7.28 использует правое внешнее объединение, чтобы включить в результат все издательства, независимо от того, проживают ли авторы в городах, где находятся издательства. Результат исполнения представлен на рис. 7.28.



Чтобы запустить листинг 7.28 в Microsoft SQL Server с использованием синтаксиса WHERE, введите:

```
SELECT a.au_fname, a.au_lname,
 p.pub_name
FROM authors a, publishers p
WHERE a.city =* p.city
ORDER BY p.pub_name ASC,
 a.au_lname ASC, a.au_fname ASC;
```

Чтобы запустить листинг 7.28 в Oracle 8i, введите:

```
SELECT a.au_fname, a.au_lname,
 p.pub_name
FROM authors a, publishers p
WHERE a.city (+) = p.city
ORDER BY p.pub_name ASC,
 a.au_lname ASC, a.au_fname ASC;
```

**Листинг 7.28.** Это правое внешнее объединение включает в результат все строки таблицы publishers, независимо от того, есть ли соответствие в столбце city таблицы authors. Результат исполнения см. на рис. 7.28

**Листинг**

```
SELECT a.au_fname, a.au_lname, p.pub_name
FROM authors a
RIGHT OUTER JOIN publishers p
ON a.city = p.city
ORDER BY p.pub_name ASC,
 a.au_lname ASC, a.au_fname ASC;
```

| au_fname  | au_lname | pub_name            |
|-----------|----------|---------------------|
| -----     | -----    | -----               |
| Christian | Kells    | Abatis Publishers   |
| Hollie    | Hull     | Core Dump Books     |
| Klee      | Hull     | Core Dump Books     |
| NULL      | NULL     | Schadenfreude Press |
| NULL      | NULL     | Tenterhooks Press   |

**Рис. 7.28.** Результат выполнения листинга 7.28. Обратите внимание, что для двух издательств в списке нет соответствий, поэтому в столбцах au\_fname и au\_lname напротив них представлены нули



**Листинг 7.29.** Это полное внешнее объединение включает в результат все строки таблиц `publishers` и `authors`, независимо от того, есть ли соответствия в столбцах `city`. Результат исполнения листинга см. на рис. 7.29

```

-- Листинг
SELECT a.au_fname, a.au_lname, p.pub_name
FROM authors a
FULL OUTER JOIN publishers p
ON a.city = p.city
ORDER BY p.pub_name ASC,
a.au_lname ASC, a.au_fname ASC;
```

Листинг 7.29 использует полное внешнее объединение, чтобы включить в результат все издательства и всех авторов, независимо от того, совпадает ли город проживания авторов с городом, где находится издательство. Результат исполнения листинга показан на рис. 7.29.

| au_fname  | au_lname    | pub_name            |
|-----------|-------------|---------------------|
| Sarah     | Buchman     | NULL                |
| Wendy     | Heydemark   | NULL                |
|           | Kellsey     | NULL                |
| Paddy     | O'Furniture | NULL                |
| Christian | Kells       | Abatis Publishers   |
| Hallie    | Hull        | Core Dump Books     |
| Klee      | Hull        | Core Dump Books     |
| NULL      | NULL        | Schadenfreude Press |
| NULL      | NULL        | Tenterhooks Press   |

**Рис. 7.29.** Результат выполнения листинга 7.29 содержит девять строк: четыре строки с авторами, у которых нет соответствий в строках таблицы `publishers`, три строки, в которых автор и издательство находятся в одном городе, и две строки для издательств, у которых нет соответствий в столбце `city` таблицы `authors`



В Microsoft SQL Server для создания полного внешнего объединения вы не можете поместить оператор \* с двух сторон от оператора сравнения. Вместо этого вы можете совместить левое и правое внешние объединения (см. раздел «Комбинирование строк с помощью оператора UNION» далее в этой главе). Чтобы запустить листинг 7.29 в SQL Server с использованием синтаксиса WHERE, введите:

```
SELECT a.au_fname, a.au_lname,
 p.pub_name
FROM authors a, publishers p
WHERE a.city *= p.city
UNION ALL
SELECT a.au_fname, a.au_lname,
 p.pub_name
FROM authors a, publishers p
WHERE a.city =* p.city
AND a.city IS NULL;
```

В Oracle для создания полного внешнего объединения вы не можете поместить оператор (+) с двух сторон от оператора сравнения. Вместо этого вы можете совместить левое и правое внешние объединения (см. раздел «Комбинирование строк с помощью оператора UNION» далее в этой главе). Чтобы запустить листинг 7.29 в Oracle 8i, введите:

```
SELECT a.au_fname, a.au_lname,
 p.pub_name
FROM authors a, publishers p
WHERE a.city = p.city (+)
UNION ALL
SELECT a.au_fname, a.au_lname,
 p.pub_name
FROM authors a, publishers p
WHERE a.city (+) = p.city
AND a.city IS NULL;
```

Microsoft Access и MySQL не поддерживают полные внешние объединения, но вы можете совмещать левое и правое внешние объединения (см. раздел «Комбинирование строк с помощью оператора UNION» далее в этой главе). В следующем примере первая таблица команды UNION представляет собой левое внешнее объединение, которое считывает все строки таблицы authors и соответствующие строки столбца city таблицы publishers. Вторая таблица команды UNION представляет собой правое внешнее объединение, которое считывает только те строки таблицы publishers, для которых нет соответствий. Чтобы запустить листинг 7.29 в Access или MySQL, введите:

```
SELECT a.au_fname, a.au_lname,
 p.pub_name
FROM authors a,
LEFT OUTER JOIN publishers p
ON a.city = p.city
UNION ALL
SELECT a.au_fname, a.au_lname,
 p.pub_name
FROM authors a,
RIGHT OUTER JOIN publishers p
ON a.city = p.city
WHERE a.city IS NULL;
```

**Листинг 7.30.** Показать количество книг, написанных авторами (в том числе в соавторстве), включая тех, кто не написал ни одной книги. Результат исполнения см. на рис. 7.30

```

Листинг
SELECT
 a.au_id,
 COUNT(ta.title_id) AS "Num books"
FROM authors a
LEFT OUTER JOIN title_authors ta
 ON a.au_id = ta.au_id
GROUP BY a.au_id
ORDER BY a.au_id ASC;
```

| au_id | Num books |
|-------|-----------|
| A01   | 3         |
| A02   | 4         |
| A03   | 2         |
| A04   | 4         |
| A05   | 1         |
| A06   | 3         |
| A07   | 0         |

**Рис. 7.30.** Результат выполнения листинга 7.30

Листинг 7.30 использует левое внешнее объединение, чтобы отобразить количество книг, написанных авторами (в том числе в соавторстве). Результат исполнения представлен на рис. 7.30. Обратите внимание, что, в отличие от листинга 7.11, автор A07 (Paddy O’Furniture) появится в результате, хотя он не написал ни одной книги.



Чтобы запустить листинг 7.30 в Oracle 8i, введите:

```

SELECT a.au_id,
 COUNT(ta.title_id)
 AS "Num books"
FROM authors a, title_authors ta
WHERE a.au_id = ta.au_id (+)
GROUP BY a.au_id
ORDER BY a.au_id ASC;
```

Листинг 7.31 использует условие `WHERE`, чтобы выполнить тестирование на наличие нулей и отобразить только тех авторов, которые не написали ни одной книги. Результат исполнения показан на рис. 7.31.



Чтобы запустить листинг 7.31 в Oracle 8i, введите:

```
SELECT a.au_id, a.au_fname,
 a.au_lname
FROM authors a, title_authors ta
WHERE a.au_id = ta.au_id (+)
 AND ta.au_id IS NULL;
```

**Листинг 7.31.** Отобразить только тех авторов, которые не написали ни одной книги (в том числе в соавторстве). Результат исполнения см. на рис. 7.31

```
Листинг
SELECT a.au_id, a.au_fname, a.au_lname
FROM authors a
LEFT OUTER JOIN title_authors ta
 ON a.au_id = ta.au_id
WHERE ta.au_id IS NULL;
```

| au_id | au_fname | au_lname    |
|-------|----------|-------------|
| A07   | Paddy    | O'Furniture |

**Рис. 7.31.** Результат выполнения листинга 7.31

**Листинг 7.32.** Отобразить авторов и их книги, тиражи которых превысили 100 000 экземпляров. Результат исполнения см. на рис. 7.32

Листинг

```
SELECT a.au_id, a.au_fname, a.au_lname,
 tta.title_id, tta.title_name, tta.sales
FROM authors a
LEFT OUTER JOIN
(SELECT ta.au_id, t.title_id,
 t.title_name, t.sales
FROM title_authors ta
INNER JOIN titles t
ON t.title_id = ta.title_id
WHERE sales > 100000) tta
ON a.au_id = tta.au_id
ORDER BY a.au_id ASC, tta.title_id ASC;
```

Листинг 7.32 комбинирует внутреннее объединение и левое внешнее объединение, чтобы отобразить авторов и их книги, тиражи которых превысили 100 000 экземпляров. В этом примере мы сначала отфильтровали результат команды INNER JOIN, а затем с помощью команды OUTER JOIN объединили его с таблицей authors, чтобы отобразить все ее строки. Результат исполнения показан на рис. 7.32.

| au_id | au_fname  | au_lname    | title_id | title_name                | sales   |
|-------|-----------|-------------|----------|---------------------------|---------|
| A01   | Sarah     | Buchman     | NULL     | NULL                      | NULL    |
| A02   | Wendy     | Heydemark   | T07      | I Blame My Mother         | 1500200 |
| A02   | Wendy     | Heydemark   | T12      | Spontaneous, Not Annoying | 100001  |
| A03   | Hallie    | Hull        | NULL     | NULL                      | NULL    |
| A04   | Klee      | Hull        | T05      | Exchange of Platitudes    | 201440  |
| A04   | Klee      | Hull        | T07      | I Blame My Mother         | 1500200 |
| A05   | Christian | Kells       | NULL     | NULL                      | NULL    |
| A06   |           | Kellsey     | NULL     | NULL                      | NULL    |
| A07   | Paddy     | O'Furniture | NULL     | NULL                      | NULL    |

**Рис. 7.32.** Результат выполнения листинга 7.32



Чтобы запустить листинг 7.32 в Oracle 8i, введите:

```
SELECT a.au_id, a.au_fname,
 a.au_lname,
 tta.title_id, tta.title_name,
 tta.sales
FROM authors a,
 (SELECT ta.au_id, t.title_id,
 t.title_name, t.sales
 FROM title_authors ta,
 titles t
 WHERE t.title_id =
 ta.title_id
 AND sales > 100000) tta
WHERE a.au_id = tta.au_id (+)
ORDER BY a.au_id ASC,
 tta.title_id ASC;
```

MySQL 4.0 и более ранних версий не поддерживает подзапросы (см. примечание **DBMS** в разделе «Принципы работы с подзапросами» в главе 8). Работая со сложными запросами, вы можете создать временную таблицу и поместить в нее результат выполнения подзапроса (см. раздел «Создание временной таблицы с помощью команды CREATE TEMPORARY TABLE» в главе 10). Чтобы запустить листинг 7.32 в MySQL, введите:

```
CREATE TEMPORARY TABLE tta
 SELECT ta.au_id, t.title_id,
 t.title_name, t.sales
 FROM title_authors ta
 INNER JOIN titles t
 ON t.title_id = ta.title_id
 WHERE sales > 100000;

SELECT a.au_id, a.au_fname,
 a.au_lname, tta.title_id,
 tta.title_name, tta.sales
FROM authors a
LEFT OUTER JOIN tta
 ON a.au_id = tta.au_id
ORDER BY a.au_id ASC,
 tta.title_id ASC;

DROP TABLE tta;
```

Таблица 7.4. Таблица employees

| emp_id | emp_name          | boss_id |
|--------|-------------------|---------|
| E01    | Lord Copper       | NULL    |
| E02    | Jocelyn Hitchcock | E01     |
| E03    | Mr. Salter        | E01     |
| E04    | William Boot      | E03     |
| E05    | Mr. Orker         | E03     |

Листинг 7.33. Отобразить имена начальников для всех сотрудников. Результат исполнения см. на рис. 7.33

Листинг

```
SELECT
 e1.emp_name AS "Employee name",
 e2.emp_name AS "Boss name"
FROM employees e1
INNER JOIN employees e2
 ON e1.boss_id = e2.emp_id;
```

|                   |             |
|-------------------|-------------|
| Employee name     | Boss name   |
| -----             | -----       |
| Jocelyn Hitchcock | Lord Copper |
| Mr. Salter        | Lord Copper |
| William Boot      | Mr. Salter  |
| Mr. Orker         | Mr. Salter  |

Рис. 7.33. Результат выполнения листинга 7.33

# Создание самообъединения

Самообъединение – это обычное объединение SQL, которое объединяет таблицу с самой собой и считывает строки из нее путем сравнения значений в нескольких столбцах одной таблицы. Самообъединения часто используются в таблицах с рефлексивными связями (связи между первичными и вторичными ключами в столбце или в нескольких столбцах таблицы и другими столбцами той же таблицы). За информацией о ключах обращайтесь к разделам «Первичные ключи» и «Внешние ключи» в главе 2.

Таблицы в нашей базе данных не имеют рефлексивных связей, поэтому мы приведем пример, который используется во многих книгах и руководствах по SQL. Предположим, что у вас есть таблица под названием employees (см. табл. 7.4).

Поле emp\_id – это первичный ключ, который идентифицирует каждого работника, а поле boss\_id – начальника этого работника. Каждый из начальников также является работником, поэтому следите за тем, чтобы данные каждого начальника, которые вы добавляете в таблицу, совпадали с данными работника. Поле boss\_id служит внешним ключом для поля emp\_id. Листинг 7.33 использует эту рефлексивную связь, чтобы сравнить строки в таблице и отобразить имена начальников для всех сотрудников (вам не потребуется использовать объединение, если нужно извлечь только идентификаторы начальников). Результат исполнения листинга представлен на рис. 7.33.

Эта же таблица (*employees*) появляется дважды в листинге 7.33 с разными названиями (*e1* и *e2*), которые используются, чтобы обозначить столбцы в условии объединения:

```
e1. boss_id = e2.emp_id
```

Как для любого другого объединения, для самообъединения необходимы две таблицы, но вместо того чтобы добавлять вторую таблицу, вы добавляете копию той же самой таблицы. Таким образом, вы можете сравнить столбец в первой копии таблицы и столбец во второй копии. Как и для всех таблиц, ваша СУБД комбинирует и считывает те строки таблицы, которые соответствуют условию объединения. Вы не создаете копию таблицы, а объединяете ее с самой собой. Но вам будет легче понять результат, если вы представите, что имеете дело с двумя копиями одной таблицы.

## Порядок создания самообъединения

Введите:

```
SELECT columns
FROM table [AS] alias1
INNER JOIN table [AS] alias2
ON join_conditions
```

*columns* — это список разделенных запятыми выражений или названий столбцов из *table*; *alias1* и *alias2* — различные названия,

которые будут использоваться для *table* в *join\_conditions* (см. раздел «Создание псевдонимов таблиц с помощью предложения AS» ранее в этой главе).

*join\_conditions* задает одно или несколько условий объединения, которые должны рассчитываться для каждой пары связанных строк. Условие объединения имеет следующий вид:

```
alias1.column op alias2.column
```

*op* может быть любым оператором сравнения: *=*, *<>*, *<*, *<=*, *>* или *>=* (см. табл. 4.2 в главе 4). Вы можете комбинировать несколько условий объединения с помощью AND или OR (см. раздел «Комбинирование условий с помощью операторов AND, OR и NOT» в главе 4).



Можно объединить таблицу с самой собой, даже если в ней нет рефлексивных связей. Обычное самообъединение сравнивает столбец в первой копии таблицы с тем же столбцом во второй ее копии. Такое условие позволяет сравнивать значения в одном столбце со значениями в другом. Мы покажем это на примерах в данном разделе.



Oracle 8i и более ранних версий не поддерживает синтаксис JOIN; советуем использовать синтаксис WHERE. Oracle 9i и более поздних версий поддерживает синтаксис JOIN.



**Листинг 7.34.** Отобразить всех авторов, которые живут в одном штате с автором A04 (Klee Hull). Результат исполнения см. на рис. 7.34

```

SELECT a1.au_id, a1.au_fname,
 a1.au_lname, a1.state
FROM authors a1
INNER JOIN authors a2
 ON a1.state = a2.state
 WHERE a2.au_id = 'A04';

```

| au_id | au_fname | au_lname | state |
|-------|----------|----------|-------|
| A03   | Hallie   | Hull     | CA    |
| A04   | Klee     | Hull     | CA    |
| A06   |          | Kellsey  | CA    |

**Рис. 7.34.** Результат выполнения листинга 7.34

Листинг 7.34 использует условие поиска WHERE и самообъединение столбца state, чтобы найти всех авторов, которые живут в одном штате с автором A04 (Klee Hull). Результат исполнения показан на рис. 7.34.



При использовании синтаксиса WHERE листинг 7.34 будет эквивалентен следующему листингу:

```

SELECT a1.au_id, a1.au_fname,
 a1.au_lname, a1.state
FROM authors a1, authors a2
WHERE a1.state = a2.state
 AND a2.au_id = 'A04';

```



Самообъединения можно преобразовать в подзапросы (см. главу 8). При использовании подзапроса листинг 7.34 будет эквивалентен следующему листингу:

```

SELECT au_id, au_fname,
 au_lname, state
FROM authors
WHERE state IN
 (SELECT state
 FROM authors
 WHERE au_id = 'A04');

```

Листинг 7.35 отображает список биографий, которые продаются лучше. Обратите внимание, что условие поиска WHERE должно быть `type = 'biography'` как для таблицы `t1`, так и для таблицы `t2`, поскольку условие объединения рассматривает столбец `type` как два разных столбца. Результат исполнения показан на рис. 7.35.

П

При использовании синтаксиса WHERE листинг 7.35 будет эквивалентен следующему листингу:

```
SELECT t1.title_id, t1.sales,
 t2.title_id AS "Better seller",
 t2.sales AS "Higher sales"
FROM titles t1, titles t2
WHERE t1.sales < t2.sales
 AND t1.type = 'biography'
 AND t2.type = 'biography'
ORDER BY t1.title_id ASC,
 t2.sales ASC;
```

**Листинг 7.35.** Для каждой биографии отобразить заголовки и данные о продажах других биографий, которые пользуются большим спросом. Результат исполнения см. на рис. 7.35

Листинг

```
SELECT t1.title_id, t1.sales,
 t2.title_id AS "Better seller",
 t2.sales AS "Higher sales"
FROM titles t1
INNER JOIN titles t2
 ON t1.sales < t2.sales
WHERE t1.type = 'biography'
 AND t2.type = 'biography'
ORDER BY t1.title_id ASC, t2.sales ASC;
```

| title_id | sales  | Better seller | Higher sales |
|----------|--------|---------------|--------------|
| -----    | -----  | -----         | -----        |
| T06      | 11320  | T12           | 100001       |
| T06      | 11320  | T07           | 1500200      |
| T12      | 100001 | T07           | 1500200      |

**Рис. 7.35.** Результат выполнения листинга 7.35

**Листинг 7.36.** Отобразить все пары авторов из штата Нью-Йорк. Результат исполнения см. на рис. 7.36

Листинг

```
SELECT
 a1.au_fname, a1.au_lname,
 a2.au_fname, a2.au_lname
FROM authors a1
INNER JOIN authors a2
 ON a1.state = a2.state
WHERE a1.state = 'NY'
ORDER BY a1.au_id ASC, a2.au_id ASC;
```

| au_fname  | au_lname | au_fname  | au_lname |
|-----------|----------|-----------|----------|
| -----     | -----    | -----     | -----    |
| Sarah     | Buchman  | Sarah     | Buchman  |
| Sarah     | Buchman  | Christian | Kells    |
| Christian | Kells    | Sarah     | Buchman  |
| Christian | Kells    | Christian | Kells    |

**Рис. 7.36.** Результат выполнения листинга 7.36

Листинг 7.36 представляет собой самообъединение, которое находит все пары авторов из штата Нью-Йорк. Результат исполнения представлен на рис. 7.36.



При использовании синтаксиса WHERE листинг 7.36 будет эквивалентен следующему листингу:

```
SELECT
 a1.au_fname, a1.au_lname,
 a2.au_fname, a2.au_lname
FROM authors a1, authors a2
WHERE a1.state = a2.state
 AND a1.state = 'NY'
ORDER BY a1.au_id ASC,
 a2.au_id ASC;
```

Первая и четвертая строки на рис. 7.36 – это повторы, так как они показывают, что Сара Бухман (Sarah Buchman) живет в том же штате, что и Сара Бухман, и аналогичную информацию о Кристиане Келлсе (Christian Kells). Чтобы удалить эти строки, мы добавили еще одно условие объединения, считывающее только те строки, в которых два автора различаются (см. листинг 7.37 и рис. 7.37).

**П** При использовании синтаксиса WHERE листинг 7.37 будет эквивалентен следующему листингу:

```
SELECT
 a1.au_fname, a1.au_lname,
 a2.au_fname, a2.au_lname
FROM authors a1, authors a2
WHERE a1.state = a2.state
 AND a1.au_id <> a2.au_id
 AND a1.state = 'NY'
ORDER BY a1.au_id ASC,
 a2.au_id ASC;
```

**Листинг 7.37.** Отобразить пары разных авторов из штата Нью-Йорк. Результат исполнения см. на рис. 7.37

```
Листинг
SELECT
 a1.au_fname, a1.au_lname,
 a2.au_fname, a2.au_lname
FROM authors a1
INNER JOIN authors a2
 ON a1.state = a2.state
 AND a1.au_id <> a2.au_id
WHERE a1.state = 'NY'
ORDER BY a1.au_id ASC, a2.au_id ASC;
```

| au_fname  | au_lname | au_fname  | au_lname |
|-----------|----------|-----------|----------|
| -----     | -----    | -----     | -----    |
| Sarah     | Buchman  | Christian | Kells    |
| Christian | Kells    | Sarah     | Buchman  |

**Рис. 7.37.** Результат выполнения листинга 7.37

**Листинг 7.38.** Отобразить пары разных авторов из штата Нью-Йорк без повторов. Результат исполнения см. на рис. 7.38

Листинг

```
SELECT
 a1.au_fname, a1.au_lname,
 a2.au_fname, a2.au_lname
FROM authors a1
INNER JOIN authors a2
 ON a1.state = a2.state
 AND a1.au_id < a2.au_id
WHERE a1.state = 'NY'
ORDER BY a1.au_id ASC, a2.au_id ASC;
```

| au_fname | au_lname | au_fname  | au_lname |
|----------|----------|-----------|----------|
| -----    | -----    | -----     | -----    |
| Sarah    | Buchman  | Christian | Kells    |

**Рис. 7.38.** Результат выполнения листинга 7.38

Но даже результат на рис. 7.37 не является оптимальным, так как две строки повторяют друг друга. Первая строка сообщает, что Сара Бухман живет в том же штате, что и Кристиан Келлс, а вторая – дублирует эту информацию. Чтобы избежать повторений, в условии второго объединения заменим оператор сравнения «не-равно» на «меньше чем» (см. листинг 7.38 и рис. 7.38).



При использовании синтаксиса WHERE листинг 7.38 будет эквивалентен следующему листингу:

```
SELECT
 a1.au_fname, a1.au_lname,
 a2.au_fname, a2.au_lname
FROM authors a1, authors a2
WHERE a1.state = a2.state
 AND a1.au_id < a2.au_id
 AND a1.state = 'NY'
ORDER BY a1.au_id ASC,
 a2.au_id ASC;
```

## Комбинирование строк с помощью оператора UNION

В последующих трех разделах мы расскажем об операторах UNION, INTERSECT и EXCEPT, предназначенных для преобразования результатов двух команд SELECT в единый результат. Вы можете смешивать эти операторы так, чтобы комбинировать несколько таблиц, а не только две. Оператор UNION широко используется; два других оператора встречаются реже.

Оператор UNION комбинирует результаты двух запросов в один результат, который объединяет строки, считанные двумя запросами (это действие отличается от комбинирования столбцов из двух таблиц). Выражение UNION удаляет из результата повторяющиеся строки; выражение UNION ALL сохраняет повторы. Операторы UNION просты, но имеют ряд ограничений:

- списки столбцов команды SELECT в двух запросах должны включать одинаковое число столбцов (названий столбцов, арифметических выражений, функций и т.д.);
- соответствующие столбцы в двух запросах должны быть заданы в одинаковом порядке;
- если имена соответствующих столбцов совпадают, то их название будет использовано в результате. Если названия соответствующих столбцов различаются, то СУБД самостоятельно определит имя столбца в результате. Большинство СУБД заимствуют названия

столбцов для результата из первого отдельного запроса в команде UNION. Если вы желаете переименовать столбец в результате, используйте предложение AS в первом запросе (см. раздел «Создание псевдонимов столбцов с помощью предложения AS» в главе 4);

- предложение ORDER BY может использоваться только в последнем запросе команды UNION. Эта сортировка применяется к конечному результату после комбинирования. Так как названия столбцов в итоге различаются для разных СУБД, то для указания порядка сортировки проще всего использовать относительные положения столбцов (см. раздел «Сортировка строк с помощью предложения ORDER BY» в главе 4);
- можно задавать предложения GROUP BY и HAVING только в отдельных запросах; их нельзя использовать для изменения конечного результата.

## Комбинирование строк

Введите:

```
select_statement1
UNION [ALL]
select_statement2;
```

*select\_statement1* и *select\_statement2* — это команды SELECT. Вам необходимо задать количество и порядок столбцов для двух команд, а тип данных в соответствующих столбцах должен быть совместимым. Если вы не зададите опцию ALL, то повторяющиеся строки будут удалены из результата.

**Листинг 7.39.** Отобразить список штатов, в которых живут авторы и находятся издательства. Результат исполнения см. на рис. 7.39

```
SELECT state FROM authors
UNION
SELECT state FROM publishers;
```

```
state

NULL
CA
CO
FL
NY
```

**Рис. 7.39.** Результат выполнения листинга 7.39

Листинг 7.39 отображает список штатов, в которых живут авторы и находятся издательства. По умолчанию UNION удаляет из результата повторяющиеся строки. Результат исполнения показан на рис. 7.39.

Листинг 7.40 аналогичен листингу 7.39, однако использует ключевое слово ALL. Поэтому в результат включены все строки, в том числе и повторы (см. рис. 7.40).

**Листинг 7.40.** Отобразить список штатов, в которых живут авторы и находятся издательства (в том числе повторы). Результат исполнения см. на рис. 7.40

```


Листинг
SELECT state FROM authors
UNION ALL
SELECT state FROM publishers;

```

```

state

NY
CO
CA
CA
NY
CA
FL
NY
CA
NULL
CA
```

**Рис. 7.40.** Результат выполнения листинга 7.40

---



**Листинг 7.41.** Отобразить список всех авторов и издательств. Результат исполнения см. на рис. 7.41

```
SELECT au_fname || ' ' || au_lname AS
"Name"
FROM authors
UNION
SELECT pub_name
FROM publishers
ORDER BY 1 ASC;
```

```
Name


Kellsey
Abatis Publishers
Christian Kells
Core Dump Books
Hallie Hull
Klee Hull
Paddy O`Furniture
Sarah Buchman
Schadenfreude Press
Tenterhooks Press
Wendy Heydemark
```

**Рис. 7.41.** Результат выполнения листинга 7.41

Листинг 7.41 отображает список всех авторов и издательств. Предложение AS в первом запросе задает названия для столбцов в результате. Для сортировки результата предложение ORDER BY использует относительное положение столбца вместо его названия. Результат исполнения показан на рис. 7.41.

Листинг 7.42 расширяет листинг 7.41 и задает дополнительный столбец `Type`, с помощью которого определяется, из какой таблицы была считана каждая строка. Условие `WHERE` считывает только авторов и издательства из штата Нью-Йорк. Результат исполнения показан на рис. 7.42.

**Листинг 7.42.** Отобразить список всех авторов и издательств из штата Нью-Йорк, отсортировав их по типу и названию. Результат исполнения см. на рис. 7.42



```
SELECT
 'author' AS "Type",
 au_fname || ' ' || au_lname AS "Name",
 state
FROM authors
WHERE state = 'NY'
UNION
SELECT
 'publisher',
 pub_name,
 state
FROM publishers
WHERE state = 'NY'
ORDER BY 1 ASC, 2 ASC;
```

| Type      | Name              | state |
|-----------|-------------------|-------|
| -----     | -----             | ----- |
| author    | Christian Kells   | NY    |
| author    | Sarah Buchman     | NY    |
| publisher | Abatis Publishers | NY    |

**Рис. 7.42.** Результат выполнения листинга 7.42

**Листинг 7.43.** Отобразить список всех авторов и издательств из штата Нью-Йорк, а также названий книг, опубликованных в этом штате, отсортировав их по типу и названию. Результат исполнения см. на рис. 7.43

```

SELECT
 'author' AS "Type",
 au_fname || ' ' || au_lname AS "Name"
FROM authors
WHERE state = 'NY'
UNION
SELECT
 'publisher',
 pub_name
FROM publishers
WHERE state = 'NY'
UNION
SELECT
 'title',
 title_name
FROM titles t
INNER JOIN publishers p
 ON t.pub_id = p.pub_id
WHERE p.state = 'NY'
ORDER BY 1 ASC, 2 ASC;

```


Листинг 7.43 добавляет третий запрос к листингу 7.42, чтобы считать также названия книг, опубликованных в штате Нью-Йорк. Результат исполнения показан на рис. 7.43.

| Type      | Name                       |
|-----------|----------------------------|
| author    | Christian Kells            |
| author    | Sarah Buchman              |
| publisher | Abatis Publishers          |
| title     | 1977!                      |
| title     | How About Never?           |
| title     | Not Without My Faberge Egg |
| title     | Spontaneous, Not Annoying  |

**Рис. 7.43.** Результат выполнения листинга 7.43

Листинг 7.44 аналогичен листингу 7.43, только отображает количество авторов, издательств и книг в штате Нью-Йорк. Результат исполнения листинга представлен на рис. 7.44.

**Листинг 7.44.** Отобразить количество авторов, издательств из штата Нью-Йорк и книг, опубликованных в этом штате, отсортировав их по типу. Результат исполнения см. на рис. 7.44



```
SELECT
 'author' AS "Type",
 COUNT(au_id) AS "Count"
FROM authors
WHERE state = 'NY'
UNION
SELECT
 'publisher',
 COUNT(pub_id)
FROM publishers
WHERE state = 'NY'
UNION
SELECT
 'title',
 COUNT(title_id)
FROM titles t
INNER JOIN publishers p
 ON t.pub_id = p.pub_id
WHERE p.state = 'NY'
ORDER BY 1 ASC;
```

| Type      | Count |
|-----------|-------|
| author    | 2     |
| publisher | 1     |
| title     | 4     |

**Рис. 7.44.** Результат выполнения листинга 7.44

**Листинг 7.45.** Повысить цены на книги по истории на 10%, а на книги по психологии – на 20%, не меняя цены на другие книги. Результат исполнения листинга показан на рис. 7.45

```

SELECT title_id, type, price,
 price * 1.10 AS "New price"
FROM titles
WHERE type = 'history'
UNION
SELECT title_id, type, price, price * 1.20
FROM titles
WHERE type = 'psychology'
UNION
SELECT title_id, type, price, price
FROM titles
WHERE type NOT IN ('psychology','history')
ORDER BY type ASC, title_id ASC;

```

В листинге 7.45 мы повторяем листинг 5.30. Но вместо условия CASE для изменения цен на книги используем несколько запросов, результат выполнения которых объединен с помощью команды UNION (см. рис. 7.45).

| title_id | type       | price | New price |
|----------|------------|-------|-----------|
| -----    | -----      | ----  | -----     |
| T06      | biography  | 19.95 | 19.95     |
| T07      | biography  | 23.95 | 23.95     |
| T10      | biography  | NULL  | NULL      |
| T12      | biography  | 12.99 | 12.99     |
| T08      | children   | 10.00 | 10.00     |
| T09      | children   | 13.95 | 13.95     |
| T03      | computer   | 39.95 | 39.95     |
| T01      | history    | 21.99 | 24.19     |
| T02      | history    | 19.95 | 21.95     |
| T13      | history    | 29.99 | 32.99     |
| T04      | psychology | 12.99 | 15.59     |
| T05      | psychology | 6.95  | 8.34      |
| T11      | psychology | 7.99  | 9.59      |

**Рис. 7.45.** Результат выполнения листинга 7.45

П

UNION является коммутативной командой: A UNION B — это то же самое, что B UNION A.



В Microsoft Access и Microsoft SQL Server используйте символ «+», чтобы связать строки (см. раздел «Объединение строк с помощью оператора II» в главе 5). Чтобы запустить листинги 7.41–7.43 в Access или SQL Server, измените выражения связи следующим образом:

```
au_fname + ' ' + au_lname
```

П

В SQL операция INTERSECT имеет более высокий приоритет по сравнению с UNION и EXCEPT, но приоритеты для вашей СУБД могут быть другими. Чтобы задать порядок расчета в запросах со смешанными операторами, пользуйтесь круглыми скобками (см. раздел «Определение последовательности вычисления» в главе 5).

С

Для объединения в одной команде UNION и UNION ALL пользуйтесь круглыми скобками, чтобы задать порядок расчетов. Давайте рассмотрим два примера:

```
SELECT * FROM table1
UNION ALL
(SELECT * FROM table2
UNION
SELECT * FROM table3);
```

и

```
(SELECT * FROM table1
UNION ALL
SELECT * FROM table2)
UNION
SELECT * FROM table3;
```

Первая команда удаляет повторы в объединении *table2* и *table3*, но не удаляет их в результате и *table1*. Вторая команда сохраняет повторы в объединении *table1* и *table2*, но удаляет их в последующем объединении с *table3*, поэтому ALL не влияет на конечный результат команды.

П

Результат исполнения команды UNION может быть отсортирован, даже если вы не задали предложение ORDER BY, так как ваша СУБД выполняет внутреннюю сортировку, чтобы идентифицировать и удалить повторяющиеся строки (результаты команды UNION ALL не подвергаются внутренней сортировке).

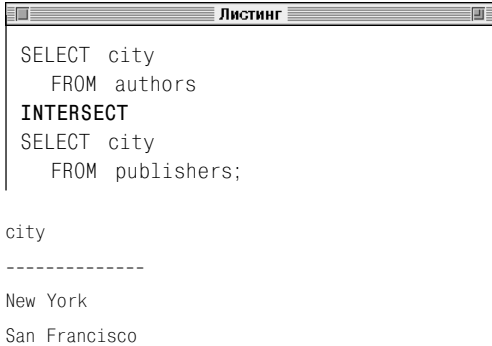
В MySQL пользуйтесь командой CONCAT(), чтобы связать строки (см. раздел «Объединение строк с помощью оператора II» в главе 5). Чтобы запустить листинги 7.41–7.43 в MySQL, измените выражения связи следующим образом:

```
CONCAT(au_fname, ' ', au_lname)
```

В PostgreSQL преобразуйте числа с плавающей запятой в листинге 7.45 в тип DECIMAL (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5). Чтобы запустить листинг 7.45 в PostgreSQL, замените расчеты новых цен на следующие:

```
price * CAST((1.10) AS DECIMAL)
price * CAST((1.10) AS DECIMAL)
```

**Листинг 7.46.** Отобразить список городов, в которых живут авторы и находятся издательства. Результат исполнения см. на рис. 7.46



```

SELECT city
 FROM authors
INTERSECT
SELECT city
 FROM publishers;

city

New York
San Francisco

```

**Рис. 7.46.** Результат выполнения листинга 7.46

## Поиск общих строк с помощью команды INTERSECT

Команда INTERSECT преобразует результаты двух запросов в один результат, который включает все общие строки в результатах выполнения двух запросов. *Пересечения* имеют такие же ограничения, как и объединения union (см. раздел «Комбинирование строк с помощью оператора UNION» ранее в этой главе).

### Поиск общих строк

Введите:

```

select_statement1
INTERSECT
select_statement2;

```

*select\_statement1* и *select\_statement2* — это команды SELECT. Вам необходимо задать количество и порядок столбцов для двух команд, а тип данных в соответствующих столбцах должен быть совместимым. Повторяющиеся строки будут удалены из результата.

Листинг 7.46 использует команду INTERSECT, чтобы отобразить список городов, в которых живут авторы и находятся издательства. Результат исполнения показан на рис. 7.46.

**П** INTERSECT является коммутативной командой: A INTERSECT B — это то же самое, что B INTERSECT A.

**П** В SQL у INTERSECT более высокий приоритет по сравнению с UNION и EXCEPT, но приоритеты в вашей СУБД могут быть другими. Чтобы задать порядок расчета в запросах со смешанными операторами, пользуйтесь круглыми скобками (см. раздел «Определение последовательности вычисления» в главе 5).

**П**

Очень полезно воспринимать UNION как логический вариант OR, а INTERSECT – как логический вариант AND (см. раздел «Комбинирование и отмена условий с помощью операторов AND, OR и NOT» в главе 4). Например, если вы желаете узнать, какие товары поставляются продавцом А, а какие – продавцом В, введите:

```
SELECT product_id
FROM vendor_a_product_list
UNION
SELECT product_id
FROM vendor_b_product_list;
```

Если вы желаете узнать, какие товары поставляются продавцами А и В, введите:

```
SELECT product_id
FROM vendor_a_product_list
INTERSECT
SELECT product_id
FROM vendor_b_product_list;
```

**П**

Если ваша СУБД не поддерживает команду INTERSECT, можно эмулировать ее с помощью INNER JOIN или подзапроса с использованием предложения EXISTS (см. главу 8). Каждая из последующих команд эквивалентна листингу 7.46 (внутреннее объединение):

```
SELECT DISTINCT authors.city
FROM authors
INNER JOIN publishers
ON authors.city =
publishers.city;
```

или (с использованием предложения EXISTS):

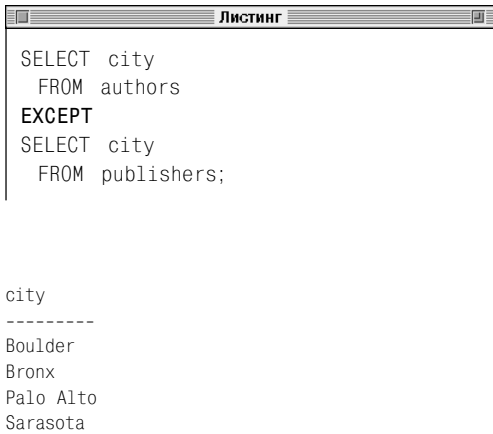
```
SELECT DISTINCT city
FROM authors
WHERE EXISTS
(SELECT *
FROM publishers
WHERE authors.city =
publishers.city);
```



поддерживают команду INTERSECT. Чтобы запустить листинг 7.46 в Access, SQL Server или MySQL, используйте эквивалентный запрос INNER JOIN, о котором мы рассказали в предыдущем примере.



**Листинг 7.47.** Отобразить список городов, в которых живут авторы, но нет издательств. Результат исполнения см. на рис. 7.47



**Рис. 7.47.** Результат выполнения листинга 7.47

## Поиск разных строк с помощью команды EXCEPT

Команда EXCEPT (по-другому ее называют *отличием*) преобразует результаты двух запросов в один результат, который включает только строки первого запроса. Разница между INTERSECT и EXCEPT состоит в том, что A INTERSECT B включает строки из таблицы A, которые повторяются в таблице B, а A EXCEPT B включает строки из таблицы A, которые не повторяются в таблице B. Отличия имеют такие же ограничения, как и объединения union (см. раздел «Комбинирование строк с помощью оператора UNION» ранее в этой главе).

### Поиск разных строк

Введите:

```

select_statement1
EXCEPT
select_statement2;

```

Здесь *select\_statement1* и *select\_statement2* — это предложения SELECT. Вам необходимо задать количество и порядок столбцов для двух команд, а тип данных в соответствующих столбцах должен быть совместимым. Повторяющиеся строки будут удалены из результата.

Листинг 7.47 использует команду EXCEPT, чтобы отобразить список городов, в которых живут авторы, но нет издательств. Результат исполнения показан на рис. 7.47.



В отличие от команд UNION и INTERSECT, команда EXCEPT не является коммутативной: A EXCEPT B — это не то же самое, что B EXCEPT A.

**П**

В SQL у `INTERSECT` более высокий приоритет по сравнению с `UNION` и `EXCEPT`, но приоритеты именно в вашей СУБД могут быть другими. Чтобы задать порядок расчета в запросах со смешанными операторами, пользуйтесь круглыми скобками (см. раздел «Определение последовательности вычисления» в главе 5).

**П**

Если ваша СУБД не поддерживает `EXCEPT`, можете продублировать ее с помощью внешнего запроса, запроса `NOT EXISTS` или `NOT IN` (см. главу 8). Каждая из последних команд эквивалентна листингу 7.47 (внешнее объединение):

```
SELECT DISTINCT a.city
 FROM authors a
 LEFT OUTER JOIN publishers p
 ON a.city = p.city
 WHERE p.city IS NULL;
```

**или (запрос NOT EXISTS)**

```
SELECT DISTINCT city
 FROM authors
 WHERE NOT EXISTS
 (SELECT *
 FROM publishers
 WHERE authors.city =
 publishers.city);
```

**или (запрос NOT IN)**

```
SELECT DISTINCT city
 FROM authors
 WHERE city NOT IN
 (SELECT city
 FROM publishers);
```



СУБД Microsoft Access, Microsoft SQL Server и MySQL не поддерживают команду `EXCEPT`. Чтобы запустить листинг 7.47 в Access, SQL Server или MySQL, используйте эквивалентный запрос `OUTER JOIN`, о котором мы рассказали в предыдущем запросе.

В Oracle оператор `EXCEPT` называется `MINUS`. Чтобы запустить листинг 7.47 в Oracle, введите:

```
SELECT city FROM authors
MINUS
SELECT city FROM publishers;
```

До настоящего момента для считывания данных из одной или нескольких таблиц мы использовали единичную команду `SELECT`. В этой главе мы рассмотрим запросы разных уровней, которые позволяют считывать или изменять данные на основании результатов другого запроса.

*Подзапрос* — это команда `SELECT`, встроенная в другую команду `SQL`. Вы можете располагать подзапрос в:

- предложении `SELECT`, `FROM`, `WHERE` или `HAVING` другой команды `SELECT`;
- другом подзапросе;
- команде `INSERT`, `UPDATE` или `DELETE`.

Допускается использовать подзапрос в любом доступном месте в `SQL`-команде. Однако ваша СУБД может запрещать располагать подзапросы в ряде мест. В этой главе мы расскажем о подзапросах, которые размещаются в команде `SELECT` или других запросах. В главе 9 будет рассказано о подзапросах, которые используются в командах `INSERT`, `UPDATE` и `DELETE`.

---

## Принципы работы с подзапросами

В этом разделе мы рассмотрим ряд терминов и приведем пример команды SELECT, которая включает простой подзапрос. В следующих разделах будет рассказано о разных типах запросов, а также их структуре и семантике.

Предположим, что вы желаете привести список издателей биографий. Первый способ сделать это состоит в том, чтобы написать два запроса: один будет считывать информацию обо всех издателях биографий (см. листинг 8.1 и рис. 8.1), а другой – приводить список результатов первого запроса (см. листинг 8.2 и рис. 8.2).

Более удобный способ – использование команды INNER JOIN (см. листинг 8.3 и рис. 8.3), о котором рассказывается в разделе «Создание внутреннего объединения с помощью команды INNER JOIN» в главе 7.

Также вы можете использовать подзапрос (см. листинг 8.4 и рис. 8.4), отмеченный полужирным шрифтом. Подзапрос еще называется *внутренним запросом*, а команда, которая включает такой запрос, – *внешним запросом*. Другими словами, вложенный запрос является внутренним для внешнего запроса. Помните, что подзапрос может находиться в другом подзапросе; поэтому понятия «внешний» и «внутренний» являются относительными для команд с несколькими запросами.

В разделе «Простые и сложные запросы» далее в этой главе мы опишем, как СУБД

**Листинг 8.1.** Отобразить список издателей биографий. Результаты показаны на рис. 8.1

```

Листинг
SELECT pub_id
FROM titles
WHERE type = 'biography';

```

```

pub_id

P01
P03
P01
P01

```

**Рис. 8.1.** Результат исполнения листинга 8.1. Чтобы привести список только один раз, можете добавить к пункту SELECT команду DISTINCT (см. раздел «Удаление повторяющихся строк с помощью ключевого слова DISTINCT» в главе 4)

**Листинг 8.2.** Этот запрос использует результат листинга 8.1, чтобы отобразить названия всех издателей биографий. Результаты см. на рис. 8.2

```

Листинг
SELECT pub_name
FROM publishers
WHERE pub_id IN ('P01', 'P03');

```

```

pub_name

Abatis Publishers
Schadenfreude Press

```

**Рис. 8.2.** Результат выполнения листинга 8.2

**Листинг 8.3.** С помощью команды `INNER JOIN` вы можете отобразить всех издателей биографий. Результат см. на рис. 8.3

```

Листинг
SELECT DISTINCT pub_name
 FROM publishers p
 INNER JOIN titles t
 ON p.pub_id = t.pub_id
 WHERE t.type = 'biography';

```

```

pub_name

Abatis Publishers
Schadenfreude Press

```

**Рис. 8.3.** Результат исполнения листинга 8.3

**Листинг 8.4.** С помощью подзапроса вы можете отобразить названия издателей биографий. Результат см. на рис. 8.4

```

Листинг
SELECT pub_name
 FROM publishers
 WHERE pub_id IN
 (SELECT pub_id
 FROM titles
 WHERE type = 'biography');

```

```

pub_name

Abatis Publishers
Schadenfreude Press

```

**Рис. 8.4.** Результат исполнения листинга 8.4

выполняет запросы. Пока вам достаточно знать, что в листинге 8.4 СУБД сначала обрабатывает внутренний запрос (помечен жирным шрифтом), затем использует его результат для запуска другого запроса и получает конечный результат. Ключевое слово `IN` вводит подзапросы для списка и работает так же, как слово `IN` в разделе «Фильтрация с помощью оператора `IN`» в главе 4. Обратите внимание, что внутренний запрос в листинге 8.4 такой же, как в листинге 8.1, а внешний запрос – такой же, как в листинге 8.2

**П** Вы можете заметить, что термин «подзапрос» используется для обозначения целой команды SQL, которая включает один или несколько запросов. Чтобы избежать ошибок, в данной книге подобная терминология не применяется.



MySQL 4.0 и более ранних версий не поддерживает подзапросы (хотя MySQL 4.1 должна их поддерживать). Примеры запросов, описанные в данной главе, не будут работать в MySQL 4.0 и более ранних версий. Вы можете решить эту проблему тремя способами: заменить запрос на объединение (см. раздел «Подзапросы и объединения» далее в этой главе), создать временную таблицу для результатов запроса (см. раздел «Создание временной таблицы с помощью команды `CREATE TEMPORARY TABLE`» в главе 10 и примеры временных таблиц в примечании **DBMS** к разделу «Создание внешних объединений с помощью команды `OUTER JOIN`» в главе 7, листинг 7.32) либо создать имитацию запроса с помощью языка хоста, например Perl, PHP или Java (в этой книге языки программирования не рассматриваются).

## Структура подзапросов

Структура подзапросов такая же, как структура обычной команды `SELECT` (см. главы 4–7). Отличия заключаются в следующем:

- вы можете поместить подзапрос в предложение `SELECT`, `FROM`, `WHERE`, `HAVING` или в другой запрос;
- подзапрос всегда должен заключаться в круглые скобки;
- не заканчивайте подзапрос точкой с запятой (вам следует завершить точкой с запятой команду, которая включает подзапрос);
- не помещайте в подзапрос предложение `ORDER BY` (подзапрос выдает мгновенный результат, который вы не видите; сортировка запроса бессмысленна);
- подзапрос включает одну команду `SELECT` (вы не можете использовать в качестве подзапроса несколько команд `SELECT`);
- подзапрос может использовать столбцы в таблицах, которые приводятся в предложении `FROM` самого подзапроса или другого подзапроса;
- если таблица появляется во внутреннем, а не во внешнем запросе, вы не сможете включить столбцы этой таблицы в конечный результат (то есть в предложение `SELECT` внешнего запроса);
- в зависимости от ситуации подзапрос может использоваться для считывания ограниченного количества строк или столбцов. В соответствии со стандартом SQL определяет запрос по числу строк и столбцов, которые он считывает (см. табл. 8.1). Во всех случаях подзапрос может также считывать пустую таблицу.

Таблица 8.1. Сравнение результатов запросов

| Запрос низшего уровня | Строки | Столбцы |
|-----------------------|--------|---------|
| Скалярный запрос      | 1      | 1       |
| Запрос строки         | 1      | ≥1      |
| Запрос столбца        | ≥1     | ≥1      |

Чаще всего вы будете работать с подзапросами в предложении `WHERE`, которые будут принимать одну из следующих форм:

- `WHERE test_expr op (subquery);`
- `WHERE test_expr [NOT] IN (subquery);`
- `WHERE test_expr op ALL (subquery);`
- `WHERE test_expr op ANY (subquery);`
- `WHERE [NOT] EXISTS (subquery);`

*test\_expr* является буквенным значением, названием столбца, выражением или запросом; *op* – это оператор сравнения (`=`, `<>`, `<`, `<=`, `>` или `>=`); *subquery* – простой или сложный запрос. Далее в этой главе мы рассмотрим все формы. Вы также можете их использовать в предложении `HAVING`.



SQL не задает максимальное количество подуровней для запросов, поэтому ваша СУБД присвоит свое ограничение. Как правило, это встроенное ограничение очень велико. Например, Microsoft SQL Server допускает 32 подуровня.

**Листинг 8.5а.** Эта команда использует подзапрос, чтобы отобразить список всех авторов, которые живут в городе, где находится издатель. Результат см. на рис. 8.5

```

SELECT au_id, city
 FROM authors
 WHERE city IN
 (SELECT city FROM publishers);

```

**Листинг 8.5б.** Эта команда эквивалентна листингу 8.5а, но вместо подзапроса использует внутреннее объединение. Результат см. на рис. 8.5

```

SELECT a.au_id, a.city
 FROM authors a
 INNER JOIN publishers p
 ON a.city = p.city;

```

| au_id | city          |
|-------|---------------|
| A03   | San Francisco |
| A04   | San Francisco |
| A05   | New York      |

**Рис. 8.5.** Результат выполнения листингов 8.5а и 8.5б

## Подзапросы и объединения

В разделе «Принципы работы с подзапросами» ранее в этой главе (листинги 8.3 и 8.4) были рассмотрены два эквивалентных запроса: один использует объединение, а другой – подзапрос. Многие запросы могут быть преобразованы в объединения. Запрос – это лишь способ сгруппировать одну таблицу с другой, не создавая при этом объединения.

Поскольку создавать и исполнять подзапросы иногда затруднительно, вы можете предпочесть им объединения. Однако некоторые задачи вы сумеете решить только с помощью подзапросов. Как правило, нет никакой разницы между командой, которая использует подзапрос, и ее эквивалентом, который использует объединение (но бывают и исключения – см. раздел «Сравнение эквивалентных запросов» далее в этой главе).

На представленные ниже запросы приведены эквивалентные команды, которые используют подзапросы и объединения, – запрос с использованием в условии оператора IN:

```

SELECT *
 FROM table1
 WHERE id IN
 (SELECT id FROM table2);

```

и внутреннее объединение:

```

SELECT table1.*
 FROM table1
 INNER JOIN table2
 ON table1.id = table2.id;

```

В качестве примера см. листинги 8.5а и 8.5б, а также рис. 8.5.

Следующие три команды тоже эквивалентны (подзапрос NOT IN):

```
SELECT *
 FROM table1
 WHERE id NOT IN
 (SELECT id FROM table2);
```

и (подзапрос NOT EXISTS):

```
SELECT table1.*
 FROM table1
 WHERE NOT EXISTS
 (SELECT id FROM table2);
```

и (внешнее объединение)

```
SELECT table1.*
 FROM table1
 LEFT OUTER JOIN table2
 ON table1.id = table2.id
 WHERE table2.id IS NULL;
```

В качестве примера см. листинги 8.6а, 8.6б и 8.6в, а также рис. 8.6. Запросы IN и EXISTS мы рассмотрим далее в этой главе.

**Листинг 8.6а.** Эта команда использует подзапрос с оператором NOT IN, чтобы отобразить список всех авторов, которые не участвовали в написании книг. Результат см. на рис. 8.6

```
Листинг
SELECT au_id, au_fname, au_lname
 FROM authors
 WHERE au_id NOT IN
 (SELECT au_id FROM title_authors);
```

**Листинг 8.6б.** Эта команда эквивалентна листингу 8.6а, но вместо оператора IN использует оператор EXISTS. Результат см. на рис. 8.6

```
Листинг
SELECT au_id, au_fname, au_lname
 FROM authors a
 WHERE NOT EXISTS
 (SELECT *
 FROM title_authors ta
 WHERE a.au_id = ta.au_id);
```

**Листинг 8.6в.** Эта команда эквивалентна листингам 8.6а и 8.6б, но вместо подзапроса используется внешнее объединение. Результат см. на рис. 8.6

```
Листинг
SELECT a.au_id, a.au_fname, a.au_lname
 FROM authors a
 LEFT OUTER JOIN title_authors ta
 ON a.au_id = ta.au_id
 WHERE ta.au_id IS NULL;
```

| au_id | au_fname | au_lname    |
|-------|----------|-------------|
| A07   | Paddy    | O'Furniture |

**Рис. 8.6.** Результат выполнения листингов 8.6а, 8.6б и 8.6в



**Листинг 8.7а.** Эта команда использует подзапрос, чтобы отобразить список всех авторов, которые живут с автором A04 (Klee Hull) в одном штате. Результат см. на рис. 8.7

```

Листинг
SELECT au_id, au_fname, au_lname, state
FROM authors
WHERE state IN
 (SELECT state
 FROM authors
 WHERE au_id = 'A04');
```

**Листинг 8.7б.** Эта команда эквивалентна листингу 8.7а, но вместо подзапроса использует внутреннее объединение. Результат см. на рис. 8.7

```

Листинг
SELECT a1.au_id, a1.au_fname,
 a1.au_lname, a1.state
FROM authors a1
INNER JOIN authors a2
 ON a1.state = a2.state
WHERE a2.au_id = 'A04';
```

| au_id | au_fname | au_lname | state |
|-------|----------|----------|-------|
| A03   | Hallie   | Hull     | CA    |
| A04   | Klee     | Hull     | CA    |
| A06   |          | Kellsey  | CA    |

**Рис. 8.7.** Результат выполнения листингов 8.7а и 8.7б

**П**

В качестве подзапроса вы также можете использовать самообъединение (см. листинги 8.7а, 8.7б и рис. 8.7). За дополнительной информацией о самообъединениях обращайтесь к разделу «Создание самообъединения» в главе 7.

**П**

Можно в любой момент преобразовать внутреннее объединение в подзапрос, но не наоборот. Это происходит потому, что внутренние объединения коммутативны; вы можете объединять таблицы А и В в любом порядке и получать один и тот же результат. Подзапросы не имеют подобного свойства (вы можете в любой момент преобразовать внешнее объединение в запрос, хотя внешние объединения не коммутативны).

## П

Подзапросы очень ценны в том случае, если вам необходимо сравнить среднее значение с другими значениями (см. листинг 8.8 и рис. 8.8). Если вы не пользуетесь подзапросом, вам придется использовать две команды `SELECT`, чтобы отобразить все книги, которые имеют самую высокую цену: один запрос, чтобы найти самую высокую цену, и один запрос, чтобы отобразить все книги, которые продаются по этой цене. За дополнительной информацией об агрегатных функциях обращайтесь к главе 6.

## П

Если в результат вы включаете столбцы из нескольких таблиц, следует использовать объединение. Листинг 8.5б использует объединение, чтобы считать список авторов, которые живут в городе, где находится издательство. Чтобы включить в результат информацию об издателе, к списку столбцов команды `SELECT` мы добавили столбец `pub_id` (см. листинг 8.9 и рис. 8.9).

Выполнить эту задачу с помощью подзапроса не представляется возможным, так как нельзя включить столбец из таблицы, которая находится только во внешнем запросе, в список столбцов команды `SELECT` внешнего запроса:

```
SELECT a.au_id, a.city, p.pub_id
FROM authors a
WHERE a.city IN
(SELECT p.city FROM publishers p); --
неправильно.
```



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Принципы работы с подзапросами» ранее в этой главе.

**Листинг 8.8.** Эта команда отобразит список всех книг, цена на которые соответствует самой высокой стоимости. Результат показан на рис. 8.8

| Листинг                                                                                                   |       |
|-----------------------------------------------------------------------------------------------------------|-------|
| <pre>SELECT title_id, price FROM titles WHERE price =       (SELECT MAX(price)        FROM titles);</pre> |       |
| title_id                                                                                                  | price |
| -----                                                                                                     | ----- |
| T03                                                                                                       | 39.95 |

**Рис. 8.8.** Результат выполнения листинга 8.8

**Листинг 8.9.** Эта команда отобразит список всех авторов, которые живут в городе, где находится издательство. Информация об издателе будет включена в результат (см. рис. 8.9)

| Листинг                                                                                                |               |        |
|--------------------------------------------------------------------------------------------------------|---------------|--------|
| <pre>SELECT a.au_id, a.city, p.pub_id FROM authors a INNER JOIN publishers p ON a.city = p.city;</pre> |               |        |
| au_id                                                                                                  | city          | pub_id |
| -----                                                                                                  | -----         | -----  |
| A03                                                                                                    | San Francisco | P02    |
| A04                                                                                                    | San Francisco | P02    |
| A05                                                                                                    | New York      | P01    |

**Рис. 8.9.** Результат выполнения листинга 8.9

## Простые и сложные запросы

Можно использовать два типа запросов – *простые* и *сложные*.

Простой подзапрос – это запрос, который может рассматриваться независимо от его внешнего запроса и обрабатывается только один раз для всей команды. Все примеры запросов, которые приводились ранее в этой главе, представляют собой простые запросы (за исключением листинга 8.66).

Сложный подзапрос не может рассматриваться отдельно от внешнего запроса; он представляет собой внутренний запрос, который зависит от данных внешнего запроса. Сложный запрос используется в том случае, если команде необходимо обработать таблицу во внутреннем запросе для каждой строки внешнего запроса. Сложные запросы имеют более сложную структуру и другой порядок исполнения, чем простые запросы. Вы можете использовать сложные запросы для решения задач, которые нельзя решить с помощью простых запросов или объединений. В этом разделе мы приведем пример простого и сложного запросов, а затем расскажем, как СУБД их исполняет. Последующие разделы данной главы содержат примеры запросов разного типа.

### Простые запросы

Чтобы рассчитать простой запрос, СУБД запускает внутренний запрос один раз, затем помещает его результат во внешний запрос. Простой запрос выполняется до и независимо от внешнего запроса.

Давайте вспомним листинг 8.5а, который мы рассматривали ранее в этой главе. Листинг 8.10 (идентичен листингу 8.5а) использует простой запрос, чтобы отобразить список всех авторов, которые живут

в том городе, где расположено издательство. Результат исполнения листинга показан на рис. 8.10.

СУБД обрабатывает запрос в два этапа в виде двух разных команд SELECT:

1. Внутренний запрос (простой запрос) считывает список всех городов, в которых находятся издательства (см. листинг 8.11 и рис. 8.11).
2. СУБД передает значения, полученные внутренним запросом при исполнении пункта 1, во внешний запрос, который находит информацию об авторах в соответствии с городами, где расположены издательства (см. листинг 8.12 и рис. 8.12).

**Листинг 8.10.** Эта команда отобразит список всех авторов, которые живут в городе, где находится издательство. Информация об издателе будет включена в результат (см. рис. 8.9)

```
Листинг
SELECT au_id, city
 FROM authors
 WHERE city IN
 (SELECT city
 FROM publishers);
```

| au_id | city          |
|-------|---------------|
| A03   | San Francisco |
| A04   | San Francisco |
| A05   | New York      |

**Рис. 8.10.** Результат выполнения листинга 8.9


**Листинг 8.11.** Эта команда отобразит список всех городов, в которых находятся издательства. Результат показан на рис. 8.11

```
Листинг
SELECT city
 FROM publishers;
```

| city          |
|---------------|
| New York      |
| San Francisco |
| Hamburg       |
| Berkley       |

**Рис. 8.11.** Результат выполнения листинга 8.11

**Листинг 8.12.** Эта команда отобразит список всех авторов, которые живут в одном из городов, считанных с помощью листинга 8.11. Результат см. на рис. 8.12



```
SELECT au_id, city
FROM authors
WHERE city IN
('New York', 'San Francisco',
'Hamburg', 'Berkley');
```

| au_id | city          |
|-------|---------------|
| A03   | San Francisco |
| A04   | San Francisco |
| A05   | New York      |

**Рис. 8.12.** Результат выполнения листинга 8.12

## Сложные запросы

Сложные запросы предполагают более сложный механизм извлечения данных по сравнению с простыми запросами. Перечислим основные параметры, которые характеризуют сложный запрос:

- отличается от простого порядком исполнения, а также количеством исполнений;
- не может быть исполнен отдельно от внешнего запроса, так как зависит от него;
- выполняется несколько раз — один раз для каждой строки, которая была выбрана с помощью внешнего запроса;
- всегда обращается к таблице, указанной в предложении `FROM` внешнего запроса;
- использует названия столбцов, чтобы ссылаться на значения, указанные во внешнем запросе. При работе со сложными запросами такие столбцы называются корреляционными переменными. За дополнительной информацией по названиям столбцов и таблиц обращайтесь к разделам «Уточнение имен столбцов» и «Создание псевдонимов таблиц с помощью предложения `AS`» в главе 7;
- базовая структура запроса, который содержит сложный подзапрос, такова:

```
SELECT outer_columns
FROM outer_table
WHERE outer_column_value IN
(SELECT inner_column
FROM inner_table
WHERE inner_column = outer_column)
```

Исполнение всегда начинается с внешнего запроса, который выбирает каждую отдельную строку в *outer\_table*. Для каждой выбранной строки СУБД исполняет сложный внутренний запрос (выделен полужирным шрифтом) один раз и помечает те строки в *inner\_table*, которые соответствуют условию `WHERE` для значения *outer\_column\_value*. СУБД протестирует условие `WHERE` для отмеченных строк в *inner\_table* и отобразит строки, которые соответствуют условию. Этот процесс будет продолжаться до тех пор, пока все строки, выбранные с помощью внешнего запроса, не будут обработаны.

Листинг 8.13 использует сложный запрос для того, чтобы отобразить список книг, уровень продаж которых выше среднего уровня продаж книг соответствующего типа (результат исполнения листинга см. на рис. 8.13). *candidate* (после пункта *titles* во внешнем запросе) и *average* (после пункта *titles* во внутреннем запросе) представляют собой названия таблицы для *titles*. При этом информация может обрабатываться таким образом, как будто она считывается из двух разных таблиц (см. раздел «Создание самообъединения» в главе 7).

В листинге 8.13 подзапрос не может рассматриваться отдельно от внешнего запроса. Для запроса необходимо значение переменной *candidate.type*. Это значение является корреляционной переменной, которая изменяется, в то время как СУБД проверяет различные строки в таблице *candidate*. Столбец *average.type* должен соответствовать *candidate.type* во внешнем запросе. Средний уровень продаж книг этого типа рассчитывается в подзапросе с использованием типов всех книг из таблицы внешнего запроса (*candidate*). Подзапрос рассчитывает средний уровень продаж книг этого типа, затем сравнивает

**Листинг 8.13.** Эта команда отобразит список всех книг, продажи которых равны или превышают средний уровень продаж книг аналогичного типа. Корреляционная переменная *candidate.type* определяет внутреннее условие, которому должны соответствовать строки внутренней таблицы *average*. Внешнее условие `WHERE (sales >=)` определяет конечное условие, которому должны соответствовать строки внутренней таблицы *average*. Результат показан на рис. 8.13

Листинг

```

SELECT
 candidate.title_id,
 candidate.type,
 candidate.sales
FROM titles candidate
WHERE sales >=
 (SELECT AVG(sales)
 FROM titles average
 WHERE average.type = candidate.type);

```

| title_id | type       | sales   |
|----------|------------|---------|
| T02      | history    | 9566    |
| T03      | computer   | 25667   |
| T05      | psychology | 201440  |
| T07      | biography  | 1500200 |
| T09      | children   | 5000    |
| T13      | history    | 10467   |

**Рис. 8.13.** Результат выполнения листинга 8.13

его со строкой в таблице `candidate`. Если продажи в таблице `candidate` выше или равны среднему уровню продаж, эта книга будет отображена в результате. СУБД рассчитывает запрос следующим образом:

1. Тип книги в первой строке `candidate` используется в подзапросе для расчета среднего уровня продаж. Рассмотрим строку для книги T01, типом которой является история. При этом значение в столбце `type` в первой строке таблицы `candidate` будет `history`. Запрос низшего уровня примет вид:

```
SELECT AVG(sales)
FROM titles average
WHERE average.type = 'history';
```

При исполнении этого запроса вы получите значение 6,866 – средний уровень продаж книг по истории. Во внешнем запросе уровень продаж книги T01 (566) сравнивается со средним уровнем продаж книг по истории. Продажи книги T01 ниже среднего уровня, поэтому в результате книга T01 показана не будет.

2. Далее рассчитывается строка книги T01 в таблице `candidate`.

T01 – тоже книга по истории, поэтому запрос низшего уровня такой же, как в пункте 1:

```
SELECT AVG(sales)
FROM titles average
WHERE average.type = 'history';
```

При исполнении этого запроса вы получите значение 6,866 – средний уровень продаж книг по истории. Во внешнем запросе уровень продаж книги T02 (9,566) сравнивается со средним уровнем продаж книг по истории. Продажи книги T02 выше среднего уровня, поэтому в результате книга T02 будет показана.

3. Далее рассчитывается строка книги T03 в таблице `candidate`.

T03 – книга по компьютерам, поэтому подзапрос примет вид:

```
SELECT AVG(sales)
FROM titles average
WHERE average.type = 'computer';
```

При исполнении этого запроса вы получите значение 25,667 – средний уровень продаж книг по компьютерам. Продажи книги T03 (25,667) равны среднему уровню (это единственная книга по компьютерам), поэтому в результате книга T03 будет показана.

4. СУБД будет повторять расчет до тех пор, пока не проверит все строки в таблице `candidate`.



Если вы можете получить одинаковый результат с использованием как простого, так и сложного запросов, мы рекомендуем пользоваться простым, так как он обрабатывается быстрее. Листинги 8.14a и 8.14b содержат два эквивалентных запроса, которые отображают список всех авторов, получающих 100% (1.0) гонорара за книгу. Листинг 8.14a (использует простой запрос) более эффективен, чем листинг 8.14b, который использует сложный запрос. В простом запросе СУБД считывает внутреннюю таблицу `title_authors` один раз. В сложном запросе СУБД должна считать `title_authors` пять раз – по одному разу для каждой строки во внешней таблице `authors`. Результат исполнения показан на рис. 8.14. Почему мы говорим, что команда, использующая сложный запрос, будет, вероятно, работать медленнее, чем эквивалентная команда, использующая простой запрос? Ведь очевидно, что сложный запрос выполняет более трудную работу. Причина в том, что ваша СУБД может распознать сложный запрос и преобразовать его в эквивалентный простой запрос перед тем, как исполнить команду. За дополнительной информацией обращайтесь к разделу «Сравнение эквивалентных запросов» далее в этой главе.



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Принципы работы с подзапросами» ранее в этой главе. В PostgreSQL вам следует конвертировать числа с плавающей запятой в листингах 8.14а и 8.14б в тип `DECIMAL` (см. раздел «Преобразование типов данных с помощью функции `CAST()`» в главе 5). Чтобы запустить листинги 8.14а и 8.14б в PostgreSQL, замените числа с плавающей запятой в каждом листинге на `CAST (1.0 AS DECIMAL)`.

**Листинг 8.14а.** Эта команда использует простой запрос, чтобы отобразить список всех авторов, которые получают 100% (1.0) гонорара за книгу. Результат см. на рис. 8.14

```

Листинг
SELECT au_id, au_fname, au_lname
FROM authors
WHERE au_id IN
 (SELECT au_id
 FROM title_authors
 WHERE royalty_share = 1.0);

```

**Листинг 8.14б.** Эта команда эквивалентна листингу 8.14а, но вместо простого запроса использует сложный. Он, вероятно, будет исполнен медленнее, чем листинг 8.14а. Результат показан на рис. 8.14

```

Листинг
SELECT au_id, au_fname, au_lname
FROM authors
WHERE 1.0 IN
 (SELECT royalty_share
 FROM title_authors
 WHERE title_authors.au_id =
 authors.au_id);

```

| au_id | au_fname  | au_lname  |
|-------|-----------|-----------|
| A01   | Sarah     | Buchman   |
| A02   | Wendy     | Heydemark |
| A04   | Klee      | Hull      |
| A05   | Christian | Kells     |
| A06   |           | Kellsey   |

**Рис. 8.14.** Результат выполнения листингов 8.14а и 8.14б



**Листинг 8.15а.** Таблицы `publishers` и `titles` содержат столбец `pub_id`, но в этом запросе вам не нужно точно его указывать, так как SQL определяет названия таблиц. Результат показан на рис. 8.15

```

Листинг
SELECT pub_name
 FROM publishers
 WHERE pub_id IN
 (SELECT pub_id
 FROM titles
 WHERE type = 'biography');
```

**Листинг 8.15б.** Этот запрос эквивалентен листингу 8.15а, только название столбца `pub_id` указано точно. Результат представлен на рис. 8.15

```

Листинг
SELECT pub_name
 FROM publishers
 WHERE publishers.pub_id IN
 (SELECT titles.pub_id
 FROM titles
 WHERE type = 'biography');
```

```

pub_name

Abatis Publishers
Schadenfreude Press
```

**Рис. 8.15.** Результат выполнения листингов 8.15а и 8.15б



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к разделу «Принципы работы с подзапросами» ранее в этой главе.

## Определение названий столбцов в подзапросах

Вспомните, что в разделе «Уточнение имен столбцов» главы 7 вы научились присваивать название столбцу в точном соответствии с названием таблицы. Это позволяет безошибочно определять саму таблицу. В командах, которые содержат подзапросы, названия столбцов определяются в соответствии с тем, какая таблица указана в предложении `FROM` на том же уровне.

Например, в листинге 8.15а названия столбцов указываются точно, что означает:

- столбец `pub_id` в предложении `WHERE` внешнего запроса определяется в соответствии с таблицей `publishers` в предложении `FROM` внешнего запроса;
- столбец `pub_id` в предложении `SELECT` внешнего запроса определяется в соответствии с таблицей `titles` в предложении `FROM` внешнего запроса.

Листинг 8.15б представляет собой листинг 8.15а, в котором столбцы указаны полностью. Результат исполнения листинга см. на рис. 8.15.



Вы не допустите ошибки, если точно обозначите название таблицы.



Вы можете использовать точные обозначения, чтобы обойти установки SQL по умолчанию для названий таблиц и указать, что столбец относится к таблице на другом уровне.



Если название столбца может относиться к нескольким таблицам на одном уровне, вам следует обозначить столбец в соответствии с таблицей (или ее названием).

## Значения null в подзапросах

Будьте внимательны относительно значений null; их присутствие существенно усложняет запросы. Если вы не удалите эти значения вовремя, можете получить неправильный результат.

Запрос может включать сравнение с NULL (обратитесь к разделу «Значение null» в главе 3). Значения null не равны между собой, и вы не можете сказать, соответствует NULL другому значению или нет. Давайте рассмотрим пример, который включает условие NOT IN (см. раздел «Проверка на вхождение во множество с помощью оператора IN» далее в этой главе). Представьте две таблицы, каждая из которых состоит из одного столбца.

Первая таблица называется table1:

```
col

1
2
```

Вторая таблица называется table2:

```
col

1
2
3
```

**Листинг 8.16.** Показать все значения в table2, которых нет в table1. Результат представлен на рис. 8.16

```

Листинг
SELECT col
 FROM table2
 WHERE col NOT IN
 (SELECT col
 FROM table1);

```

```
col

3
```

**Рис. 8.16а.** Результат выполнения листинга 8.16 при условии, что table1 не содержит NULL

```
col

```

**Рис. 8.16б.** Результат выполнения листинга 8.16 при условии, что table1 содержит NULL

Если запустить листинг 8.16, чтобы отобразить все значения из table2, которых нет в table1, получится результат, показанный на рис. 8.16а. Давайте добавим в table1 значение null:

```
col

1
2
NULL
```

При повторном запуске листинга 8.16 мы получим результат, отображенный на рис. 8.16б (пустую таблицу), что логически правильно, но не соответствует ожидаемому результату. Почему таблица пуста? Ответ требует некоторых расчетов.

Мы можем переместить оператор NOT вне запроса, не изменяя при этом листинг 8.16:

```
SELECT col
FROM table2
WHERE NOT col IN
 (SELECT col FROM table2);
```

Оператор IN определяет, соответствует ли значение в table2 любому значению в table1. Можно переписать запрос в более компактном виде:

```
SELECT col
FROM table2
WHERE NOT ((col = 1)
 OR (col = 2)
 OR (col = NULL));
```

Если использовать законы Моргана (см. табл. 4.6 в главе 4), запрос примет следующий вид:

```
SELECT col
FROM table2
WHERE (col <> 1)
 AND (col <> 2)
 AND (col <> NULL);
```

Последняя строка, col <> NULL, всегда неизвестна. Обратитесь к таблице истинности AND (табл. 4.3 в главе 4), и вы увидите, что все условие поиска для предложения WHERE становится неизвестным, а оно всегда отвергает неизвестное условие.

**C**

Чтобы исправить листинг 8.16 и избежать добавления NULL в table1, дополните запрос условием IS NOT NULL (см. раздел «Проверка на значение null с помощью оператора IS NULL» в главе 4):

```
SELECT col
FROM table2
WHERE col NOT IN
 (SELECT col
 FROM table 1
 WHERE col IS NOT NULL);
```



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Принципы работы с подзапросами» ранее в этой главе.

## Использование подзапросов в качестве выражений в списке заголовков столбцов

Из глав 4, 5 и 6 вы узнали, что предложения команды SELECT могут быть отдельными буквами, названиями столбцов или выражениями. SQL также позволяет вставлять в список предложения SELECT подзапросы.

Запрос, который используется в качестве столбца, должен быть скалярным. Помните, что скалярный запрос считывает одно значение, то есть результат расчета одной строки или столбца (см. табл. 8.1). В большинстве случаев вам придется использовать функцию или условие запрета в предложении WHERE запроса, чтобы гарантировать выборку только одной строки.

Структура списка предложений команды SELECT такая же, как структура всех других списков. Исключение состоит в том, что вы можете указать в качестве имени столбца запрос, помещенный в скобки. В листинге 8.17 показаны два простых запроса в виде заголовков столбцов. Эти запросы используются для того, чтобы отобразить каждую биографию, ее цену, среднюю цену всех книг (не только биографий) и разницу в цене между биографией и средней ценой всех книг. Функция AVG() гарантирует, что каждый запрос считает только одно значение. Результат исполнения листинга представлен на рис. 8.17. Помните, что функция AVG() игнорирует нули при расчете среднего значения (см. раздел «Порядок расчета среднего значения с помощью функции AVG()» в главе 6).

**Листинг 8.17.** Отобразить каждую биографию, ее цену, среднюю цену всех книг и разницу в цене между биографией и средней ценой всех книг. Результат показан на рис. 8.17

```

Листинг
SELECT title_id,
 price,
 (SELECT AVG(price) FROM titles)
 AS "AVG(price)",
 price - (SELECT AVG(price) FROM titles)
 AS "Difference"
FROM titles
WHERE type='biography';

```

| title_id | price | AVG(price) | Difference |
|----------|-------|------------|------------|
| T06      | 19.95 | 18.3875    | 1.5625     |
| T07      | 23.95 | 18.3875    | 5.5625     |
| T10      | NULL  | 18.3875    | NULL       |
| T12      | 12.99 | 18.3875    | -5.3975    |

**Рис. 8.17.** Результат выполнения листинга 8.17

**Листинг 8.18.** Отобразить список всех авторов всех книг в одной строке. Результат см. на рис. 8.18

```

SELECT title_id,
 (SELECT au_id
 FROM title_authors ta
 WHERE au_order = 1
 AND title_id = t.title_id)
 AS "Author 1",
 (SELECT au_id
 FROM title_authors ta
 WHERE au_order = 2
 AND title_id = t.title_id)
 AS "Author 2",
 (SELECT au_id
 FROM title_authors ta
 WHERE au_order = 3
 AND title_id = t.title_id)
 AS "Author 3"
FROM titles t;

```

| title_id | Author 1 | Author 2 | Author 3 |
|----------|----------|----------|----------|
| T01      | A01      | NULL     | NULL     |
| T02      | A01      | NULL     | NULL     |
| T03      | A05      | NULL     | NULL     |
| T04      | A03      | A04      | NULL     |
| T05      | A04      | NULL     | NULL     |
| T06      | A02      | ULL      | NULL     |
| T07      | A02      | A04      | NULL     |
| T08      | A06      | NULL     | NULL     |
| T09      | A06      | NULL     | NULL     |
| T10      | A02      | NULL     | NULL     |
| T11      | A06      | A03      | A04      |
| T12      | A02      | NULL     | NULL     |
| T13      | A01      | NULL     | NULL     |

**Рис. 8.18.** Результат выполнения листинга 8.18

Листинг 8.18 использует сложные запросы, чтобы отобразить список авторов всех книг в одной строке в виде списка. Результат исполнения листинга представлен на рис. 8.18. Обратите внимание, что в каждом предложении WHERE SQL точно определяет title\_id при помощи заголовка таблицы ta, который задан в предложении FROM запроса. См. раздел «Определение названий столбцов в подзапросах» ранее в этой главе (если вы желаете научиться работать с данным запросом более эффективно, см. советы и примечания в этом разделе).

В листинге 8.19 мы повторяем листинг 7.30 из главы 7. Однако в этом примере, чтобы отобразить список книг, которые написал каждый автор (один или в соавторстве), вместо внешнего объединения мы используем сложный запрос. Результат исполнения листинга показан на рис. 8.19.

Листинг 8.20 использует сложный запрос, чтобы отобразить список всех авторов и даты публикации последних книг. Чтобы сделать точную ссылку на таблицу, необходимо в каждом запросе, содержащем объединение, определить название каждого столбца. Результат исполнения листинга показан на рис. 8.20.

Листинг 8.21 использует сложный запрос, чтобы рассчитать промежуточную сумму для продаж всех книг. Промежуточная сумма представляет собой простой расчет: мы желаем рассчитать для каждой книги сумму продаж всех книг, которые следуют перед ней в алфавитном порядке. В данном случае мы рассматриваем те книги, заголовки которых в title\_id расположены перед заголовком текущей книги. Обратите внимание, что заголовки таблиц используются для обозначения одной и той же таблицы в двух контекстах. Запрос

считывает сумму продаж всех книг, которые находятся перед текущей книгой в алфавитном порядке. Порядок определяется строкой `t1.title_id`. Результат исполнения листинга показан на рис. 8.21.

**П** Вы также можете использовать запрос в предложении `FROM`. В советах к разделу «Исключение повторных значений с помощью предложения `DISTINCT`» в главе 6 мы применяли запрос `FROM`, чтобы повторить отдельную функцию. Листинг 8.22 использует этот запрос для расчета наибольшего количества книг, написанных любым автором (в том числе в соавторстве). Результат исполнения листинга показан на рис. 8.22. Обратите внимание на то, что внешний запрос использует заголовок столбца (`ta`) и ярлык столбца (`count_titles`), чтобы создать ссылку на результат внутреннего запроса.

**П** Вы также можете использовать запрос в виде выражения в форме столбца в командах `UPDATE`, `INSERT` и `DELETE` (см. главу 9), но не в предложении `ORDER BY`.

**С** Чтобы более эффективно работать с листингом 8.18, используйте выражения `CASE` вместо сложных запросов (см. раздел «Вычисление условных значений с помощью выражения `CASE`» в главе 5):

```
SELECT title_id,
 MIN (CASE au_order WHEN 1
 THEN au_id
 END)
 AS "Author 1",
 MIN (CASE au_order WHEN 2
 THEN au_id
 END)
 AS "Author 2",
 MIN (CASE au_order WHEN 3
 THEN au_id
 END)
 AS "Author 3"
FROM title_authors
GROUP BY title_id
ORDER BY title_id ASC;
```

**Листинг 8.19.** Отобразить список книг, которые написал каждый автор (один или в соавторстве), включая авторов, не написавших ничего. Результат см. на рис. 8.19

| Листинг                                                                                                                                                                    |           |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------|
| <pre>SELECT au_id,        (SELECT COUNT(*)         FROM title_authors ta         WHERE ta.au_id = a.au_id)         AS "Num books" FROM authors a ORDER BY au_id ASC;</pre> |           |
| au_id                                                                                                                                                                      | Num books |
| -----                                                                                                                                                                      | -----     |
| A01                                                                                                                                                                        | 3         |
| A02                                                                                                                                                                        | 4         |
| A03                                                                                                                                                                        | 2         |
| A04                                                                                                                                                                        | 4         |
| A05                                                                                                                                                                        | 1         |
| A06                                                                                                                                                                        | 3         |
| A07                                                                                                                                                                        | 0         |

**Рис. 8.19.** Результат выполнения листинга 8.19



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Принципы работы с подзапросами» ранее в этой главе.

В Microsoft Access вам следует увеличить точность расчета средней цены в листинге 8.17. Для этих целей используйте функцию конвертации — `Cdbl()` (см. примечание **DBMS** в разделе «Преобразование типов данных с помощью функции `CAST()`» в главе 5). Чтобы запустить листинг 8.17 в Access, замените оба параметра `AVG(price)` на `Cdbl(AVG(price))`.

**Листинг 8.20.** Отобразить список всех авторов и даты публикации последних книг. Результат см. на рис. 8.19

```

SELECT au_id,
 (SELECT MAX(pubdate)
 FROM titles t
 INNER JOIN title_authors ta
 ON ta.title_id = t.title_id
 WHERE ta.au_id = a.au_id)
 AS "Latest pub date"
FROM authors a;
```

| au_id | Latest pub date |
|-------|-----------------|
| A01   | 2000-08-01      |
| A02   | 2000-08-31      |
| A03   | 2000-11-30      |
| A04   | 2001-01-01      |
| A05   | 2000-09-01      |
| A06   | 2002-05-31      |
| A07   | NULL            |

**Рис. 8.20.** Результат выполнения листинга 8.19

**Листинг 8.21.** Рассчитать промежуточную сумму продаж всех книг. Результат см. на рис. 8.21

```

SELECT t1.title_id, t1.sales,
 (SELECT SUM(t2.sales)
 FROM titles t2
 WHERE t2.title_id < t1.title_id)
 AS "Running total"
FROM titles t1;
```

| title_id | sales   | Running total |
|----------|---------|---------------|
| T01      | 566     | NULL          |
| T02      | 9566    | 566           |
| T03      | 25667   | 10132         |
| T04      | 13001   | 35799         |
| T06      | 201440  | 48800         |
| T06      | 11320   | 250240        |
| T07      | 1500200 | 261560        |
| T08      | 4095    | 1761760       |
| T09      | 5000    | 1765855       |
| T10      | NULL    | 1770855       |
| T11      | 94123   | 1770855       |
| T12      | 100001  | 1864978       |
| T13      | 10467   | 1964979       |

**Рис. 8.21.** Результат выполнения листинга 8.21

**Листинг 8.22.** Рассчитать наибольшее количество книг, написанных любым автором (в том числе в соавторстве). Результат представлен на рис. 8.22

```

SELECT MAX(ta.count_titles)
→ AS "Max titles"
FROM (SELECT COUNT(*) AS "count_titles"
 FROM title_authors
 GROUP BY au_id) ta;
```

Max titles

**Рис. 8.22.** Результат выполнения листинга 8.22

## Сравнение значений, возвращаемых подзапросом, с использованием операторов сравнения

Вы можете использовать запрос в качестве фильтра в предложении `WHERE` или `HAVING` внешнего запроса с помощью одного из операторов сравнения (`=`, `<>`, `<`, `<=`, `>` или `>=`). Перечислим важные параметры при сравнении запроса:

- операторы сравнения работают так же, как в других выражениях (см. табл. 4.2 в главе 4);
- запрос может быть простым или сложным (см. раздел «Простые и сложные запросы» ранее в этой главе);
- список столбцов команды `SELECT` запроса может включать только одно выражение или название столбца;
- сравниваемые значения должны относиться к одному типу данных или быть конвертируемыми (см. раздел «Преобразование типов данных с помощью функции `CAST()`» в главе 5);
- сравнения могут быть зависимыми и независимыми, что обусловлено конкретной СУБД (см. примечание **DBMS** в разделе «Фильтрация строк с помощью предложения `WHERE`» главы 4);
- запрос должен возвращать одно значение (результат одной строки или столбца). Запрос, который возвращает несколько значений, приводит к ошибке;
- если результат запроса содержит строки со значением `null`, сравнение даст неверный результат.

Сложность при написании этих выражений заключается в том, чтобы запрос возвращал одно значение. Это достигается несколькими способами:

- при использовании агрегатной функции на негруппированной таблице всегда получается одно значение (см. главу 6);
- при использовании объединения с внешним запросом, включающим фильтрацию по ключу, всегда получается одно значение.

### Сравнение значений запроса

В предложении `WHERE` команды `SELECT` печатайте `WHERE test_expr op (subquery)`.

*test\_expr* – это буквенное значение, название столбца, выражение или запрос, который считывает одно значение; *op* – оператор сравнения (`=`, `<>`, `<`, `<=`, `>` или `>=`); *subquery* – скалярный запрос, который возвращает один столбец и ни одной или одну строку.

Если значение, считанное с помощью запроса, соответствует сравнению с *test\_expr*, результат сравнения определяется как истинный. Результат определяется как ложный, если значение запроса не соответствует сравнению. Результат сравнения не определен, если подзапрос не вернул ни одной строки или вернул `NULL`.

При использовании предложения `HAVING` синтаксис имеет такую же структуру:

`HAVING test_expr op (subquery)`

Листинг 8.23 проверяет результат простого запроса на соответствие списку всех авторов, которые живут в штате, где находится издательство Tenterhooks Press. Такое



**Листинг 8.23.** Отобразить список всех авторов, которые живут в штате, где находится издательство Tenterhooks. Результат см. на рис. 8.23

```

SELECT au_id, au_fname, au_lname, state
FROM authors
WHERE state =
 (SELECT state
 FROM publishers
 WHERE pub_name = 'Tenterhooks
Press');
```

| au_id | au_fname | au_lname | state |
|-------|----------|----------|-------|
| A03   | Hallie   | Hull     | CA    |
| A04   | Klee     | Hull     | CA    |
| A06   |          | Kellsey  | CA    |

**Рис. 8.23.** Результат выполнения листинга 8.23

**Листинг 8.24.** Отобразить список всех авторов, которые живут в штате, где находится издательство XXX. Результат показан на рис. 8.24

```

SELECT au_id, au_fname, au_lname, state
FROM authors
WHERE state =
 (SELECT state
 FROM publishers
 WHERE pub_name = 'XXX');
```

| au_id | au_fname | au_lname | state |
|-------|----------|----------|-------|
| ----- | -----    | -----    | ----- |

**Рис. 8.24.** Результат выполнения листинга 8.24

название имеет только одно издательство, поэтому предложение WHERE подзапроса гарантирует, что он вернет одно значение. Результат исполнения листинга показан на рис. 8.23.

Листинг 8.24 аналогичен листингу 8.23, отличается лишь названием издательства. Не существует издательства XXX, поэтому подзапрос возвращает пустую таблицу. Сравнение выполняется с NULL, вот почему окончательный результат пустой (см. рис. 8.24).

Листинг 8.25 отображает список всех книг, продажи которых выше среднего уровня. Запросы, работающие с операторами сравнения, часто используют функции, чтобы считать лишь одно значение. Результат исполнения листинга представлен на рис. 8.25.

**Листинг 8.25.** Отобразить список всех книг, продажи которых выше среднего уровня. Результат см. на рис. 8.25

```

SELECT title_id, sales
FROM titles
WHERE sales >
 (SELECT AVG(sales)
 FROM titles);
```

| title_id | sales   |
|----------|---------|
| T05      | 201440  |
| T07      | 1500200 |

**Рис. 8.25.** Результат выполнения листинга 8.25

Чтобы отобразить список авторов всех книг, продажи которых выше среднего уровня, мы добавили к листингу 8.25 внутреннее объединение (см. листинг 8.26 и рис. 8.26).

Вспомните (см. начало этой главы), что вы можете использовать подзапрос практически в любом месте выражения, поэтому

```
WHERE (subquery) op (subquery)
```

является допустимым выражением.

Левый подзапрос должен считать одно значение. Листинг 8.27 эквивалентен листингу 8.26, но мы убрали внутреннее объединение и заменили его сложным запросом слева от оператора сравнения. Результат исполнения листинга показан на рис. 8.27.

Вы можете включить предложение GROUP BY или HAVING в запрос, если знаете, что в результате будет считано только одно значение. Листинг 8.28 показывает список книг, цены на которые выше, чем на самую дорогую биографию. Результат исполнения листинга представлен на рис. 8.28.

Листинг 8.29 использует запрос в предложении HAVING, чтобы отобразить список издательств, средние продажи которых выше общих средних продаж. Запрос считывает одно значение (среднее для всех продаж). Результат исполнения листинга показан на рис. 8.29.

Листинг 8.30 использует сложный запрос, чтобы отобразить список авторов, гонорар которых меньше, чем самый высокий гонорар любого соавтора книги. Внешний запрос выделяет строки title\_authors (то есть ta1) одну за другой. Подзапрос рассчитает самый высокий гонорар для каждой книги на основании выделенной области внешнего запроса. Для каждого

**Листинг 8.26.** Отобразить с помощью объединения и запроса список авторов всех книг, продажи которых выше среднего уровня. Результат показан на рис. 8.26

```

SELECT ta.au_id, ta.title_id
FROM titles t
INNER JOIN title_authors ta
 ON ta.title_id = t.title_id
WHERE sales >
 (SELECT AVG(sales)
 FROM titles)
ORDER BY ta.au_id ASC, ta.title_id ASC;
```

| au_id | title_id |
|-------|----------|
| ----- | -----    |
| A02   | T07      |
| A04   | T05      |
| A04   | T07      |

**Рис. 8.26.** Результат выполнения листинга 8.26

**Листинг 8.27.** Отобразить с помощью двух запросов список авторов всех книг, продажи которых выше среднего уровня. Результат см. на рис. 8.27

```

SELECT au_id, title_id
FROM title_authors ta
WHERE
 (SELECT AVG(sales)
 FROM titles t
 WHERE ta.title_id = t.title_id)
 >
 (SELECT AVG(sales)
 FROM titles)
ORDER BY au_id ASC, title_id ASC;
```

| au_id | title_id |
|-------|----------|
| ----- | -----    |
| A02   | T07      |
| A04   | T05      |
| A04   | T07      |

**Рис. 8.27.** Результат выполнения листинга 8.27

**Листинг 8.28.** Отобразить список книг, цены на которые выше, чем на самую дорогую биографию. Результат см. на рис. 8.28

Листинг

```
SELECT title_id, price
FROM titles
WHERE price >
 (SELECT MAX(price)
 FROM titles
 GROUP BY type
 HAVING type = 'biography');
```

| title_id | price |
|----------|-------|
| -----    | ----- |
| T03      | 39.95 |
| T13      | 29.99 |

**Рис. 8.28.** Результат выполнения листинга 8.28

**Листинг 8.29.** Отобразить список издательств, средние продажи которых выше общих средних продаж. Результат см. на рис. 8.29

Листинг

```
ELECT pub_id, AVG(sales) AS "AVG(sales)"
FROM titles
GROUP BY pub_id
HAVING AVG(sales) >
 (SELECT AVG(sales)
 FROM titles);
```

| pub_id | AVG(sales) |
|--------|------------|
| -----  | -----      |
| P03    | 506744.33  |

**Рис. 8.29.** Результат выполнения листинга 8.29

возможного значения ta1 СУБД выполняет подзапрос и помещает строку в результат при условии, что гонорар автора меньше, чем заданный максимум. Результат исполнения листинга показан на рис. 8.30.

Листинг 8.31 использует сложный подзапрос, чтобы симитировать предложение GROUP BY и отобразить список всех книг, цена на которые выше, чем средняя цена на книги данного типа. Для каждого возможного значения t1 СУБД выполняет подзапрос и включает строку в результат при условии, что цена в этой строке больше, чем заданный средний уровень. Нет необходимости точно группировать

**Листинг 8.30.** Отобразить список авторов, гонорар которых меньше, чем самый высокий гонорар любого соавтора книги. Результат см. на рис. 8.30

Листинг

```
SELECT ta1.au_id, ta1.title_id,
 ta1.royalty_share
FROM title_authors ta1
WHERE ta1.royalty_share <
 (SELECT MAX(ta2.royalty_share)
 FROM title_authors ta2
 WHERE ta1.title_id = ta2.title_id);
```

| au_id | title_id | royalty_share |
|-------|----------|---------------|
| ----- | -----    | -----         |
| A04   | T04      | 0.40          |
| A03   | T11      | 0.30          |
| A04   | T11      | 0.30          |

**Рис. 8.30.** Результат выполнения листинга 8.30

результат по типу книги, так как строки, для которых рассчитывается средняя цена, задаются предложением WHERE подзапроса. Результат исполнения листинга показан на рис. 8.31.

Листинг 8.32 использует ту же структуру, что и листинг 8.31, чтобы отобразить список всех книг, продажи которых ниже, чем продажи бестселлера данного типа. Результат исполнения листинга представлен на рис. 8.32.

**Листинг 8.31.** Отобразить список всех книг, цена на которые выше, чем средняя цена на книги данного типа. Результат см. на рис. 8.31

```

Листинг
SELECT type, title_id, price
 FROM titles t1
 WHERE price >
 (SELECT AVG(t2.price)
 FROM titles t2
 WHERE t1.type = t2.type)
 ORDER BY type ASC, title_id ASC;

```

| type       | title_id | price |
|------------|----------|-------|
| -----      | -----    | ----- |
| biography  | T06      | 19.95 |
| biography  | T07      | 23.95 |
| children   | T09      | 13.95 |
| history    | T13      | 29.99 |
| psychology | T04      | 12.99 |

**Рис. 8.31.** Результат выполнения листинга 8.31



Если запрос считывает несколько строк, вы можете использовать ключевые слова ALL и ANY, чтобы изменить оператор сравнения, либо ввести запрос с помощью оператора IN.



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Принципы работы с подзапросами» ранее в этой главе.

**Листинг 8.32.** Отобразить список всех книг, продажи которых ниже, чем продажи бестселлера данного типа. Результат см. на рис. 8.32

```

Листинг
SELECT type, title_id, sales
 FROM titles t1
 WHERE sales <
 (SELECT MAX(sales)
 FROM titles t2
 WHERE t1.type = t2.type
 AND sales IS NOT NULL)
 ORDER BY type ASC, title_id ASC;

```

| type       | title_id | sales  |
|------------|----------|--------|
| -----      | -----    | -----  |
| biography  | T06      | 11320  |
| biography  | T12      | 100001 |
| children   | T08      | 4095   |
| history    | T01      | 566    |
| history    | T02      | 9566   |
| psychology | T04      | 13001  |
| psychology | T11      | 94123  |

**Рис. 8.32.** Результат выполнения листинга 8.32

## Проверка на вхождение во множество с помощью оператора IN

В разделе «Фильтрация с помощью оператора IN» в главе 4 вы научились использовать IN в предложении WHERE, чтобы сравнивать значение, значение в столбце или выражение со списком значений. Вы также можете использовать запрос, чтобы создать список. Перечислим важные параметры при проверке на принадлежность значения множеству, представленному подзапросом:

- IN работает со значениями в результате запроса так же, как со списком значений (см. раздел «Фильтрация с помощью оператора IN» в главе 4);
- запрос может быть простым или сложным (см. раздел «Простые и сложные запросы» ранее в этой главе);
- список столбцов команды SELECT запроса может включать только одно выражение или название столбца;
- сравниваемые значения должны относиться к одному типу данных или быть конвертируемыми (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5);
- сравнения могут быть зависимыми и независимыми, смотря какую СУБД вы используете (см. примечание DBMS в разделе «Фильтрация строк с помощью предложения WHERE» в главе 4);
- запрос должен считать одно значение (результат одной строки или столбца). Запрос, который считывает несколько значений, приводит к ошибке;

- вы не можете использовать оператор NOT IN, чтобы проверить на отсутствие значения в списке. Если вы зададите NOT IN, СУБД будет выполнять действие, указанное в команде SQL, при условии, что в результате запроса нет соответствующего значения.

### Проверка на вхождение значения во множество

В предложении WHERE команды SELECT напечатайте WHERE *test\_expr* [NOT] IN (*subquery*).

*test\_expr* – это буквенное значение, название столбца, выражение или запрос, который возвращает значение; *subquery* – запрос, который считывает один столбец и любое количество строк.

Если значение *test\_expr* равно любому значению, полученному с помощью запроса, условие IN задается как истинное. Условие IN задается как ложное, если результат запроса пустой, ни одна строка в результате запроса не соответствует *test\_expr* или если все значения в результате запроса равны NULL. Чтобы инвертировать результат условия, укажите NOT.

Предложение HAVING имеет такую же структуру: HAVING *test\_expr* [NOT] (*subquery*).

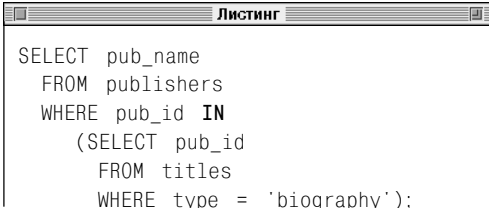
Листинг 8.33 отображает список всех издательств, которые опубликовали биографии. СУБД выполняет эту команду в два этапа. Сначала внутренний запрос считывает информацию обо всех издательствах, которые опубликовали биографии (P01 и P03). Затем СУБД помещает эти значения во внешний запрос, который найдет имена в соответствии с информацией в таблице publishers. Результат исполнения листинга показан на рис. 8.33.

Продemonстрируем версию листинга 8.33 с использованием объединения:

```
SELECT DISTINCT pub_name
 FROM publishers p
 INNER JOIN titles t
 ON p.pub_id = t.pub_id
 AND type = 'biography';
```

Листинг 8.34 эквивалентен листингу 8.33, только он использует NOT IN, чтобы отобразить список всех издательств, которые не публиковали биографии. Результат исполнения листинга см. на рис. 8.34. Команда, представленная в этом листинге, не может быть конвертирована в объединение. Аналогичное объединение имеет другое значение – отображает список всех издательств, которые опубликовали какую-либо книгу, но не биографию.

**Листинг 8.33.** Отобразить список всех издательств, которые опубликовали биографии. Результат показан на рис. 8.33



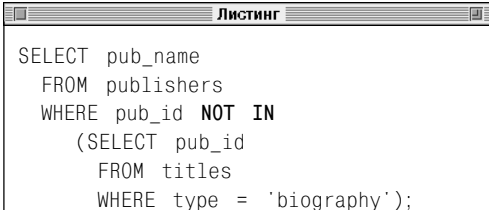
```
SELECT pub_name
 FROM publishers
 WHERE pub_id IN
 (SELECT pub_id
 FROM titles
 WHERE type = 'biography');
```

```
pub_name

Abatis Publishers
Schadenfreude Press
```

**Рис. 8.33.** Результат выполнения листинга 8.33

**Листинг 8.34.** Отобразить список всех издательств, которые не опубликовали биографии. Результат см. на рис. 8.34



```
SELECT pub_name
 FROM publishers
 WHERE pub_id NOT IN
 (SELECT pub_id
 FROM titles
 WHERE type = 'biography');
```

```
pub_name

Core Dump Books
Tenterhooks Press
```

**Рис. 8.34.** Результат выполнения листинга 8.34

**Листинг 8.35.** Отобразить список всех авторов, которые не написали ни одной книги. Результат показан на рис. 8.35

```

Листинг
SELECT au_id, au_fname, au_lname
 FROM authors
 WHERE au_id NOT IN
 (SELECT au_id
 FROM title_authors);

```

| au_id | au_fname | au_lname    |
|-------|----------|-------------|
| ----- | -----    | -----       |
| A07   | Paddy    | O'Furniture |

**Рис. 8.35.** Результат выполнения листинга 8.35

**Листинг 8.36.** Отобразить список всех авторов, которые опубликовали книги в издательстве P03. Результат см. на рис. 8.36

```

Листинг
SELECT DISTINCT a.au_id, au_fname, au_lname
 FROM title_authors ta
 INNER JOIN authors a
 ON ta.au_id = a.au_id
 WHERE title_id IN
 (SELECT title_id
 FROM titles
 WHERE pub_id = 'P03');

```

| au_id | au_fname | au_lname  |
|-------|----------|-----------|
| ----- | -----    | -----     |
| A01   | Sarah    | Buchman   |
| A02   | Wendy    | Heydemark |
| A04   | Klee     | Hull      |

**Рис. 8.36.** Результат выполнения листинга 8.36

Листинг 8.35 эквивалентен листингу 7.31 в главе 7, только он использует запрос вместо внешнего объединения, чтобы отобразить список всех авторов, которые не написали ни одной книги (в том числе в соавторстве). Результат исполнения листинга см. на рис. 8.35.

Листинг 8.36 отображает всех авторов, которые опубликовали книги в издательстве P03 (Schadenfreude Press). Объединение с таблицей authors необходимо, чтобы включить имена авторов (не только информацию о них) в результат. Результат исполнения листинга представлен на рис. 8.36.

Запрос может включать один или несколько подзапросов. В листинге 8.37 приведен список авторов, которые участвовали в написании хотя бы одной биографии. Самый глубокий запрос считывает заголовки T06, T07, T10 и T12. СУБД обрабатывает запрос на более высоком уровне с помощью данных об этих книгах и выдает информацию об авторах. Наконец, запрос самого высокого уровня использует информацию об авторах, чтобы найти их имена. Результат исполнения листинга представлен на рис. 8.37.

Если вы размещаете запрос на большом количестве уровней, это усложняет выполнение команды. Чаще всего проще преобразовать запрос в соединение. Приведем вариант листинга 8.37 с использованием соединения:

```
SELECT DISTINCT a.au_id, au_fname, au_lname
FROM authors a
INNER JOIN title_authors ta
ON a.au_id = ta.au_id
INNER JOIN titles t
ON t.title_id = ta.title_id
WHERE type = 'biography';
```

Листинг 8.38 отображает список соавторов (`au_order > 1`), которые живут в Калифорнии и получают менее 50% гонорара за книгу. Результат исполнения листинга представлен на рис. 8.38. Покажем вариант листинга 8.38 с использованием объединения:

```
SELECT DISTINCT a.au_id, au_fname, au_lname
FROM authors a
INNER JOIN title_authors ta
ON a.au_id = ta.au_id
WHERE state = 'CA'
AND royalty_share < 0.5
AND au_order > 1;
```

**Листинг 8.37.** Отобразить список всех авторов, которые участвовали в написании хотя бы одной биографии. Результат см. на рис. 8.37

| Листинг                                                                                                                                                                             |          |           |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|-----------|
| <pre>SELECT au_id, au_fname, au_lname FROM authors WHERE au_id IN (SELECT au_id FROM title_authors WHERE title_id IN (SELECT title_id FROM titles WHERE type = 'biography'));</pre> |          |           |
| au_id                                                                                                                                                                               | au_fname | au_lname  |
| -----                                                                                                                                                                               | -----    | -----     |
| A02                                                                                                                                                                                 | Wendy    | Heydemark |
| A04                                                                                                                                                                                 | Klee     | Hull      |

**Рис. 8.37.** Результат выполнения листинга 8.37

**Листинг 8.38.** Отобразить список соавторов, которые живут в Калифорнии и получают менее 50% гонорара за книгу. Результат см. на рис. 8.38

| Листинг                                                                                                                                                                      |          |          |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|----------|
| <pre>SELECT au_id, au_fname, au_lname FROM authors WHERE state = 'CA' AND au_id IN (SELECT au_id FROM title_authors WHERE royalty_share &lt; 0.5 AND au_order &gt; 1);</pre> |          |          |
| au_id                                                                                                                                                                        | au_fname | au_lname |
| -----                                                                                                                                                                        | -----    | -----    |
| A03                                                                                                                                                                          | Hallie   | Hull     |
| A04                                                                                                                                                                          | Klee     | Hull     |

**Рис. 8.38.** Результат выполнения листинга 8.38



**Листинг 8.39.** Отобразить список соавторов книги. Результат показан на рис. 8.39

```

ЛИСТИНГ
SELECT au_id, au_fname, au_lname
FROM authors a
WHERE au_id IN
 (SELECT au_id
 FROM title_authors
 WHERE royalty_share < 1.0);

```

| au_id | au_fname | au_lname  |
|-------|----------|-----------|
| ----- | -----    | -----     |
| A02   | Wandy    | Heydemark |
| A03   | Hallie   | Hull      |
| A04   | Klee     | Hull      |
| A06   |          | Kellsey   |

**Рис. 8.39.** Результат выполнения листинга 8.39

**Листинг 8.40.** Отобразить список авторов, у которых нет соавторов. Результат см. на рис. 8.40

```

ЛИСТИНГ
SELECT a.au_id, au_fname, au_lname
FROM authors a
WHERE 1.0 IN
 (SELECT royalty_share
 FROM title_authors ta
 WHERE ta.au_id = a.au_id);

```

| au_id | au_fname  | au_lname  |
|-------|-----------|-----------|
| ----- | -----     | -----     |
| A01   | Sarah     | Buchman   |
| A02   | Wendy     | Heydemark |
| A04   | Klee      | Hull      |
| A05   | Christian | Kells     |
| A06   |           | Kellsey   |

**Рис. 8.40.** Результат выполнения листинга 8.40

Листинг 8.39 отображает список соавторов книги. Чтобы определить, является ли конкретное лицо соавтором книги, проверяется его гонорар за книгу. Если гонорар меньше 100% (1.0), то это соавтор, в противном случае – у автора нет соавторов. Результат исполнения листинга см. на рис. 8.39.

Листинг 8.40 использует сложный запрос, чтобы отобразить список авторов, у которых нет соавторов (то есть тех авторов, которые получили за книгу 100% (1.0) гонорара). Результат исполнения листинга см. на рис. 8.40. СУБД считает каждую строку в таблице внешнего запроса authors как вариант для включения в результат. Когда СУБД проверяет первую строку в authors, она задает корреляционную переменную a.au\_id как равную A01 (Sarah Buchmann) и подставляет это значение во внутренний запрос:

```

SELECT royalty_share
FROM title_authors ta
WHERE ta.au_id = 'A01';

```

Внутренний запрос выдает 1.0, поэтому внешний запрос принимает следующий вид:

```

SELECT a.au_id, au_fname, au_lname
FROM authors a
WHERE 1.0 IN (1.0)

```

Условие WHERE истинно, поэтому автор A01 включается в результат. СУБД повторяет эту процедуру для всех авторов (см. раздел «Простые и сложные запросы» ранее в этой главе).

Листинг 8.41 отображает список авторов, которые написали книги как в соавторстве, так и самостоятельно. Внутренний запрос считывает информацию об авторах, у которых нет соавторов, а внешний запрос сравнивает эту информацию с информацией о соавторах. Результат исполнения листинга показан на рис. 8.41.

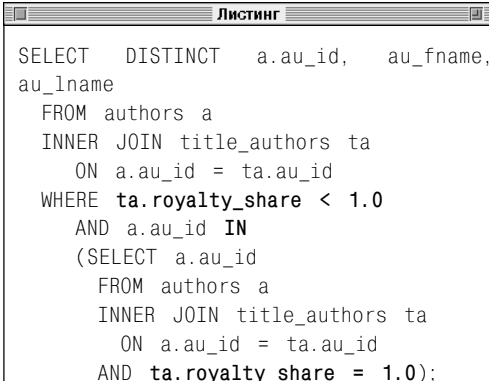
Можно переписать листинг 8.41 в виде объединения или пересечения. Приведем вариант листинга 8.41 с использованием объединения:

```
SELECT DISTINCT a.au_id, au_fname, au_lname
FROM authors a
INNER JOIN title_authors ta1
 ON a.au_id = ta1.au_id
INNER JOIN title_authors ta2
 ON a.au_id = ta2.au_id
WHERE ta1.royalty_share < 1.0
 AND ta2.royalty_share = 1.0;
```

Продемонстрируем вариант листинга 8.41 с использованием пересечения (см. раздел «Поиск общих строк с помощью команды INTERSECT» в главе 7):

```
SELECT DISTINCT a.au_id, au_fname, au_lname
FROM authors a
INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
WHERE ta.royalty_share < 1.0
INTERSECT
SELECT DISTINCT a.au_id, au_fname, au_lname
FROM authors a
INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
WHERE ta.royalty_share = 1.0;
```

**Листинг 8.41.** Отобразить список авторов, которые написали книги как в соавторстве, так и самостоятельно. Результат см. на рис. 8.41



```
SELECT DISTINCT a.au_id, au_fname,
au_lname
FROM authors a
INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
WHERE ta.royalty_share < 1.0
 AND a.au_id IN
 (SELECT a.au_id
 FROM authors a
 INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
 AND ta.royalty_share = 1.0);
```

| au_id | au_fname | au_lname  |
|-------|----------|-----------|
| ----- | -----    | -----     |
| A02   | Wendy    | Heydemark |
| A04   | Klee     | Hull      |
| A06   |          | Kellsey   |

**Рис. 8.41.** Результат выполнения листинга 8.41

**Листинг 8.42.** Отобразить типы книг, которые были опубликованы несколькими издательствами. Результат см. на рис. 8.42

```
Листинг
SELECT DISTINCT t1.type
FROM titles t1
WHERE t1.type IN
 (SELECT t2.type
 FROM titles t2
 WHERE t1.pub_id <> t2.pub_id);

type

biography
history
```

**Рис. 8.42.** Результат выполнения листинга 8.42

Листинг 8.42 использует сложный запрос, чтобы отобразить типы книг, которые были опубликованы несколькими издательствами. Результат исполнения листинга показан на рис. 8.42. Приведем вариант листинга 8.42 с использованием самостоятельного объединения:

```
SELECT DISTINCT t1.type
FROM titles t1
INNER JOIN titles t2
 ON t1.type = t2.type
 AND t1.pub_id <> t2.pub_id;
```

**П** Оператор `IN` эквивалентен ключевому слову `ANY` (см. раздел «Сравнение некоторых значений запроса с помощью ключевого слова `ANY`» далее в этой главе).

**П** Оператор `NOT IN` эквивалентен выражению `<> ALL` (не `<> ANY`) (см. раздел «Сравнение всех значений запроса с помощью ключевого слова `ALL`» далее в этой главе).



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Принципы работы с подзапросами» ранее в этой главе.

Чтобы запустить листинг 8.41 в Microsoft Access, напечатайте:

```
SELECT DISTINCT a.au_id, au_fname,
au_lname
FROM (authors AS a
 INNER JOIN title_authors AS ta1
 ON a.au_id = ta1.au_id)
 INNER JOIN title_authors AS ta2
 ON a.au_id = ta2.au_id
WHERE ta1.royalty_share < 1.0
 AND ta2.royalty_share = 1.0;
```

В PostgreSQL вам следует конвертировать числа с плавающей запятой в листингах 8.38–8.41 в `DECIMAL` (см. раздел «Преобразование типов данных с помощью функции `CAST()`» в главе 5). Чтобы запустить листинги 8.38–8.41 в PostgreSQL, измените числа с плавающей запятой следующим образом (листинг 8.38):

```
CAST (0.5 AS DECIMAL)
(листинг 8.39):
CAST (1.0 AS DECIMAL)
(листинг 8.40):
CAST (1.0 AS DECIMAL)
(листинг 8.41):
```

`CAST (1.0 AS DECIMAL)` (в двух местах) Некоторые СУБД позволяют одновременно тестировать несколько значений с использованием следующей структуры:

```
SELECT columns
FROM table1
WHERE (col1, col2, ..., colN) IN
 (SELECT colA, colB, ..., colN
 FROM table2);
```

Выражение `слева от IN` – это список столбцов в таблице `table1`. Запрос считает количество столбцов, которое есть в списке. СУБД сравнивает значения в соответствующих столбцах. Например, следующий запрос работает в Oracle и PostgreSQL:

```
SELECT au_id, city, state
FROM authors
WHERE (city, state) IN
 (SELECT city, state
 FROM publishers);
```

В результате выполнения запроса появляется список авторов, которые живут в городе и штате, где находится определенное издательство:

| au_id | city          | state |
|-------|---------------|-------|
| A03   | San Francisco | CA    |
| A04   | San Francisco | CA    |
| A05   | New York      | NY    |

## Сравнение всех значений запроса с помощью ключевого слова ALL

Вы можете использовать ключевое слово ALL, чтобы определить, меньше или больше значение, чем все значения в результате запроса. Перечислим важные параметры для запросов, которые используют ключевое слово ALL:

- ALL изменяет оператор сравнения в тесте запроса и сопровождает его операторами =, <>, <, <=, > или >= (см. раздел «Сравнение значений, возвращаемых подзапросом, с использованием операторов сравнения» ранее в этой главе);
- комбинация оператора сравнения и ключевого слова ALL сообщает СУБД, как применить тест сравнения ко всем значениям, которые считал запрос. Например, < ALL означает, что сравнение должно быть меньше, чем любое значение в результате запроса, > ALL – сравнение должно быть больше, чем любое значение в результате запроса;
- если ALL используется с операторами <, <=, > или >=, сравнение эквивалентно расчету максимального или минимального значения в результате запроса. < ALL говорит о том, что значение меньше, чем любое другое значение в запросе, то есть меньше минимального значения; > ALL – больше максимального значения. В табл. 8.2 перечислены эквивалентные выражения ALL и функции для работы со столбцами. В листинге 8.45 мы покажем, как заменить запрос > ALL с помощью MAX ();

Таблица 8.2. Выражения, эквивалентные ALL

| Выражение ALL    | Функция столбца         |
|------------------|-------------------------|
| < ALL (subquery) | < MIN (subquery values) |
| > ALL (subquery) | > MAX (subquery values) |

- сравнение = ALL возможно, но используется нечасто. = ALL всегда будет ложным условием, только если все значения, считанные запросом, не идентичны (и не равны значению при тестировании);
- запрос может быть простым или сложным (см. раздел «Простые и сложные запросы» ранее в этой главе);
- список столбцов команды SELECT запроса может включать только одно выражение или название столбца;
- сравниваемые значения должны относиться к одному типу данных или быть конвертируемыми (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5);
- сравнения могут быть зависимыми и независимыми, это обусловлено конкретной СУБД (см. примечание DBMS в разделе «Фильтрация строк с помощью предложения WHERE» главы 4);
- запрос должен считать одно значение (результат одной строки или столбца). Запрос, который считывает несколько значений, приводит к ошибке;
- если результат запроса не считывает строки, условие ALL становится истинным.

## Сравнение всех значений запроса

В предложении WHERE команды SELECT напечатайте:

```
WHERE test_expr op ALL (subquery)
```

*test\_expr* — это буквенное значение, название столбца, выражение или запрос, который считывает одно значение; *op* — оператор сравнения (=, <>, <, <=, > или >=); *subquery* — скалярный запрос, который считывает один столбец.

Если все значения, считанные с помощью запроса, соответствуют условию ALL или результат запроса пустой (имеет строки с нулями), условие ALL задается как истинное. Условие ALL задается как ложное, если какое-либо (хотя бы одно) значение запроса не соответствует условию ALL или если любое значение запроса равно нулю.

Предложение HAVING имеет такую же структуру: HAVING *test\_expr* op ALL (*subquery*).

Листинг 8.43 отображает список всех авторов, которые живут в городе, где нет ни одного издательства. Внутренний запрос находит все города, в которых есть издательства, а внешний запрос сравнивает город, в котором живет каждый автор, с городами, в которых есть издательства. Результат исполнения листинга см. на рис. 8.43.

Вы можете использовать NOT IN, чтобы повторить листинг 8.43:

```
SELECT au_id, au_lname, au_fname, city
FROM authors
WHERE city NOT IN
(SELECT city FROM publishers);
```

**Листинг 8.43.** Отобразить список авторов, которые живут в городе, где нет ни одного издательства. Результат показан на рис. 8.43

```

Листинг
SELECT au_id, au_lname, au_fname, city
FROM authors
WHERE city <> ALL
(SELECT city
FROM publishers);
```

| au_id | au_lname    | au_fname | city      |
|-------|-------------|----------|-----------|
| A01   | Buchman     | Sarah    | Bronx     |
| A02   | Heydemark   | Wendy    | Boulder   |
| A06   | Kellsey     |          | Palo Alto |
| A07   | O'Furniture | Paddy    | Sarasota  |

**Рис. 8.43.** Результат выполнения листинга 8.43

**Листинг 8.44.** Отобразить список книг (не биографий), цена которых меньше, чем цена биографий. Результат показан на рис. 8.44

```

Листинг
SELECT title_id, title_name
FROM titles
WHERE type <> 'biography'
AND price < ALL
(SELECT price
FROM titles
WHERE type = 'biography'
AND price IS NOT NULL);

```

| title_id | title_name                      |
|----------|---------------------------------|
| T05      | Exchange of Platitudes          |
| T08      | Just Wait Until After School    |
| T11      | Perhaps It's a Gladular Problem |

**Рис. 8.44.** Результат выполнения листинга 8.44

**Листинг 8.45.** Отобразить список книг, которые продаются лучше, чем книги, написанные автором A06 (в том числе в соавторстве). Результат см. на рис. 8.45

```

Листинг
SELECT title_id, title_name
FROM titles
WHERE sales > ALL
(SELECT sales
FROM title_authors ta
INNER JOIN titles t
ON t.title_id = ta.title_id
WHERE ta.au_id = 'A06'
AND sales IS NOT NULL);

```

| title_id | title_name                |
|----------|---------------------------|
| T05      | Exchange of Platitudes    |
| T07      | I Blame My Mother         |
| T12      | Spontaneous, Not Annoying |

**Рис. 8.45.** Результат выполнения листинга 8.45

Листинг 8.44 отображает список всех книг (не биографий), цена которых меньше, чем цена биографий. Внутренний запрос находит все цены на биографии. Внешний запрос находит самую низкую цену в списке и определяет, дешевле ли другие книги. Результат исполнения листинга см. на рис. 8.44. Условие `price IS NOT NULL` необходимо, так как цена на биографию T10 равна нулю. Без этого условия весь запрос будет выдавать NULL, поскольку невозможно определить, что цена меньше значения null (см. раздел «Значение null» в главе 3).

Листинг 8.45 отображает список книг, которые продаются лучше, чем книги, написанные автором A06 (в том числе в соавторстве). Внутренний запрос использует соединение, чтобы найти информацию о продажах всех книг, которые написал автор A06. Внешний запрос обрабатывает показатели самых высоких продаж в списке и определяет разницу в продажах. Результат исполнения листинга см. на рис. 8.45. Условие `IS NOT NULL` необходимо на случай, если цены книг автора A06 будут равны NULL. Мы можем повторить листинг 8.45 с помощью предложений `GROUP BY`, `HAVING` и `MAX()` (вместо `ALL`):

```

SELECT title_id
FROM titles
GROUP BY title_id
HAVING MAX (sales) >
(SELECT MAX (sales)
FROM title_authors ta
INNER JOIN titles t
ON t.title_id = ta.title_id
WHERE ta.au_id = 'A06');

```

Листинг 8.46 использует сложный запрос в пункте HAVING внешнего запроса, чтобы отобразить типы книг, уровень продаж которых более чем в два раза превышает средние показатели продажи книг этого типа. Внутренний запрос рассчитывается один раз для каждой группы, указанной во внешнем запросе (для каждого типа книги). Результат исполнения листинга см. на рис. 8.46.



Ключевое слово ALL является эквивалентом оператора NOT IN (см. раздел «Проверка на входжение во множество с помощью оператора IN» ранее в этой главе).



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Принципы работы с подзапросами» ранее в этой главе.

В PostgreSQL следует конвертировать числа с плавающей запятой в листинге 8.46 в тип DECIMAL (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5). Чтобы запустить листинг 8.46 в PostgreSQL, измените числа с плавающей запятой следующим образом:

```
CAST(2.0 AS DECIMAL)
```

**Листинг 8.46.** Отобразить типы книг, уровень продаж которых более чем в два раза выше среднего уровня продаж книг этого типа. Результат показан на рис. 8.46

```

Листинг
SELECT t1.type
FROM titles t1
GROUP BY t1.type
HAVING MAX(t1.sales) >= ALL
 (SELECT 2.0 * AVG(t2.sales)
 FROM titles t2
 WHERE t1.type = t2.type);

type

biography

```

**Рис. 8.46.** Результат выполнения листинга 8.46



Таблица 8.3. Выражения, эквивалентные ANY

| Выражение ANY    | Функция столбца         |
|------------------|-------------------------|
| < ANY (subquery) | < MIN (subquery values) |
| > ANY (subquery) | > MAX (subquery values) |

# Сравнение некоторых значений запроса с помощью ключевого слова ANY

Вы можете использовать слово ANY аналогично ALL (см. предыдущий раздел). Это слово определяет, равно ли, меньше или больше значение, чем любое (хотя бы одно) значение в результате запроса. Перечислим важные параметры запросов, которые используют это ключевое слово:

- ANY изменяет оператор сравнения в тесте запроса и сопровождает его операторами =, <>, <, <=, > или >= (см. раздел «Сравнение значений, возвращаемых подзапросом, с использованием операторов сравнения» ранее в этой главе);
- комбинация оператора сравнения и ANY сообщает СУБД, как применить тест сравнения ко всем значениям, которые считал запрос. Например, < ANY означает, что сравнение должно быть меньше, чем хотя бы одно значение в результате запроса; > ANY означает, что сравнение должно быть больше, чем хотя бы одно значение в результате запроса;
- если ANY используется с операторами <, <=, > или >=, сравнение будет эквивалентно расчету максимального или минимального значения в результате запроса. < ANY свидетельствует о том, что значение меньше, чем хотя бы одно значение в запросе, то есть меньше минимального значения; > ANY – больше максимального значения. В табл. 8.3 перечислены эквивалентные выражения ANY и функции для работы со столбцами. В листинге 8.49 мы покажем, как замечать запрос > ANY с помощью MIN();

- сравнение = ANY эквивалентно IN (см. раздел «Проверка на вхождение во множество с помощью оператора IN» ранее в этой главе);
- запрос может быть простым или сложным (см. раздел «Простые и сложные запросы» ранее в этой главе);
- список команды SELECT запроса может включать только одно выражение или название столбца;
- сравниваемые значения должны относиться к одному типу данных или быть конвертируемыми (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5);
- сравнения могут быть зависимыми и независимыми, что обусловлено конкретной СУБД (см. примечание **DBMS** в разделе «Фильтрация строк с помощью предложения WHERE» главы 4);
- запрос должен считать одно значение (результат одной строки или столбца). Запрос, который считывает несколько значений, приводит к ошибке;
- если результат запроса не считывает строки, условие ANY становится ложным.

### Сравнение некоторых значений запроса

В предложении WHERE команды SELECT напечатайте `WHERE test_expr op ANY (subquery)`.

Здесь *test\_expr* – буквенное значение, название столбца, выражение или запрос, который считывает одно значение; *op* – оператор сравнения (=, <>, <, <=, > или >=); *subquery* – скалярный запрос, который считывает один столбец.

Если хотя бы одно значение, считанное с помощью запроса, соответствует усло-

вию ANY, то это условие задается как истинное. Условие ANY задается как ложное, если ни одно значение запроса не соответствует условию или если запрос пуст (содержит строки с нулями) либо содержит только нули.

Предложение HAVING имеет такую же структуру:

```
HAVING test_expr op ANY (subquery)
```

Листинг 8.47 отображает список всех авторов, которые живут в городе, где расположено издательство. Внутренний запрос находит все города, в которых есть издательства, а внешний запрос сравнивает город, в котором живет каждый автор, с городами, в которых есть издательства. Результат исполнения листинга показан на рис. 8.47.

Вы можете использовать IN, чтобы повторить листинг 8.47:

```
SELECT au_id, au_lname, au_fname, city
FROM authors
WHERE city IN
(SELECT city FROM publishers);
```

Листинг 8.48 отображает список всех книг (не биографий), цена которых меньше, чем цена хотя бы одной биографии. Внутренний запрос находит все цены на биографии. Внешний запрос находит самую высокую цену в списке и определяет, дешевле ли все биографии. Результат исполнения листинга показан на рис. 8.48. В отличие от сравнения с ANY в листинге 8.44, условие `price IS NOT NULL` здесь не нужно, даже несмотря на то, что цена на биографию T10 равна NULL. СУБД не выясняет, истинны ли все сравнения по цене, а уточняет, истинно ли хотя бы одно сравнение, поэтому сравнение с NULL игнорируется.

**Листинг 8.47.** Отобразить список всех авторов, которые живут в городе, где расположено издательство. Результат см. на рис. 8.47

Листинг

```
SELECT au_id, au_lname, au_fname, city
FROM authors
WHERE city = ANY
 (SELECT city
 FROM publishers);
```

| au_id | au_lname | au_fname  | city          |
|-------|----------|-----------|---------------|
| ----- | -----    | -----     | -----         |
| A03   | Hull     | Hallie    | San Francisco |
| A04   | Hull     | Klee      | San Francisco |
| A05   | Kells    | Christian | New York      |

**Рис. 8.47.** Результат выполнения листинга 8.47

**Листинг 8.48.** Отобразить список всех книг (не биографий), цена которых меньше, чем цена хотя бы одной биографии. Результат показан на рис. 8.48

Листинг

```
SELECT title_id, title_name
FROM titles
WHERE type <> 'biography'
 AND price < ANY
 (SELECT price
 FROM titles
 WHERE type = 'biography');
```

| title_id | title_name                       |
|----------|----------------------------------|
| -----    | -----                            |
| T01      | 1977!                            |
| T02      | 200 Years of German Humor        |
| T04      | But I Did It Unconsciously       |
| T05      | Exchange of Platitudes           |
| T08      | Just Wait Until After School     |
| T09      | Kiss My Boo-Boo                  |
| T11      | Perhaps It's a Glandular Problem |

**Рис. 8.48.** Результат выполнения листинга 8.48

Листинг 8.49 отображает список всех книг, которые продаются лучше, чем хотя бы одна книга, написанная автором A06 (в том числе в соавторстве). Внутренний запрос использует соединение, чтобы найти информацию о продажах всех книг, которые написал автор A06. Внешний запрос изучает показатели самых низких продаж в списке и определяет разницу в продажах. Результат исполнения листинга представлен на рис. 8.49. В отличие от примера с ALL в листинге 8.45, условие IS NOT NULL

**Листинг 8.49.** Отобразить список всех книг, которые продаются лучше, чем хотя бы одна книга, написанная автором A06 (в том числе в соавторстве). Результат см. на рис. 8.49

Листинг

```
SELECT title_id, title_name
FROM titles
WHERE sales > ANY
 (SELECT sales
 FROM title_authors ta
 INNER JOIN titles t
 ON t.title_id = ta.title_id
 WHERE ta.au_id = 'A06');
```

| title_id | title_name                          |
|----------|-------------------------------------|
| -----    | -----                               |
| T02      | 200 Years of German Humor           |
| T03      | Ask Your System Administrator       |
| T04      | But I Did It Unconsciously          |
| T05      | Exchange of Platitudes              |
| T06      | How About Never?                    |
| T07      | I Blame My Mother                   |
| T09      | Kiss My Boo-Boo                     |
| T11      | Perhaps It's a Glandular Problem    |
| T12      | Spontaneouse, Not Annoying          |
| T13      | What Are You Civilian Applications? |

**Рис. 8.49.** Результат выполнения листинга 8.49

здесь не нужно. Мы можем повторить листинг 8.49 с помощью предложений GROUP BY, HAVING и MIN() (вместо ANY):

```
SELECT title_id
FROM titles
GROUP BY title_id
HAVING MIN (sales) >
 (SELECT MIN (sales)
 FROM title_authors ta
 INNER JOIN titles t
 ON t.title_id = ta.title_id
 WHERE ta.au_id = 'A06');
```

**П**

Ключевое слово ANY эквивалентно IN, однако  $\lt \>$  ANY не является эквивалентом NOT IN. Если запрос считал значения x, y, z, то выражение:

```
test_expr $\lt \>$ ANY (subquery)
```

является эквивалентом

```
test_expr $\lt \>$ x OR
```

```
test_expr $\lt \>$ y OR
```

```
test_expr $\lt \>$ z
```

Но выражение

```
test_expr NOT IN (subquery)
```

является эквивалентом

```
test_expr $\lt \>$ x AND
```

```
test_expr $\lt \>$ y AND
```

```
test_expr $\lt \>$ z
```

(NOT IN эквивалентно  $\lt \>$  ALL).

**П**

В SQL слова ANY и SOME являются синонимами. Во многих СУБД вы можете использовать ключевое слово SOME вместо ANY.



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Принципы работы с подзапросами» ранее в этой главе.

## Проверка наличия выборки с помощью оператора EXISTS

В этой главе мы использовали операторы IN, ALL и ANY, чтобы сравнить значения полей в условии фильтрации со значениями, полученными с помощью подзапроса. Операторы EXISTS и NOT EXISTS не сравнивают значения, а проверяют, вернул ли подзапрос хотя бы одну строку или нет. Перечислим основные параметры такой проверки:

- тест на наличие строк в результате выполнения подзапроса не сравнивает значения, поэтому не сопровождается оператором сравнения;
- запрос может быть простым или сложным, но обычно он сложный (см. раздел «Простые и сложные запросы» ранее в этой главе);
- запрос может считать любое количество столбцов и строк;
- предложение SELECT в подзапросе задается как SELECT \* для считывания всех столбцов. Нет необходимости указывать отдельные названия столбцов, так как EXISTS просто проверяет наличие строк, которые соответствуют условию запроса; сами значения в строках не рассматриваются;
- все запросы с IN, ALL и ANY могут сопровождаться EXISTS или NOT EXISTS. В некоторых примерах далее мы приведем эквивалентные выражения;
- если запрос считывает хотя бы одну строку, тест на EXISTS становится истинным, а тест на NOT EXISTS – ложным;

- если запрос не считывает строки, тест на NOT EXISTS становится истинным, а тест на EXISTS – ложным;
- строка запроса, содержащая во всех полях NULL, все равно считается строкой (тест на EXISTS становится истинным, а тест на NOT EXISTS – ложным);
- поскольку тест EXISTS не выполняет сравнений, он не имеет проблем со значениями null, как тесты, использующие операторы сравнения (IN, ALL или ANY).

### Выполнение теста на наличие выборки

В предложении WHERE команды SELECT напечатайте WHERE [NOT] EXISTS (*subquery*).

Здесь *subquery* – это запрос, который считывает любое количество столбцов и строк.

Если запрос считывает хотя бы одну строку, условие EXISTS задается как истинное. Если запрос считывает пустые строки, условие EXISTS задается как ложное. Чтобы изменить результат тестирования, укажите NOT.

Предложение HAVING имеет такую же структуру:

HAVING [NOT] EXISTS (*subquery*)

Листинг 8.50 отображает список всех издательств, которые опубликовали биографии. Этот запрос поочередно рассматривает информацию о каждом издательстве и определяет, не выполняется ли для него тест на наличие результата подзапроса. Первое издательство – P01 (Abatis Publishers). СУБД определяет, существуют ли в таблице titles строки, в которых

pub\_id — это P01, а type — biography. Если такая строка есть, то Abatis Publishers включается в конечный результат. СУБД повторяет эту процедуру для всех издательств. Результат исполнения листинга см. на рис. 8.50. Если мы желаем отобразить информацию об издательствах, которые не публиковали биографии, следует заменить EXISTS на NOT EXISTS. См. листинг 8.33 ранее в этой главе (эквивалентный запрос, который использует IN).

Листинг 8.51 отображает список авторов, которые не написали ни одной книги (в том числе в соавторстве). Результат исполнения листинга см. на рис. 8.51. См. листинг 8.35 ранее в этой главе (эквивалентный запрос, который использует NOT IN).

Листинг 8.52 отображает список авторов, которые живут в городе, где расположено издательство. Результат исполнения листинга представлен на рис. 8.52. См. листинг 8.47 ранее в этой главе (эквивалентный запрос, который использует + ANY).

Вспомните из раздела «Поиск общих строк с помощью команды INTERSECT» в главе 7 о том, что вы можете использовать команду INTERSECT, чтобы считать общие строки для двух таблиц. Также для этого вы можете использовать оператор EXISTS. Листинг 8.53 содержит список городов, в которых живут авторы и расположены издательства. Результат исполнения листинга показан на рис. 8.53. См. листинг 7.46 в главе 7 (эквивалентный запрос, который использует команду INTERSECT).

**Листинг 8.50.** Отобразить список всех издательств, которые опубликовали биографии. Результат показан на рис. 8.50

```

Листинг
SELECT pub_name
 FROM publishers p
 WHERE EXISTS
 (SELECT *
 FROM titles t
 WHERE t.pub_id = p.pub_id
 AND type = 'biography');
```

```

pub_name

Abatis Publishers
Achadenfreude Press
```

**Рис. 8.50.** Результат выполнения листинга 8.50

**Листинг 8.51.** Отобразить список авторов, которые не написали ни одной книги (в том числе в соавторстве). Результат представлен на рис. 8.51

```

Листинг
SELECT au_id, au_fname, au_lname
 FROM authors a
 WHERE NOT EXISTS
 (SELECT *
 FROM title_authors ta
 WHERE ta.au_id = a.au_id);
```

```

au_id au_fname au_lname

A07 Paddy O'Furniture
```

**Рис. 8.51.** Результат выполнения листинга 8.51

**Листинг 8.52.** Отобразить список авторов, которые живут в городе, где расположено издательство. Результат см. на рис. 8.52

```

Листинг

SELECT au_id, au_lname, au_fname, city
 FROM authors a
 WHERE EXISTS
 (SELECT *
 FROM publishers p
 WHERE p.city = a.city);

```

| au_id | au_lname | au_fname  | city          |
|-------|----------|-----------|---------------|
| A03   | Hull     | Hallie    | San Francisco |
| A04   | Hull     | Klee      | San Francisco |
| A05   | Kells    | Christian | New York      |

**Рис. 8.52.** Результат выполнения листинга 8.52

**Листинг 8.53.** Отобразить список городов, в которых живут авторы и расположены издательства. Результат представлен на рис. 8.53

```

Листинг

SELECT DISTINCT city
 FROM authors a
 WHERE EXISTS
 (SELECT *
 FROM publishers p
 WHERE p.city = a.city);

```

| city          |
|---------------|
| New York      |
| San Francisco |

**Рис. 8.53.** Результат выполнения листинга 8.53

Вы также можете повторить этот запрос с помощью внутреннего объединения:

```

SELECT DISTINCT a.city
 FROM authors a
 INNER JOIN publishers p
 ON a.city = p.city;

```

Вспомните из раздела «Поиск разных строк с помощью команды EXCEPT» в главе 7 о том, что вы можете использовать команду EXCEPT, чтобы считать строки из одной таблицы, которых нет в другой таблице. Также для этого вы можете использовать оператор NOT EXISTS. Листинг 8.54 отображает список городов, в которых живут авторы, но нет издательств. Результат исполнения листинга см. на рис. 8.54. См. листинг 7.47 в главе 7 (эквивалентный запрос, который использует EXCEPT).

**Листинг 8.54.** Отобразить список городов, в которых живут авторы, но нет издательств. Результат см. на рис. 8.54

```

Листинг

SELECT DISTINCT city
 FROM authors a
 WHERE NOT EXISTS
 (SELECT *
 FROM publishers p
 WHERE p.city = a.city);

```

| city      |
|-----------|
| Boulder   |
| Bronx     |
| Palo Alto |
| Sarasota  |

**Рис. 8.54.** Результат выполнения листинга 8.54

Вы также можете повторить этот запрос с помощью оператора NOT IN:

```
SELECT DISTINCT city
FROM authors
WHERE city NOT IN
 (SELECT city
 FROM publishers);
```

Или с помощью внешнего объединения:

```
SELECT DISTINCT a.city
FROM authors a
LEFT OUTER JOIN publishers p
ON a.city = p.city
WHERE p.city IS NULL;
```

Листинг 8.55 отображает список авторов, которые написали (в том числе в соавторстве) три или более книги. Результат исполнения листинга представлен на рис. 8.55.

Листинг 8.56 содержит два условия на наличие выборки, чтобы отобразить список авторов, которые написали (в том числе в соавторстве) не только книги для детей, но и книги по психологии. Результат исполнения листинга представлен на рис. 8.56.

Листинг 8.57 выполняет тест на уникальность, чтобы определить, есть ли повторы в столбце au\_id таблицы authors. Если в столбце есть повторяющиеся строки, запрос отобразит Yes; в противном случае результат будет пустым, то есть ни одна строка возвращена не будет. Результат исполнения листинга показан на рис. 8.57. au\_id является ключевым столбцом в таблице authors, поэтому не содержит повторов.

Листинг 8.58 выполняет аналогичный тест для таблицы title\_authors, которая действительно содержит повторяющиеся значения au\_id. Результат исполнения листинга показан на рис. 8.58. Вы можете добавить к предложению GROUP BY сгруппированные столбцы, чтобы определить, повторяются ли они.

**Листинг 8.55.** Отобразить список авторов, которые написали (в том числе в соавторстве) три или более книги. Результат см. на рис. 8.55

```
Листинг
SELECT au_id, au_fname, au_lname
FROM authors a
WHERE EXISTS
 (SELECT *
 FROM title_authors ta
 WHERE ta.au_id = a.au_id
 HAVING COUNT(*) >= 3);
```

| au_id | au_fname | au_lname  |
|-------|----------|-----------|
| A01   | Sarah    | Buchman   |
| A02   | Wendy    | Heydemark |
| A04   | Klee     | Hull      |
| A06   |          | Kellsey   |

**Рис. 8.55.** Результат выполнения листинга 8.55

**Листинг 8.56.** Отобразить список авторов, которые написали (в том числе в соавторстве) как книги для детей, так и книги по психологии. Результат см. на рис. 8.56

```
Листинг
SELECT au_id, au_fname, au_lname
FROM authors a
WHERE EXISTS
 (SELECT *
 FROM title_authors ta
 INNER JOIN titles t
 ON t.title_id = ta.title_id
 WHERE ta.au_id = a.au_id
 AND t.type = 'children')
AND EXISTS
 (SELECT *
 FROM title_authors ta
 INNER JOIN titles t
 ON t.title_id = ta.title_id
 WHERE ta.au_id = a.au_id
 AND t.type = 'psychology');
```

| au_id | au_fname | au_lname |
|-------|----------|----------|
| A06   |          | Kellsey  |

**Рис. 8.56.** Результат выполнения листинга 8.56



**Листинг 8.57.** Проверить, есть ли повторяющиеся значения в столбце `au_id` таблицы `authors`.  
Результат см. на рис. 8.57

```

Листинг
SELECT DISTINCT 'Yes' AS "Duplicates?"
WHERE EXISTS
 (SELECT *
 FROM authors
 GROUP BY au_id
 HAVING COUNT(*) > 1);

```

Duplicates?  
-----

**Рис. 8.57.** Результат выполнения листинга 8.57

**Листинг 8.58.** Проверить, есть ли повторяющиеся значения в столбце `au_id` таблицы `title_authors`.  
Результат см. на рис. 8.58

```

Листинг
SELECT DISTINCT 'Yes' AS "Duplicates?"
WHERE EXISTS
 (SELECT *
 FROM title_authors
 GROUP BY au_id
 HAVING COUNT(*) > 1);

```

Duplicates?  
-----

Yes

**Рис. 8.58.** Результат выполнения листинга 8.58

**П**

Вы также можете использовать функцию `COUNT(*)`, чтобы определить, считывает ли запрос хотя бы одну строку. `COUNT(*)`, как правило, не так эффективна, как `EXISTS`. СУБД перестает обрабатывать запрос `EXISTS` тогда, когда определит, что он считывает хотя бы одну строку. В то же время `COUNT(*)` заставляет СУБД обработать весь запрос. Он эквивалентен листингу 8.52, но работает медленнее:

```

SELECT au_id, au_lname, au_fname, city
FROM authors a
WHERE
 (SELECT COUNT(*)
 FROM publishers p
 WHERE p.city = a.city) > 0;

```

**П**

Хотя `SELECT *` является самой распространенной формой предложения `SELECT` в запросе `EXISTS`, вы можете использовать запрос `SELECT column` или `SELECT constant_value`, чтобы ускорить выполнение запроса (это следует делать в том случае, если СУБД не может самостоятельно определить, что ей не нужно создавать всю таблицу для запроса `EXISTS`). За дополнительной информацией обращайтесь к разделу «Сравнение эквивалентных запросов» далее в этой главе.

**П**

Хотя мы используем `SELECT COUNT(*)` в некоторых специальных запросах (см. примечание **DBMS** к данному разделу), вы должны быть внимательными при использовании функции в предложении `SELECT` запроса. Например, тест на наличие (см. листинг 8.59) будет истинным, так как `COUNT(*)` всегда будет считывать строку (в данном случае с нулем). Результат листинга (см. рис. 8.59) является ошибочным, так как не существует издательства с индексом XXX.



MySQL 4.0 и более ранних версий не поддерживает подзапросы. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Принципы работы с подзапросами» ранее в этой главе.

Чтобы запустить листинги 8.57 и 8.58 в Oracle, добавьте предложение `FROM DUAL` к внешнему запросу (см. примечание **DBMS** к разделу «Создание производных столбцов» в главе 5).

Чтобы запустить листинги 8.55, 8.57 и 8.58 в Microsoft Access, напечатайте (листинг 8.55):

```
SELECT au_id, au_fname, au_lname
FROM authors a
WHERE EXISTS
 (SELECT COUNT(*)
 FROM title_authors ta
 WHERE ta.au_id = a.au_id
 HAVING COUNT(*) >= 3);
```

(листинг 8.57):

```
SELECT DISTINCT 'YES' AS
 "Duplicates?"
FROM authors
WHERE EXISTS
 (SELECT COUNT(*)
 FROM authors
 GROUP BY au_id
 HAVING COUNT(*) >= 1);
```

(листинг 8.58):

```
SELECT DISTINCT 'YES' AS
 "Duplicates?"
FROM title_authors
WHERE EXISTS
 (SELECT COUNT(*)
 FROM title_authors
 GROUP BY au_id
 HAVING COUNT(*) > 1);
```

Чтобы запустить листинги 8.55, 8.57 и 8.58 в PostgreSQL, напечатайте (листинг 8.55):

```
SELECT au_id, au_fname, au_lname
FROM authors a
WHERE EXISTS
 (SELECT COUNT(*)
 FROM title_authors ta
 WHERE ta.au_id = a.au_id
 HAVING COUNT(*) >= 3);
```

(листинг 8.57):

```
SELECT 'YES' AS "Duplicates?"
WHERE EXISTS
 (SELECT COUNT(*)
 FROM authors
 GROUP BY au_id
 HAVING COUNT(*) > 1);
```

(листинг 8.58):

```
SELECT 'YES' AS "Duplicates?"
WHERE EXISTS
 (SELECT COUNT(*)
 FROM title_authors
 GROUP BY au_id
 HAVING COUNT(*) > 1);
```

**Листинг 8.59.** Вы должны быть внимательными при использовании функций в предложении `SELECT` запроса. Результат см. на рис. 8.59

```

Листинг
SELECT pub_id
FROM publishers
WHERE EXISTS
 (SELECT COUNT(*)
 FROM titles
 WHERE pub_id = 'XXX');
```

```
pub_id

P01
P02
P03
P04
```

**Рис. 8.59.** Результат выполнения листинга 8.59

**Листинг 8.60.** Запросы отображают список авторов, которые написали (в том числе в соавторстве) хотя бы одну книгу. Результат показан на рис. 8.60

```

Листинг

SELECT DISTINCT a.au_id
 FROM authors a
 INNER JOIN title_authors ta
 ON a.au_id = ta.au_id;

SELECT DISTINCT a.au_id
 FROM authors a, title_authors ta
 WHERE a.au_id = ta.au_id;

SELECT au_id
 FROM authors a
 WHERE au_id IN
 (SELECT au_id
 FROM title_authors);

SELECT au_id
 FROM authors a
 WHERE au_id = ANY
 (SELECT au_id
 FROM title_authors);

SELECT au_id
 FROM authors a
 WHERE EXISTS
 (SELECT *
 FROM title_authors ta
 WHERE a.au_id = ta.au_id);

SELECT au_id
 FROM authors a
 WHERE 0 <
 (SELECT COUNT(*)
 FROM title_authors ta
 WHERE a.au_id = ta.au_id);

```

```

au_id

A01
A02
A03
A04
A05
A06

```

**Рис. 8.60.** Результат выполнения команд листинга 8.60

## Сравнение эквивалентных запросов

Как вы уже знаете, один и тот же запрос можно отобразить разными способами (разная структура, одинаковая семантика). Чтобы разъяснить эту особенность, мы написали один и тот же запрос шестью семантически одинаковыми способами. Каждая из команд в листинге 8.60 отображает список авторов, которые написали (в том числе в соавторстве) хотя бы одну книгу. Результат исполнения листинга показан на рис. 8.60.

Первые два запроса (внутреннее объединение) будут работать с одинаковой скоростью. Третий запрос (использует подзапросы), пожалуй, самый медленный из всех. СУБД может (и, вероятно, так и сделает) остановить обработку других запросов, если найдет одно соответствующее значение. Но запрос в последней команде должен пересчитать все подходящие строки до того, как определить соответствие истинному или ложному условию. Внутренние объединения должны работать примерно с той же скоростью, что и самая быстрая команда, которая использует запросы.

Вам может понравиться эта гибкость при написании программы, но ее не любят специалисты, которые создают программы для оптимизации работы СУБД. Причина этого в том, что им приходится рассчитывать все возможные способы для выражения запроса, определять, какой из них работает быстрее, а также переформулировать ваш запрос оптимальным образом (а на это уходит очень много времени). Если ваша СУБД располагает хорошей программой оптимизации, она будет обрабатывать все шесть запросов листинга 8.60 с одинаковой скоростью. Но на это не стоит рассчитывать, поэтому вам придется поэкспериментировать с СУБД, чтобы увидеть, какая версия функционирует быстрее всех.

# ДОБАВЛЕНИЕ, ОБНОВЛЕНИЕ И УДАЛЕНИЕ СТРОК

---

## 9

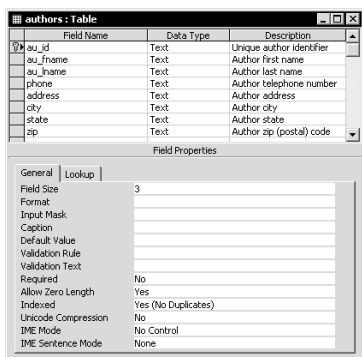
Вы узнали о том, как использовать команду `SELECT`, чтобы считывать и анализировать данные в таблицах. В настоящей главе мы объясним, как использовать команды `SQL`, чтобы изменять данные в таблице:

- команда `INSERT` добавляет в таблицу новые строки;
- команда `UPDATE` изменяет значения в существующих строках таблицы;
- команда `DELETE` удаляет строки из таблицы.

Перечисленные команды не считывают результат, но ваша СУБД выдаст сообщение о том, была ли выполнена команда или нет (и, соответственно, была ли изменена таблица или нет). Чтобы увидеть эффект, который команда произвела на таблицу, введите `SELECT`, например `SELECT * FROM table`.

В отличие от `SELECT`, которая только получает доступ к данным, указанные команды изменяют данные, поэтому администратор базы данных может не разрешить вам их использовать.

---



**Рис. 9.1.** Отображение названий столбцов в Microsoft Access

## Отображение названий столбцов в таблице

Чтобы использовать команды `INSERT`, `UPDATE` или `DELETE`, необходимо знать названия столбцов той таблицы, данные в которой вы изменяете, а именно:

- порядок столбцов в таблице;
- название каждого столбца;
- тип данных в каждом столбце;
- является ли столбец ключевым;
- должны ли значения в столбце быть уникальными;
- разрешено ли вводить в столбце значения `null`;
- значения по умолчанию для столбца (если есть).

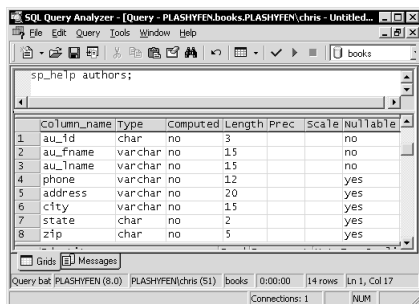
В табл. 2.2–2.6 главы 2 мы привели обозначения столбцов для таблиц в базе данных. Вы можете получить эту информацию с помощью инструментов СУБД, которые работают с объектами баз данных. В этом разделе мы расскажем, как использовать такие инструменты, чтобы отобразить названия столбцов.

### Порядок отображения названий столбцов таблицы в Microsoft Access

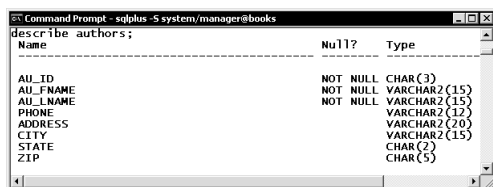
1. Нажмите клавишу **F11**, чтобы открыть окно **Database** (База данных).
2. Щелкните по кнопке **Tables** (Таблицы), которая находится под кнопкой **Objects** (Объекты).
3. Щелкните по таблице в списке.
4. Нажмите на кнопку **Design** (Конструктор) на панели инструментов, чтобы открыть таблицу в режиме редактирования – **Design View** (см. рис. 9.1).

## Порядок отображения названий столбцов таблицы в Microsoft SQL Server

1. Запустите SQL Query Analyzer или консольный интерпретатор `osql` (см. раздел «Microsoft SQL Server» в главе 1). Команда выдает очень много информации, поэтому мы рекомендуем выбрать пункты меню **Query** ⇒ **Results in Grid** (Запрос ⇒ Результат в виде таблицы) в SQL Query Analyzer.
2. Введите команду `sp_help table;` (*table* – это название таблицы).
3. В SQL Query Analyzer выберите пункты меню **Query** ⇒ **Execute** (Запрос ⇒ Выполнить) либо нажмите клавишу **F5** (см. рис. 9.2) или в строке `osql` нажмите клавишу **Enter**, введите `go` и нажмите на **Enter**.



**Рис. 9.2.** Отображение названий столбцов таблицы в Microsoft SQL Server



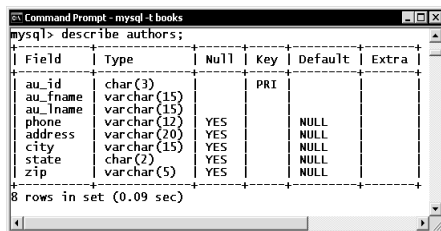
**Рис. 9.3.** Отображение названий столбцов таблицы в Oracle

## Порядок отображения названий столбцов таблицы в Oracle

1. Запустите программу SQL\*Plus или утилиту командной строки `sqlplus` (см. раздел «Oracle» в главе 1).
2. Введите `describe table;`, нажмите клавишу **Enter** (см. рис. 9.3). *table* – это название таблицы.

## Порядок отображения названий столбцов таблицы в MySQL

1. Запустите монитор `mysql` (см. раздел «MySQL» в главе 1).
2. Введите `describe table;`, нажмите клавишу **Enter** (см. рис. 9.4). *table* – это название таблицы.



**Рис. 9.4.** Отображение названий столбцов таблицы в MySQL

| Attribute           | Table                 | Type | Modifier |
|---------------------|-----------------------|------|----------|
| au_id               | character(3)          |      | not null |
| au_fname            | character varying(15) |      | not null |
| au_lname            | character varying(15) |      | not null |
| phone               | character varying(12) |      |          |
| address             | character varying(20) |      |          |
| city                | character varying(15) |      |          |
| state               | character(2)          |      |          |
| zip                 | character(5)          |      |          |
| Index: authors_pkey |                       |      |          |

**Рис. 9.5.** Отображение названий столбцов таблицы в PostgreSQL

## Порядок отображения названий столбцов таблицы в PostgreSQL

1. Запустите монитор `psql` (см. раздел «PostgreSQL» в главе 1).
2. Введите `\d table`, нажмите клавишу **Enter** (см. рис. 9.5). `table` – это название таблицы. Обратите внимание, что после команды вы не ставите точку с запятой.

**С**

Если вы желаете отобразить названия столбцов таблицы и сохранить порядок, в котором они появляются, но при этом не показывать другие данные в таблице, введите:

```
SELECT * FROM table WHERE 1 = 2;
```

`table` – это название таблицы, а `1 = 2` отображает какое-либо условие, которое всегда будет ложным.

**С**

За общей информацией о столбцах обращайтесь к разделу «Таблицы, столбцы и строки» в главе 2. За дополнительной информацией о ключах обращайтесь к разделам «Первичные ключи» и «Внешние ключи» в главе 2. За информацией о типах данных обращайтесь к разделу «Типы данных» в главе 3.

**П**

В главе 10 рассказывается о том, как создавать и удалять столбцы таблиц, а также изменять их названия.

## Вставка строк с помощью команды INSERT

Команда INSERT добавляет новые строки в таблицу. В этом разделе мы расскажем, как использовать эту команду, чтобы:

- добавить строку с помощью положения столбцов в таблице (INSERT VALUES);
- добавить строку с помощью названий столбцов (INSERT VALUES);
- добавить строки из одной таблицы в другую (INSERT SELECT).

Перечислим важные параметры команды INSERT:

- при добавлении строк с помощью положения в таблице вы добавляете значения в новую строку в той же последовательности, в которой они появляются в таблице (см. раздел «Добавление строки с помощью положения столбца» далее в этом разделе). При вставке строки в столбец вы задаете название столбца, в который добавляете значения для новой строки (см. подраздел «Добавление строки с помощью названий столбцов» далее в этом разделе).

Следует всегда добавлять строки с помощью названий столбцов. При этом ваш запрос будет работать и в том случае, если кто-то изменит порядок столбцов в таблице или добавит новые столбцы;

- при помощи команды INSERT VALUES вы указываете точные значения, которые должны быть вставлены в таблицу. При помощи команды INSERT SELECT вы выбираете строки из другой таблицы, которые желаете поместить в текущую;

- INSERT VALUES добавляет в таблицу одну строку, а INSERT SELECT – любое количество строк;
- каждое добавленное значение должно быть того же типа (или иметь возможность для конвертации), что и другие данные в столбце (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5);
- чтобы сохранить систему ссылок, вставленный внешний ключ должен содержать либо NULL, либо значение существующего ключа из первичной или уникальной ссылки вторичного ключа (см. разделы «Первичные ключи» и «Внешние ключи» в главе 2);
- добавленное значение не может отменить ограничения. См. раздел «Проверка значений столбца с помощью ограничения CHECK» в главе 10;
- ни одно выражение не должно приводить к арифметической ошибке (например, переполнению или делению на нуль);
- вспомните из раздела «Таблицы, столбцы и строки» в главе 2, что порядок строк в таблице не имеет значения и что вы не можете управлять расположением строк, поэтому новые строки могут появиться в любом месте таблицы.

### Добавление строки с помощью положения столбца

Введите:

```
INSERT INTO table
VALUES(value1, value2, ..., valueN);
```

*table* – это название таблицы, в которую вы добавляете строку, *value1*, *value2*, ..., *valueN* – это список буквенных обозначений или выражений, который задает значения для всех столбцов в новой строке.



**Листинг 9.1.** Команда INSERT добавит в таблицу authors новую строку, вставив значения в том порядке, в котором идут столбцы в списке. Результат исполнения листинга показан на рис. 9.6

```

Листинг
INSERT INTO authors
VALUES(
 'A08',
 'Michael',
 'Polk',
 '512-953-1231',
 '4028 Guadalupe St',
 'Austin',
 'TX',
 '78701');

```

**Листинг 9.2.** Команда INSERT добавит в таблицу authors новую строку, вставив значения в том порядке, в котором идут столбцы в списке. Результат исполнения листинга показан на рис. 9.6

```

Листинг
INSERT INTO authors(
 au_id,
 au_fname,
 au_lname,
 phone,
 address,
 city,
 state,
 zip)
VALUES(
 'A09',
 'Irene',
 'Bell',
 '415-225-4689',
 '810 Throckmorton Ave',
 'Mill Valley',
 'CA',
 '94941');

```

Количество значений должно соответствовать количеству столбцов в *table*, а значения должны быть указаны в той же последовательности, что и столбцы. СУБД вставляет каждое значение в столбец, который совпадает с положением значения в *table*, *value1* добавляется в первый столбец новой строки, *value2* – во второй столбец и т.д.

Команда-пример добавит в таблицу одну строку (см. листинг 9.1).

### Добавление строки с помощью названий столбцов

Введите:

```

INSERT INTO table
(column1, column2, ..., columnN)
VALUES(value1, value2, ..., valueN);

```

*table* – это название таблицы, в которую вы добавляете строку; *column1, column2, ..., columnN* – список названий столбцов в *table*; *value1, value2, ..., valueN* – список буквенных обозначений или выражений, которые задают значения для указанных столбцов в новой строке.

Количество значений должно соответствовать количеству столбцов в списке, а значения должны быть указаны в той же последовательности, что и названия столбцов. СУБД вставляет каждое значение в столбец, используя соответствующие значения в списке. Значение *value1* добавляется в столбец *column1* новой строки, значение *value2* – в столбец *column2* и т.д. Пропущенному столбцу присваивается значение по умолчанию или NULL.

Команда-пример добавит в таблицу одну строку. Проще всего указывать названия столбцов в том же порядке, в котором они приведены в таблице (см. листинг 9.2), но вы можете перечислить их в произвольном

порядке (см. листинг 9.3). В любом случае значения в предложении `VALUES` должны соответствовать последовательности, в которой вы указали названия столбцов.

Если вы желаете указать значения только для определенных столбцов, можете пропустить некоторые столбцы (см. листинг 9.4). Если вы пропустите столбец, СУБД должна сама указать для него значение на основании названия столбца. СУБД добавит значение по умолчанию (если оно было задано) или `NULL` (если возможно). Если вы пропустите столбец, который не имеет значения по умолчанию и в который нельзя добавить `NULL`, СУБД выдаст сообщение об ошибке и не добавит новую строку. За информацией о том, как задать значение по умолчанию и разрешить значение `null` для столбца, обращайтесь к разделам «Присвоение значения по умолчанию с помощью ограничения `DEFAULT`» и «Запрет значения `null` с помощью ограничения `NOT NULL`» в главе 10. На рис. 9.6 показаны новые строки в таблице `authors` после исполнения листингов 9.1–9.4.

## Добавление строк из одной таблицы в другую

Введите:

```
INSERT INTO table
[(column1, column2, ..., columnN)]
select_statement;
```

*table* – это название таблицы, в которую вы добавляете строку; *column1, column2, ..., columnN* – список названий столбцов в ней; *select\_statement* – любая команда `SELECT`, считывающая строки данных, которые должны быть добавлены в таблицу.

**Листинг 9.3.** Вам необязательно указывать названия столбцов в том же порядке, в котором они идут в таблице. В этом примере мы изменили порядок названий столбцов и соответствующих значений. Результат исполнения листинга см. на рис. 9.6

```
Листинг
INSERT INTO authors(
 zip,
 phone,
 address,
 au_lname,
 au_fname,
 state,
 au_id,
 city)
VALUES(
 '60614',
 '312-998-0020',
 '1937 N. Clark St',
 'Weston',
 'Dianne',
 'IL',
 'A10',
 'Chicago');
```

**Листинг 9.4.** Мы добавили строку для нового автора, но пропустили столбцы и значения для адреса автора. СУБД автоматически добавила в пропущенные столбцы `NULL`. Результат исполнения листинга показан на рис. 9.6

```
Листинг
INSERT INTO authors(
 au_id,
 au_fname,
 au_lname,
 phone)
VALUES(
 'A11',
 'Max',
 'Allard',
 '212-502-0955');
```

| au_id | au_fname  | au_lname    | phone        | address              | city          | state | zip   |
|-------|-----------|-------------|--------------|----------------------|---------------|-------|-------|
| ----- | -----     | -----       | -----        | -----                | -----         | ----- | ----- |
| A01   | Sarah     | Buchman     | 718-496-7223 | 75 West 205 St       | Bronx         | NY    | 10468 |
| A02   | Wendy     | Heydemark   | 303-986-7020 | 2922 Baseline Rd     | Boulder       | CO    | 80303 |
| A03   | Hallie    | Hull        | 415-549-4278 | 3800 Waldo Ave, #14F | San Francisco | CA    | 94123 |
| A04   | Klee      | Hull        | 415-549-4278 | 3800 Waldo Ave, #14F | San Francisco | CA    | 94123 |
| A05   | Christian | Kells       | 212-771-4680 | 114 Horatio St       | New York      | NY    | 10014 |
| A06   |           | Kellsey     | 650-836-7128 | 390 Serra Mall       | Palo Alto     | CA    | 94305 |
| A07   | Paddy     | O'Furniture | 941-925-0752 | 1442 Main St         | Sarasota      | FL    | 34236 |
| A08   | Michael   | Polk        | 512-953-1231 | 4028 Guadalupe St    | Austin        | TX    | 78701 |
| A09   | Irene     | Bell        | 415-225-4689 | 810 Throckmorton Ave | Mill Valley   | CA    | 94941 |
| A10   | Dianne    | Weston      | 312-998-0020 | 1937 N. Clark St     | Chicago       | IL    | 60614 |
| A11   | Max       | Allard      | 212-502-0955 | NULL                 | NULL          | NULL  | NULL  |

**Рис. 9.6.** После исполнения листингов 9.1–9.4 в таблицу `authors` были добавлены четыре новые строки

| pub_id | pub_name             | city        | state | country       |
|--------|----------------------|-------------|-------|---------------|
| -----  | -----                | -----       | ----- | -----         |
| P05    | This is Pizza? Press | New York    | NY    | USA           |
| P06    | This is Beer? Press  | Toronto     | ON    | Canada        |
| P07    | This is Irany? Press | London      | NULL  | United Kindom |
| P08    | This is Fame? Press  | Los Angeles | CA    | USA           |

**Рис. 9.7.** Таблица `new_publishers` используется в листингах 9.5–9.7. Она имеет такую же структуру, как таблица `publishers`

Количество столбцов в результате исполнения команды `select_statement` должно соответствовать количеству столбцов в `table` или списке столбцов. СУБД игнорирует названия столбцов при выполнении команды `select_statement` и вместо них использует положение в столбце. Для первого столбца в `table` или `column1` используется первый столбец `select_statement` в процессе выполнения и т.д. Пропущенному столбцу задается значение по умолчанию или `NULL`.

Команда-пример добавит в `table` несколько строк.

В остальных примерах этого раздела используется таблица `new_publishers` (см. рис. 9.7), которую мы создали, чтобы показать, как работает команда `INSERT SELECT`. Таблица `new_publishers` имеет ту же структуру, что и `publishers`, и используется только в качестве источника новых строк, но не изменяется в результате исполнения команды `INSERT`.

Листинг 9.5 добавляет строки с информацией об издательствах Лос-Анджелеса из таблицы `new_publishers` в `publishers`. Мы пропустили список столбцов, поэтому СУБД будет использовать положения столбцов в таблице `publishers` для добавления значений. Команда листинга 9.5 добавит в `publishers` одну строку; результат исполнения листинга показан на рис. 9.8.

Листинг 9.6 добавляет строки с информацией о неамериканских издательствах из таблицы `new_publishers` в `publishers`. Названия столбцов в пунктах `INSERT` и `SELECT` здесь одинаковые, однако это необязательно, так как СУБД игнорирует названия столбцов, считанные командой `SELECT`, и использует их положения. Эта команда добавит в `publishers` две строки; результат исполнения листинга показан на рис. 9.8.

Предложение `SELECT` может считать пустой результат (нуль строк). Листинг 9.7 добавляет строки с информацией об издательствах XXX из таблицы `new_publishers` в `publishers`. Мы можем использовать `SELECT*` вместо списка названий столбцов, так как `new_publishers` и `publishers` имеют одинаковую структуру. Команда из листинга 9.7 не добавит в `publishers` ни одной строки, поскольку издательства с названием XXX в таблице `new_publishers` нет.

**Листинг 9.5.** Добавить строки с информацией об издательствах Лос-Анджелеса из таблицы `new_publishers` в `publishers`. Результат исполнения листинга см. на рис. 9.8

```

Листинг
INSERT INTO publishers
 SELECT
 pub_id,
 pub_name,
 city,
 state,
 country
 FROM new_publishers
 WHERE city = 'Los Angeles';

```

**Листинг 9.6.** Добавить строки с информацией о неамериканских издательствах из таблицы `new_publishers` в `publishers`. Результат исполнения листинга см. на рис. 9.8

```

Листинг
INSERT INTO publishers(
 pub_id,
 pub_name,
 city,
 state,
 country)
 SELECT
 pub_id,
 pub_name,
 city,
 state,
 country
 FROM new_publishers
 WHERE country <> 'USA';

```

| pub_id | pub_name             | city          | state | country       |
|--------|----------------------|---------------|-------|---------------|
| -----  | -----                | -----         | ----- | -----         |
| P01    | Abatis Publishers    | New York      | NY    | USA           |
| P02    | Core Dump Books      | San Francisco | CA    | USA           |
| P03    | Schadenfreude Press  | Hamburg       | NULL  | Germany       |
| P04    | Tenterhooks Press    | Berkeley      | CA    | USA           |
| P06    | This is Beer? Press  | Toronto       | ON    | Canada        |
| P07    | This is Irany? Press | London        | NULL  | United Kindom |
| P08    | This is Fame? Press  | Los Angeles   | CA    | USA           |

**Рис. 9.8.** После исполнения листингов 9.5–9.7 в таблицу `publishers` были добавлены три новые строки

**Листинг 9.7.** Добавить строки с информацией об издательствах XXX из таблицы `new_publishers` в `publishers`. Результат исполнения листинга см. на рис. 9.8

```

Листинг
INSERT INTO publishers(
 pub_id,
 pub_name,
 city,
 state,
 country)
SELECT *
FROM new_publishers
WHERE pub_name = 'XXX';

```



В SQL ключевое слово `INTO` является опцией для команды `INSERT`. Microsoft Access, Oracle и PostgreSQL обязательно требуют ввода `INTO`, поэтому мы никогда его не пропускаем. Посмотрите в документации, как ваша СУБД обрабатывает значения при вставке в столбцы, данные в которых автоматически создают новый первичный ключ для строки (см. примечание **DBMS** в разделе «Первичные ключи» главы 2). СУБД может не разрешить добавление значений в эти столбцы.

На рис. 9.8 показана таблица `publishers` после исполнения листингов 9.5–9.7.



Процесс добавления строк в таблицу в первый раз называется *заполнением таблицы*.



Вы можете использовать команду `SELECT INTO`, чтобы создать новую таблицу и заполнить ее строками, которые вернет команда `SELECT` (см. раздел «Создание новой таблицы на основе существующей с помощью команды `SELECT INTO`» в главе 10).



Если вы желаете быть особенно внимательными при добавлении строк, можете проверить команду `INSERT` с помощью временной таблицы (см. разделы «Создание временной таблицы с помощью команды `CREATE TEMPORARY TABLE`» и «Создание новой таблицы на основе существующей с помощью команды `SELECT INTO`» в главе 10).



Вы также можете добавлять строки с помощью представлений (см. раздел «Изменение данных через представления» в главе 12).



Если вы применяете транзакции, воспользуйтесь командой `COMMIT` после команды `INSERT`, чтобы сделать изменения постоянными. За информацией о транзакциях обращайтесь к главе 13.



Если `table1` и `table2` имеют одинаковую структуру, можете вставить все строки из `table2` в `table1`. Для этого введите:

```

INSERT INTO table1
SELECT * FROM table2;

```

## Изменение строк с помощью команды UPDATE

Команда UPDATE изменяет значения в существующих строках таблицы. Вы можете использовать эту команду, чтобы изменять:

- все строки в таблице;
- отдельные строки в таблице.

Чтобы обновить строки, нужно указать:

- какую таблицу изменять;
- названия столбцов, которые нужно изменить, а также новые значения;
- условие поиска с целью нахождения строк для обновления (опционально).

Перечислим важные параметры команды UPDATE:

- использует предложение WHERE, в котором указывается, какие строки нужно изменить. Без предложения WHERE команда UPDATE изменит все строки в таблице;
- может быть небезопасна, потому что вы можете случайно пропустить предложение WHERE (и изменить все строки) или неправильно указать условие поиска для WHERE (и изменить не те строки). Перед запуском команды UPDATE мы рекомендуем запустить команду SELECT с тем же предложением WHERE, что и для обновления строк. Команда SELECT отобразит все строки, которые будут изменены СУБД при запуске команды UPDATE. Чтобы отобразить только количество этих строк, пользуйтесь командой SELECT COUNT (\*);
- каждое измененное значение должно быть того же типа (или иметь возможность для конвертации), что и другие данные в столбце (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5);

- чтобы сохранить ссылочную целостность, СУБД позволяет указать действие, которое будет выполняться автоматически с помощью команды UPDATE при изменении значения, на которое указывает вторичный ключ (см. советы к разделу «Задание внешнего ключа с помощью ограничения FOREIGN KEY» в главе 10);
- измененное значение не может отменить ограничение, наложенное на столбец (см. раздел «Проверка значений столбца с помощью ограничения CHECK» в главе 10);
- ни одно выражение не должно приводить к арифметической ошибке (например, переполнению или делению на нуль);
- вспомните из раздела «Таблицы, столбцы и строки» в главе 2, что порядок строк в таблице не имеет значения и что вы не можете управлять расположением строк, поэтому новые строки могут появиться в любом месте таблицы.

## Изменение строк

Введите:

```
UPDATE table
 SET column = expr
 [WHERE search_condition];
```

*table* – это название таблицы, которую вы будете обновлять; *column* – название столбца (с данными для изменения) в *table*; *expr* – буквенное обозначение, выражение или запрос, который считывает одно значение. Значение, считанное *expr*, заменит существующее значение в *column*. Чтобы изменить значения в нескольких столбцах, введите в пункте SET список выражений (*column = expr*), разделенных запятыми. Вы можете задавать список полей для обновления в любом порядке. Условие *search\_condition* задает условия, которые должны соблюдаться для изменяемых

**Листинг 9.8.** Заменить значение `contract` нулем во всех строках `titles`. Результат исполнения листинга показан на рис. 9.9

```
UPDATE titles
SET contract = 0;
```

**Листинг 9.9.** Удвоить цену на книги по истории. Результат исполнения листинга показан на рис. 9.9

```
UPDATE titles
SET price = price * 2.0
WHERE type = 'history';
```

**Листинг 9.10.** Изменить столбцы `type` и `pages` для книг по психологии. Результат исполнения листинга см. на рис. 9.9

```
UPDATE titles
SET type = 'self help',
 pages = NULL
WHERE type = 'psychology';
```

**Листинг 9.11.** Уменьшить в два раза продажи книг, которые находятся на среднем уровне. Результат исполнения листинга см. на рис. 9.9

```
UPDATE titles
SET sales = sales * 0.5
WHERE sales >
 (SELECT AVG(sales)
 FROM titles);
```

строк. Этими условиями могут быть условия `WHERE` (операторы сравнения, `LIKE`, `BETWEEN`, `IN` и `IS NULL`, см. главу 4) или условия запроса (операторы сравнения, `IN`, `ALL`, `ANY` и `EXISTS`, см. главу 8) в комбинации с `AND`, `OR` и `NOT`. Если вы опустите предложение `WHERE`, будут изменены все строки в таблице.

Листинг 9.8 заменяет значение `contract` нулем. Отсутствие предложения `WHERE` сообщает СУБД, что следует изменить значения в столбце `contract` во всех строках. Команда в листинге 9.8 изменяет 13 строк. Результат исполнения см. на рис. 9.9.

Листинг 9.9 использует арифметическое выражение и условие `WHERE`, чтобы удвоить цену на книги по истории. Команда, представленная в этом листинге, изменяет три строки. Результат исполнения см. на рис. 9.9.

Листинг 9.10 изменяет столбцы `type` и `pages` для книг по психологии. Вы используете только одно предложение `SET`, чтобы изменить несколько столбцов, разделив выражения `column = expr` запятыми (не следует ставить запятую в конце последнего выражения). Команда из листинга 9.10 изменяет три строки. Результат исполнения показан на рис. 9.9.

Листинг 9.11 использует подзапрос и функцию, чтобы уменьшить в два раза продажи книг, которые продаются лучше, чем все книги в целом. Команда, представленная в этом листинге, изменяет две строки. Результат исполнения показан на рис. 9.9.

Вы можете изменить значения в таблице на основании значений из другой таблицы. Листинг 9.12 использует подзапросы, чтобы изменить дату публикации для всех книг, которые написала Сара Бухманн (Sarah Buchmann). Представленная в этом листинге команда изменяет три строки. Результат исполнения показан на рис. 9.9.

Предположим, что издательство Abatis Publishers (P01) купило издательство Tenterhooks Press (P04). Теперь все книги последнего выпускаются в Abatis Publishers. Листинг 9.13 меняет информацию об издательстве в `titles` с P04 на P01. Запрос в предложении `WHERE` считывает `pub_id` для издательства Tenterhooks Press. СУБД использует `pub_id`, чтобы считать из таблицы `titles` все книги, которые выпускаются издательством Tenterhooks Press. Затем СУБД с помощью значения, считанного запросом в предложении `SET`, изменяет соответствующие строки в таблице `titles`. Поскольку запросы используются с неизменным оператором сравнения, они должны быть скалярными запросами и считывать единственные значения (то есть результат одной строки или столбца). См. раздел «Сравнение значений, возвращаемых подзапросом, с использованием операторов сравнения» в главе 8. Команда из листинга 9.13 изменяет пять строк.

На рис. 9.9 показана таблица `titles` после исполнения листингов 9.8–9.13. Каждый листинг изменяет данные в отдельном столбце (столбцах). Измененные значения выделены полужирным шрифтом.

**С** СУБД будет рассчитывать выражения в предложении `SET` или `WHERE` с использованием значений, которые находились в столбцах до начала изменений. Рассмотрим команду `UPDATE`:

```
UPDATE mytable
SET col1 = col1 * 2,
 col2 = col1 * 4,
 col3 = col2 * 8,
WHERE col1 = 1
 AND col2 = 2;
```

СУБД задает `col1` равным 2, `col2` – равным 4 (1×4, а не 2×4), а `col3` – равным 16 (2×8, а не 4×8).

**Листинг 9.12.** Заменить дату публикации для всех книг, которые написала Сара Бухманн, на 1 января 2003 года. Результат исполнения листинга показан на рис. 9.9

```
Листинг
UPDATE titles
SET pubdate = DATE '2003-01-01'
WHERE title_id IN
 (SELECT title_id
 FROM title_authors
 WHERE au_id IN
 (SELECT au_id
 FROM authors
 WHERE au_fname = 'Sarah'
 AND au_lname = 'Buchman'));
```

**Листинг 9.13.** «Приписать» все книги, выпущенные издательством Tenterhooks Press, издательству Abatis Publishers. Результат исполнения листинга показан на рис. 9.9

```
Листинг
UPDATE titles
SET pub_id =
 (SELECT pub_id
 FROM publishers
 WHERE pub_name = 'Abatis Publishers')
WHERE pub_id =
 (SELECT pub_id
 FROM publishers
 WHERE pub_name = 'Tenterhooks Press');
```



| title_id | title_name                          | type             | pub_id     | pages       | price        | sales         | pubdate           | contract |
|----------|-------------------------------------|------------------|------------|-------------|--------------|---------------|-------------------|----------|
| -----    |                                     |                  |            |             |              |               |                   |          |
| T01      | 1977!                               | history          | P01        | 107         | <b>43.98</b> | 566           | <b>2003-01-01</b> | <b>0</b> |
| T02      | 200 Years of German Humor           | history          | P03        | 14          | <b>39.90</b> | 9566          | <b>2003-01-01</b> | <b>0</b> |
| T03      | Ask Your System Administrator       | computer         | P02        | 1226        | 39.95        | 25667         | 2000-09-01        | <b>0</b> |
| T04      | But I Did It Unconsciously          | <b>self help</b> | <b>P01</b> | NULL        | 12.99        | 13001         | 1999-05-31        | <b>0</b> |
| T05      | Exchange of Platitudes              | <b>self help</b> | <b>P01</b> | NULL        | 6.95         | <b>100720</b> | 2001-01-01        | <b>0</b> |
| T06      | How About Never?                    | biography        | P01        | 473         | 19.95        | 11320         | 2000-07-31        | <b>0</b> |
| T07      | I Blame My Mother                   | biography        | P03        | 333         | 23.95        | <b>750100</b> | 1999-10-01        | <b>0</b> |
| T08      | Just Wait Until After School        | children         | <b>P01</b> | 86          | 10.00        | 4095          | 2001-06-01        | <b>0</b> |
| T09      | Kiss My Boo-Boo                     | children         | <b>P01</b> | 22          | 13.95        | 5000          | 2002-05-31        | <b>0</b> |
| T10      | Not Without My Faberge Egg          | biography        | P01        | NULL        | NULL         | NULL          | NULL              | <b>0</b> |
| T11      | Perhaps It's a Glandular Problem    | <b>self help</b> | <b>P01</b> | <b>NULL</b> | 7.99         | 94123         | 2000-11-30        | <b>0</b> |
| T12      | Spontaneous, Not Annoying           | biography        | P01        | 507         | 12.99        | 100001        | 2000-08-31        | <b>0</b> |
| T13      | What Are The Civilian Applications? | history          | P03        | 802         | <b>59.98</b> | 10467         | <b>2003-01-01</b> | <b>0</b> |

**Рис. 9.9.** Таблица `titles` после исполнения листингов 9.8–9.13. Измененные значения отмечены полужирным шрифтом

**П**

Если вы желаете быть особенно внимательными при изменении строк, можете проверить команду `UPDATE` с помощью временной копии таблицы (см. разделы «Создание временной таблицы с помощью команды `CREATE TEMPORARY TABLE`» и «Создание новой таблицы на основе существующей с помощью команды `SELECT INTO`» в главе 10).

**П**

Вы также можете изменять строки с помощью представлений (см. раздел «Изменение данных через представления» в главе 12).

**С**

Если вы применяете транзакции, воспользуйтесь командой `COMMIT` после команды `UPDATE`, чтобы сделать изменения постоянными. За информацией о транзакциях обращайтесь к главе 13.



В Microsoft Access при работе с датами не нужно вводить ключевое слово `DATE`. Выделять буквенное значение следует с помощью символов `#`, а не кавычек. Чтобы запустить листинг 9.12 в Access, в качестве значения даты используйте `#2003-01-01#`.

Microsoft Access не поддерживает скалярные запросы в предложении `SET`. Чтобы запустить листинг 9.13 в Access, разделите команду `UPDATE` на две команды: сначала с помощью команды `SELECT` выберите `pub_id` для издательства Abatis Publishers из таблицы `publishers`, затем с помощью `pub_id` измените значения для всех книг издательства Tenterhooks Press в таблице `titles`.

В Microsoft SQL Server при работе с датами не следует вводить ключевое слово `DATE`. Чтобы запустить листинг 9.12 в SQL Server, используйте значение даты `'2003-01-01'`.

MySQL 4.0 и более ранних версий не поддерживает подзапросы и не сможет выполнить листинги 9.11, 9.12 и 9.13. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Основные принципы работы с подзапросами» главы 8.

В PostgreSQL вам следует заменить числа с плавающей запятой в листингах 9.9 и 9.11 на `DECIMAL` (см. раздел «Преобразование типов данных с помощью функции `CAST()`» в главе 5). Чтобы запустить листинги 9.9 и 9.11 в PostgreSQL, используйте значения с плавающей запятой `CAST(2.0 AS DECIMAL)` (листинг 9.9) и `CAST(0.5 AS DECIMAL)` (листинг 9.11).

Посмотрите в документации, как ваша СУБД обрабатывает значения при вставке в столбцы, данные в которых автоматически создают новое значение для первичного ключа (см. примечание **DBMS** в разделе «Первичные ключи» главы 2). СУБД может не разрешить изменять значения в этих столбцах.

## Удаление строк с помощью команды DELETE

Команда DELETE удаляет строки из таблицы. Вы можете использовать эту команду, чтобы удалять:

- все строки в таблице;
- отдельные строки в таблице.

Чтобы удалить строки, нужно указать:

- строки в какой таблице следует удалить;
- условие поиска для нахождения удаляемых строк (опционально).

Перечислим важные параметры команды DELETE:

- в отличие от команд INSERT и UPDATE, эта команда не требует ввода названий столбцов, так как удаляет строки целиком;
- удаляет строки из таблицы, но не удаляет саму таблицу. Даже если вы уберете из таблицы все строки, сама таблица останется. Если вы желаете удалить таблицу (вместе со всеми данными, индексами и т.д.), обратитесь к разделу «Удаление таблицы с помощью команды DROP TABLE» в главе 10;
- использует (опционально) предложение WHERE, чтобы определить, какие именно строки следует удалить. Если вы не укажете условие поиска, команда DELETE удалит все строки в таблице;
- может быть небезопасна, потому что вы можете случайно пропустить предложение WHERE (и удалить все строки) или неправильно указать условие поиска для WHERE (и удалить не те строки). Перед запуском команды DELETE рекомендуем запустить команду SELECT с таким же предложением WHERE. Команда

SELECT отобразит все строки, которые будут удалены СУБД при запуске команды DELETE. Чтобы отобразить только количество таких строк, пользуйтесь командой SELECT COUNT (\*);

- чтобы сохранить ссылочную целостность, СУБД позволяет указать действие, которое будет выполняться автоматически с помощью команды DELETE при удалении строк, на которые указывает вторичный ключ (см. советы к разделу «Задание внешнего ключа с помощью ограничения FOREIGN KEY» в главе 10);
- ни одно выражение не должно приводить к арифметической ошибке (например, переполнению или делению на ноль);
- вспомните из раздела «Таблицы, столбцы и строки» в главе 2, что порядок строк в таблице не имеет значения и что вы не можете управлять расположением строк, поэтому их удаление может произвольным образом изменить расположение других строк в таблице.

### Удаление строк

Введите:

```
DELETE FROM table
[WHERE search_condition];
```

table – это название таблицы, из которой вы будете удалять строки. search\_condition задает условия, которые должны соблюдаться для удаляемых строк. Этими условиями могут быть условия WHERE (операторы сравнения, LIKE, BETWEEN, IN и IS NULL, см. главу 4) или условия запроса (операторы сравнения, IN, ALL, ANY и EXISTS, см. главу 8) в комбинации с AND, OR и NOT. Если вы опустите предложение WHERE, будут удалены все строки в таблице.

В следующих примерах мы будем игнорировать ссылки. Разумеется, не следует это делать при работе с настоящей базой данных.

Листинг 9.14 удаляет все строки в `royalties`. Отсутствие предложения `WHERE` сообщает СУБД, что следует удалить все строки. Команда, представленная в этом листинге, удаляет 13 строк. Результат исполнения показан на рис. 9.10.

В листинге 9.15 предложение `WHERE` сообщает СУБД, что из таблицы `authors` следует удалить всех авторов с фамилией Халл (Hull). Команда этого листинга удаляет две строки. Результат исполнения см. на рис. 9.11.

Вы можете использовать данные из одной таблицы, чтобы удалить данные в другой таблице. Листинг 9.16 использует подзапрос, чтобы удалить из `title_authors` все книги, выпущенные издательствами P01 и P04. Эта команда удаляет 12 строк. Результат исполнения показан на рис. 9.12.

**Листинг 9.14.** Удалить все строки в `royalties`.  
Результат исполнения листинга см. на рис. 9.10

Листинг

DELETE FROM royalties;

| title_id | advance | royalty_rate |
|----------|---------|--------------|
| -----    | -----   | -----        |

**Рис. 9.10.** Результат выполнения листинга 9.14

**Листинг 9.15.** Удалить из `authors` всех авторов с фамилией Халл. Результат исполнения листинга см. на рис. 9.11

Листинг

DELETE FROM authors  
WHERE au\_lname = 'Hull';

**Листинг 9.16.** Удалить из `title_authors` все книги, выпущенные издательствами P01 и P04. Результат исполнения листинга см. на рис. 9.12

Листинг

DELETE FROM title\_authors  
WHERE title\_id IN  
(SELECT title\_id  
FROM titles  
WHERE pub\_id IN ('P01', 'P04'));

| au_id | au_fname  | au_lname    | phone        | address          | city      | state | zip   |
|-------|-----------|-------------|--------------|------------------|-----------|-------|-------|
| ----- | -----     | -----       | -----        | -----            | -----     | ----- | ----- |
| A01   | Sarah     | Buchman     | 718-496-7223 | 75 West 205 St   | Bronx     | NY    | 10468 |
| A02   | Wendy     | Heydemark   | 303-986-7020 | 2922 Baseline Rd | Boulder   | CO    | 80303 |
| A05   | Christian | Kells       | 212-771-4680 | 114 Horatio St   | New York  | NY    | 10014 |
| A06   |           | Kellsey     | 650-836-7128 | 390 Serra Mall   | Palo Alto | CA    | 94305 |
| A07   | Paddy     | O'Furniture | 941-925-0752 | 1442 Main St     | Sarasota  | FL    | 34236 |

**Рис. 9.11.** Результат выполнения листинга 9.15

| title_id | au_id | au_order | royalty_share |
|----------|-------|----------|---------------|
| -----    | ----- | -----    | -----         |
| T02      | A01   | 1        | 1.00          |
| T03      | A05   | 1        | 1.00          |
| T07      | A02   | 1        | 0.50          |
| T07      | A04   | 2        | 0.50          |
| T13      | A01   | 1        | 1.00          |

Рис. 9.12. Результат выполнения листинга 9.16

**П**

Если вы желаете быть особенно внимательными при удалении строк, можете проверить команду `DELETE` с помощью временной копии таблицы (см. разделы «Создание временной таблицы с помощью команды `CREATE TEMPORARY TABLE`» и «Создание новой таблицы на основе существующей с помощью команды `SELECT INTO`» в главе 10).

**П**

Вы также можете удалять строки с помощью представлений (см. раздел «Изменение данных через представления» в главе 12).

**С**

Если вы применяете транзакции, воспользуйтесь командой `COMMIT` после команды `DELETE`, чтобы сделать изменения постоянными. За информацией о транзакциях обращайтесь к главе 13.

**С**

Если вы желаете удалить все строки в таблице, лучше использовать команду `TRUNCATE` вместо `DELETE`. Команда `TRUNCATE` не является стандартной командой SQL, но поддерживается Microsoft SQL Server, Oracle, MySQL и PostgreSQL. Эта команда функционально идентична `DELETE` без предложения `WHERE` и тоже удаляет все строки в таблице. Но `TRUNCATE` работает быстрее и потребляет меньше системных ресурсов, чем `DELETE`, потому что не сканирует всю таблицу и не записывает результаты в журнал (см. главу 13). Есть еще одно отличие: при работе с `TRUNCATE` вы сможете отменить изменения, если допустите ошибку. Приведем структуру команды:

```
TRUNCATE TABLE table;
```

*table* — это таблица, которую вы желаете изменить. За дополнительной информацией о команде `TRUNCATE` обращайтесь к руководству по вашей СУБД.



В SQL ключевое слово `FROM` является опцией для команды `DELETE`. Microsoft Access, MySQL и PostgreSQL требуют обязательного ввода этого слова, поэтому мы нигде его не опускаем.

MySQL 4.0 и более ранних версий не поддерживает подзапросы и не сможет выполнить листинг 9.16. За дополнительной информацией обращайтесь к примечанию **DBMS** в разделе «Основные принципы работы с подзапросами» главы 8.

# СОЗДАНИЕ, ИЗМЕНЕНИЕ И УДАЛЕНИЕ ТАБЛИЦ

---

## 10

Многие СУБД располагают интерактивными графическими инструментами, с помощью которых вы можете создавать и изменять таблицы и их свойства, например названия столбцов и ограничения. В текущей главе мы объясним, как использовать инструменты SQL, чтобы работать с таблицами:

- команда `CREATE TABLE` создает новую таблицу;
- команда `ALTER TABLE` изменяет структуру существующей таблицы;
- команда `DROP TABLE` удаляет таблицу и все ее данные;
- команда `CREATE TEMPORARY TABLE` создает временную таблицу, которая затем будет удалена;
- команда `SELECT INTO` создает новую таблицу на основе уже существующей.

Перечисленные команды не считывают результат, но ваша СУБД выдаст сообщение о том, была ли выполнена команда или нет. Чтобы увидеть эффект, который команда произвела на таблицу, изучите структуру таблиц с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9. Эти команды изменяют объекты базы данных (а в некоторых случаях и сами данные), вот почему администратор базы данных может не разрешить их использовать.

## Порядок создания таблиц

Дизайнеры баз данных тратят много времени на упорядочивание таблиц и организацию ограничений и ссылок, чтобы затем написать одну строку SQL-кода. Если вы будете создавать таблицы для рабочих баз данных, мы рекомендуем изучить принципы проектирования баз данных, которые описываются в главе 2.

Вспомните из раздела «Таблицы, столбцы и строки» в главе 2, что базы данных строятся на основе таблиц. Для обычного пользователя и для SQL-программиста база данных представляет собой группу таблиц (и более ничего). Чтобы создать таблицу, следует указать:

- название таблицы;
- названия столбцов;
- тип данных для столбцов;
- ограничения.

Названия таблицы и столбцов должны соответствовать требованиям SQL. См. раздел «Синтаксис SQL» в главе 3 (каждая СУБД имеет собственные требования). Типом данных для каждого столбца могут быть символы, числа, информация о дате и времени или данные другого типа (см. раздел «Типы данных» в главе 3). *Ограничения* позволяют задавать свойства, например вводить значения null, значения по

умолчанию, ключи и допустимые значения.

Вы создаете новую таблицу с помощью команды CREATE TABLE, которая имеет следующую структуру:

```
CREATE TABLE table
(
 column1 data_type1 [col_constraints1],
 column2 data_type2 [col_constraints2],
 ...
 columnN data_typeN [col_constraintsN]
 [, table_constraint1]
 [, table_constraint2]
 ...
 [, table_constraintN]
);
```

Каждое обозначение столбца включает его название, тип данных, а также список ограничений для столбца (опционально). Не следует разделять ограничения для разных столбцов с помощью запятых. Список ограничений таблицы вводится после описания последнего столбца. За описанием каждого столбца (кроме последнего) и ограничений следует запятая. Чтобы получить дополнительную информацию об ограничениях, обратитесь к разделу «Основные принципы работы с ограничениями».

Ввод описания каждого столбца таблицы начинается с новой строки<sup>1</sup>.

<sup>1</sup> Это не является требованием стандарта, однако повышает читабельность команд. — Прим. науч. ред.



Таблица 10.1. Ограничения

| Ограничения | Описание                                                                                   |
|-------------|--------------------------------------------------------------------------------------------|
| NOT NULL    | Не разрешает присваивать столбцу значение null                                             |
| DEFAULT     | Задаёт для столбца значение по умолчанию                                                   |
| PRIMARY KEY | Задаёт столбец (столбцы) первичного ключа для таблицы                                      |
| FOREIGN KEY | Задаёт столбец (столбцы) вторичного ключа для таблицы                                      |
| UNIQUE      | Не разрешает добавлять в столбец повторяющиеся значения                                    |
| CHECK       | Ограничивает значения, которые могут добавляться в столбец, с помощью логических выражений |

## Основные принципы работы с ограничениями

Ограничения позволяют указать требования к значениям в столбцах (см. табл. 10.1). Ваша СУБД будет использовать эти требования, чтобы автоматически редактировать таблицу. Ограничения бывают двух типов:

- *ограничение столбца* является частью описания столбца и действует только для данного столбца;
- *ограничение таблицы* не зависит от ограничений столбца и может влиять на несколько столбцов в таблице. Чтобы включить в ограничения требования для нескольких столбцов, следует использовать ограничение таблицы.

Вы можете указать несколько ограничений для столбца или таблицы. Использование ограничений будет зависеть от контекста. Например, если первичный ключ содержит один столбец, вы можете задать его в качестве ограничения столбца или таблицы. Если первичный ключ включает два столбца или более, следует использовать ограничение таблицы.

Если вы укажете названия ограничений, это упростит работу с ними. Например, вы легко сможете изменить или удалить такое ограничение с помощью команды ALTER TABLE. Названия ограничений вводить необязательно, но многие SQL-программисты и дизайнеры баз данных присваивают названия всем ограничениям. Вы можете не давать названий ограничениям NOT NULL и DEFAULT, но мы рекомендуем указать названия всех других ограничений (несмотря на то, что в примерах данной книги мы этого не сделали).

Если вы не присвоили названия ограничению, СУБД сделает это за вас. Подобные названия включают много символов, и их

очень непросто использовать; лучше с помощью команды `CONSTRAINT` придумайте собственное название ограничению. Такие названия в предупреждениях и сообщениях об ошибках, что служит еще одной причиной для того, чтобы самостоятельно их присваивать.

### Присвоение названия ограничению

Перед тем как вводить описание ограничения, напечатайте:

```
CONSTRAINT constraint_name
```

*constraint\_name* — это название ограничения, которое будет использоваться SQL для его идентификации. Такие названия для одной таблицы должны быть уникальными.

Ограничения с названиями будут показаны далее в примерах этой главы.



MySQL не поддерживает предложение `CONSTRAINT` для ограничений столбцов (но вы можете использовать его для ограничений таблиц).

Листинг 10.1. Создать таблицу titles

```
CREATE TABLE titles
(
 title_id CHAR(3) ,
 title_name VARCHAR(40) ,
 type VARCHAR(10) ,
 pub_id CHAR(3) ,
 pages INTEGER ,
 price DECIMAL(5,2) ,
 sales INTEGER ,
 pubdate DATE ,
 contract SMALLINT
);
```

Листинг 10.2. Создать таблицу title\_authors

```
CREATE TABLE title_authors
(
 title_id CHAR(3) ,
 au_id CHAR(3) ,
 au_order SMALLINT ,
 royalty_share DECIMAL(5,2)
);
```

## Создание новой таблицы с помощью команды CREATE TABLE

В этом разделе мы покажем, как создать новую таблицу с помощью команды CREATE TABLE. В следующих разделах вы научитесь добавлять к этой команде ограничения столбца и таблицы.

### Создание новой таблицы

Введите:

```
CREATE TABLE table
(
 column1 data_type1,
 column2 data_type2,
 ...
 column data_typeN
);
```

*table* — это название новой таблицы, которую вы создаете; *column1*, *column2*, ..., *columnN* — названия столбцов в *table*. Вы должны создать хотя бы один столбец.

*data\_type1*, *data\_type2*, ..., *data\_typeN* задают тип данных SQL для соответствующих столбцов. Тип данных может включать длину, масштаб или точную спецификацию (см. главу 3).

Названия таблицы в базе данных и каждого столбца в таблице должны быть уникальными.

Листинг 10.1 создает таблицу *titles*, листинг 10.2 — таблицу *title\_authors*.

**С**

Чтобы увидеть результат исполнения команды `CREATE TABLE`, изучите структуру таблицы с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.

**П**

Если вы попытаетесь создать таблицу с названием, которое уже существует в базе данных, СУБД выдаст ошибку. Во избежание сохранения одной таблицы вместо другой SQL требует, чтобы до создания таблицы вы удалили таблицу с тем же названием с помощью команды `DROP TABLE` (см. раздел «Удаление таблицы с помощью команды `DROP TABLE`» далее в этой главе).

**С**

Сразу после создания таблица пуста (не имеет строк). Чтобы заполнить таблицу данными, пользуйтесь командой `INSERT` (см. раздел «Вставка строк с помощью команды `INSERT`» в главе 9).

**С**

По умолчанию значения `null` в столбцах разрешены. Если вы желаете их запретить, обратитесь к разделу «Запрет значения `null` с помощью ограничения `NOT NULL`» далее в этой главе.

**С**

Чтобы изменить структуру существующей таблицы, обратитесь к разделу «Изменение таблицы с помощью команды `ALTER TABLE`» далее в этой главе.

**С**

Чтобы создать таблицу на основе структуры и данных существующей таблицы, обратитесь к разделу «Создание новой таблицы на основе существующей с помощью команды `SELECT INTO`» далее в этой главе.



Microsoft SQL Server не поддерживает тип данных `DATE`. Чтобы запустить листинг 10.1 в SQL Server, в столбце `pubdate` воспользуйтесь типом данных `DATETIME`.

MySQL без предупреждения заменит столбцы `VARCHAR`, которые состоят менее чем из четырех символов, на `CHAR`.

## Запрет значения null с помощью ограничения NOT NULL

От способности столбца принимать значения null зависит, могут ли строки содержать NULL, то есть обязательно ли вводить значения в этих строках или нет. Про значение null и его свойства мы уже рассказывали в разделе «Значение null» главы 3. Вкратце повторим:

- значение null является маркером, который сообщает, что значение не было введено;
- значение null отображает значение, которого нет, которое неизвестно или которое неприменимо. Значение null в столбце price свидетельствует о том, что цена неизвестна или не была указана, а не о том, что товар не имеет цены или что цена равна нулю;
- значение null – не то же самое, что (0), пустое поле или пробел (' ');
- значение null не относится ни к одному типу данных и может быть вставлено в любой столбец, который это допускает;
- в командах SQL слово NULL указывает на значение null.

При установке ограничения на значение null следует учитывать следующие факторы:

- ограничение на значение null всегда является ограничением столбца, а не таблицы (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);
- вы задаете ограничения на значение null с помощью ключевых слов NULL или NOT

NULL в обозначении столбца в команде CREATE TABLE;

- желательно избегать разрешения значения null, так как это усложняет запросы;
- запрет значения null в столбце может помочь сохранить целостность данных, так как при их вводе обязательно указывать значения. СУБД не будет вставлять или изменять строку, если столбец (для которого запрещено значение null) содержит NULL;
- некоторые ограничения (например, PRIMARY KEY) не могут использоваться в столбцах, для которых разрешено значение null;
- значение null влияет на ограничения вторичных ключей (см. раздел «Задание вторичного ключа с помощью ограничения FOREIGN KEY» далее в этой главе);
- если вы добавили строку с помощью команды INSERT, но не указали значение столбца, который допускает значение null, ваша СУБД вставит NULL (при условии, что нет ограничения DEFAULT). См. разделы «Вставка строк с помощью команды INSERT» в главе 9 и «Присвоение значения по умолчанию с помощью ограничения DEFAULT» далее в этой главе;
- вы можете ввести NULL непосредственно в столбце, для которого разрешено значение null, причем независимо от того, какой тип данных или значение по умолчанию указаны для этого столбца;
- если вы не укажете NULL или NOT NULL, по умолчанию значение null будет разрешено.

## Установка ограничения NOT NULL для столбца

Добавьте к описанию столбца в команде CREATE TABLE следующее ограничение столбца:

```
[CONSTRAINT constraint_name]
[NOT] NULL
```

Укажите ограничение NULL, чтобы разрешить значение null для столбца, или NOT NULL, чтобы запретить. Если вы не указали ничего, по умолчанию задается ограничение NULL. Дополнительную информацию об общей структуре команды CREATE TABLE см. в разделе «Порядок создания таблиц» ранее в этой главе. Ключевое слово CONSTRAINT является опциональным, а *constraint\_name* — это название ограничения (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе).

Листинг 10.3 создает таблицу authors и задает для каждого столбца ограничение на значение null. Адреса и телефонные номера часто пропускаются, поэтому для этих столбцов мы разрешили значение null. Обратите внимание, что в столбцах с именами и фамилиями мы их запретили. Если приводится только фамилия автора и отсутствует его имя (например, автор A06, Келси (Kellsey)), в столбец au\_lname мы добавим фамилию, а в столбец au\_fname — пустую строку ("). Либо мы можем разрешить значение null в поле au\_fname и вставлять его в столбец au\_fname для авторов, имена которых не указаны. Либо мы можем разрешить значение null в столбцах au\_fname и au\_lname и добавить ограничение, которое требует ввода данных хотя бы в одном столбце строки. Дизайнер базы данных принимает подобное решение еще до создания таблицы.

**Листинг 10.3.** Создать таблицу authors и задать для каждого столбца ограничение на значение null

```
Листинг
CREATE TABLE authors
(
 au_id CHAR(3) NOT NULL ,
 au_fname VARCHAR(15) NOT NULL ,
 au_lname VARCHAR(15) NOT NULL ,
 phone VARCHAR(12) NULL ,
 address VARCHAR(20) NULL ,
 city VARCHAR(15) NULL ,
 state CHAR(2) NULL ,
 zip CHAR(5) NULL
);
```

**Листинг 10.4.** Создать базу данных titles. Если в описании столбца ограничение опущено, то для него СУБД разрешает значение null

```
Листинг
CREATE TABLE titles
(
 title_id CHAR(3) NOT NULL ,
 title_name VARCHAR(40) NOT NULL ,
 type VARCHAR(10) ,
 pub_id CHAR(3) NOT NULL ,
 pages INTEGER ,
 price DECIMAL(5,2) ,
 sales INTEGER ,
 pubdate DATE ,
 contract SMALLINT NOT NULL
);
```

Листинг 10.4 создает базу данных titles. Если в описании столбца ограничение опущено, то для него СУБД разрешает значение null

**С** Чтобы увидеть результат исполнения команды CREATE TABLE, изучите структуру таблицы с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.

**П** При добавлении строки в таблицу нужно указать точные значения для столбцов, в которых запрещено значение null (и не заданы значения по умолчанию). Например, команда INSERT для таблицы authors, созданной листингом 10.3, выглядит так:

```
INSERT INTO authors (
 au_id,
 au_fname,
 au_lname)
VALUES (
 'A08',
 'Michael',
 'Polk');
```

СУБД автоматически помещает NULL в столбцы таблицы authors, которые не были указаны в списке столбцов команды INSERT (phone, address и т.д.); см. раздел «Вставка строк с помощью команды INSERT» в главе 9.

**С** При добавлении данных для строкового поля не помещайте ключевое слово NULL в кавычки. Если вы это сделаете, СУБД интерпретирует это как ввод слова NULL, а не как значение null.

**П** Вы можете найти значение null с помощью команды IS NULL (см. раздел «Проверка на значение null с помощью оператора IS NULL» в главе 4).

**С** Чтобы отобразить значение вместо NULL, можно использовать функцию COALESCE() (см. раздел «Проверка на значения null с использованием функции COALESCE()» в главе 5).

**П** Можно использовать функцию NULLIF(), чтобы конвертировать значения в расчеты (см. раздел «Сравнение выражений с помощью функции NULLIF()» в главе 5).

**П** Все столбцы, для которых было задано ограничение PRIMARY KEY, должны быть обозначены как NOT NULL. Если вы не зададите этот параметр, СУБД автоматически укажет для всех столбцов PRIMARY KEY параметр NOT NULL (см. раздел «Задание первичного ключа с помощью ограничения PRIMARY KEY» далее в этой главе).



Microsoft SQL Server не поддерживает тип данных DATE. Чтобы запустить листинг 10.4 в SQL Server, замените тип данных в столбце pubdate на DATETIME.

MySQL не поддерживает ключевое слово CONSTRAINT для ограничений столбцов (но вы можете использовать его для ограничений таблиц).

Oracle распознает пустую строку (') как NULL (см. примечание **DBMS** в разделе «Значение null» главы 3). Для СУБД, которые мы рассматриваем в данной главе, ограничение на значение null является опцией, но другие СУБД могут потребовать, чтобы вы задали для каждого столбца параметр NULL или NOT NULL.

Чтобы узнать, как СУБД работает с ограничением на значение null для столбцов, данные в которых автоматически создают уникальные значения для строк, обратитесь к документации. См. примечание **DBMS** в разделе «Первичные ключи» главы 2.

## Присвоение значения по умолчанию с помощью ограничения DEFAULT

Значение по умолчанию присваивает значение, которое СУБД задает для столбца, если вы пропустите значение при добавлении строки. См. раздел «Вставка строк с помощью команды INSERT» в главе 9. При работе со значениями по умолчанию вам следует помнить, что:

- ограничение для значения по умолчанию всегда является ограничением столбца, а не таблицы (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);
- вы задаете ограничение для значения по умолчанию с помощью ключевого слова DEFAULT в описании столбца в команде CREATE TABLE;
- значение по умолчанию может быть любым выражением, результатом исполнения которого является константа;
- значение по умолчанию должно относиться к тому же типу данных (или быть конвертируемым), что и соответствующий столбец (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5);
- столбец должен быть такой длины, чтобы вместить значение по умолчанию;
- если вы добавили строку с помощью INSERT, но не указали значение для столбца, ваша СУБД будет использовать значение по умолчанию. Если для столбца не задано значение по умолчанию, будет использоваться значение null;
- если для столбца установлено ограничение NOT NULL, но не задано значение по умолчанию, при добавлении строки с

помощью INSERT необходимо указать значения. В противном случае СУБД отобразит сообщение об ошибке и не вставит строку (см. раздел «Вставка строк с помощью команды INSERT» в главе 9).

### Задание столбцу значения по умолчанию

Добавьте к описанию столбца в команде CREATE TABLE следующее ограничение столбца:

```
[CONSTRAINT constraint_name]
DEFAULT expr
```

*expr* — это выражение, которое является константой, например буквой, встроенной функцией, математическим выражением или NULL. Если не было задано ограничение значению по умолчанию, указывается ограничение NULL. Информацию об общей структуре команды CREATE TABLE см. в разделе «Порядок создания таблиц» ранее в этой главе. Ключевое слово CONSTRAINT является опциональным, а *constraint\_name* — это название ограничения (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе).

Листинг 10.5 задает значения по умолчанию для столбцов в таблице titles. Для столбцов title\_id и pub\_id не присвоены значения по умолчанию и установлено ограничение NOT NULL, поэтому значения для них вы должны указать в команде INSERT. Пометка DEFAULT NULL для столбца pages эквивалентна пропуску ограничения DEFAULT. Значения по умолчанию pubdate и contract показывают, что они могут быть более сложными выражениями, чем обычные буквенные.

Листинг 10.6 содержит команду INSERT, которой вы можете пользоваться, чтобы добавить строку в таблицу titles (созданную листингом 10.5).



**С** Чтобы увидеть результат исполнения команды CREATE TABLE, изучите структуру таблицы с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.



Microsoft Access не разрешает применять арифметические выражения в ограничении DEFAULT; пользуйтесь цифрами. Чтобы отобразить системную дату, пользуйтесь функцией DATE(), а не CURRENT\_DATE (см. примечание **DBMS** в разделе «Извлечение значений текущих даты и времени» главы 5). Чтобы запустить листинг 10.5 в Access,

замените ограничение значения по умолчанию для столбца pubdate на DEFAULT DATE(), а ограничение значения по умолчанию для столбца contract — на DEFAULT 0. Microsoft SQL Server не поддерживает тип данных DATE. Вместо него используйте DATETIME. Чтобы отобразить дату системы, пользуйтесь функцией GETDATE(), а не CURRENT\_DATE (см. примечание **DBMS** в разделе «Извлечение значений текущих даты и времени» главы 5). Чтобы запустить листинг 10.5 в SQL Server, замените тип данных для столбца pubdate на DATETIME, а ограничение значения по умолчанию — на DEFAULT GETDATE 0.

**Листинг 10.5.** Задать значения по умолчанию для столбцов в таблице titles

```

CREATE TABLE titles
(
 title_id CHAR(3) NOT NULL
 title_name VARCHAR(40) NOT NULL DEFAULT ''
 type VARCHAR(10) NULL DEFAULT 'undefined'
 pub_id CHAR(3) NOT NULL
 pages INTEGER NULL DEFAULT NULL
 price DECIMAL(5,2) NOT NULL DEFAULT 0.00
 sales INTEGER NULL
 pubdate DATE NULL DEFAULT CURRENT_DATE
 contract SMALLINT NOT NULL DEFAULT (3 * 7) - 21
);

```

**Листинг 10.6.** СУБД добавляет значения по умолчанию для столбцов, пропущенных в команде INSERT. Если не было задано значение по умолчанию, СУБД добавит значение null. Результат исполнения показан на рис. 10.1

```

INSERT INTO titles(title_id, pub_id) VALUES('T14', 'P01');

```

| title_id | title_name | type      | pub_id | pages | price | sales | pubdate    | contract |
|----------|------------|-----------|--------|-------|-------|-------|------------|----------|
| T14      |            | undefined | P01    | NULL  | 0.00  | NULL  | 2002-05-06 | 0        |

**Рис. 10.1.** Листинг 10.6 добавит в таблицу titles новую строку

В Oracle ограничения для значений по умолчанию следуют за типом данных до всех других ограничений столбцов, включая ограничение на значение null. Oracle 9i и более поздних версий поддерживает CURRENT\_DATE. В Oracle 8i и более ранних версиях вместо CURRENT\_DATE следует использовать SYSDATE (см. примечание **DBMS** в разделе «Извлечение значений текущих даты и времени» главы 5). Oracle воспринимает пустую строку (') как значение null, поэтому мы заменили значение по умолчанию title\_name символом пробела (см. примечание **DBMS** в разделе «Значение null» главы 3). Версия листинга 10.5 для Oracle представлена в листинге 10.7.

Oracle не поддерживает ключевое слово CONSTRAINT для ограничения DEFAULT (но вы можете использовать его для других ограничений столбцов и всех ограничений таблиц).

В MySQL значения по умолчанию должны быть буквенными. Например, вы не можете задать по умолчанию результат функции или арифметического выражения. Это значит, что вы не можете задать в качестве

значения по умолчанию для столбца с датой CURRENT\_DATE. Чтобы запустить листинг 10.5 в MySQL, удалите ограничение значения по умолчанию столбца pubdate (или замените значение по умолчанию на буквенное), а также замените ограничение значения по умолчанию для столбца contract на DEFAULT 0.

MySQL не поддерживает ключевое слово CONSTRAINT для ограничений столбца (но вы можете использовать его для ограничений таблицы).

Если для столбца не было задано ограничение DEFAULT, но задано ограничение NOT NULL, некоторые СУБД будут устанавливать значение по умолчанию в соответствии с типом данных столбца. Например, MySQL будет присваивать значение по умолчанию для числовых столбцов, в которых запрещены значения null.

Чтобы узнать, как СУБД работает с ограничениями значения по умолчанию для столбцов, данные в которых автоматически создают уникальные значения для строк, обратитесь к документации. См. примечание **DBMS** к разделу «Первичные ключи» в главе 2.

**Листинг 10.7.** В Oracle ограничение для значения по умолчанию должно следовать перед другими ограничениями столбца

| Листинг             |              |                      |           |
|---------------------|--------------|----------------------|-----------|
| CREATE TABLE titles |              |                      |           |
| (                   |              |                      |           |
| title_id            | CHAR(3)      |                      | NOT NULL, |
| title_name          | VARCHAR(40)  | DEFAULT ' '          | NOT NULL, |
| type                | VARCHAR(10)  | DEFAULT 'undefined'  | NULL ,    |
| pub_id              | CHAR(3)      |                      | NOT NULL, |
| pages               | INTEGER      | DEFAULT NULL         | ,         |
| price               | DECIMAL(5,2) | DEFAULT 0.00         | NOT NULL, |
| sales               | INTEGER      |                      | NULL ,    |
| pubdate             | DATE         | DEFAULT SYSDATE      | NULL ,    |
| contract            | SMALLINT     | DEFAULT (3 * 7) - 21 | NOT NULL  |
| );                  |              |                      |           |

## Задание первичного ключа с помощью ограничения PRIMARY KEY

Про *первичные ключи* мы уже рассказывали в одноименном разделе главы 2. Вкратце повторим:

- первичный ключ идентифицирует каждую строку в таблице как уникальную;
- две строки не могут иметь одинаковый первичный ключ;
- первичные ключи не могут содержать NULL;
- в каждой таблице может быть только один первичный ключ;
- первичный ключ – это столбец или несколько столбцов. *Простой первичный ключ* состоит из одного столбца; *сложный* – из нескольких;
- в *сложном ключе* значения в одном столбце могут повторяться, но комбинация значений во всех столбцах ключа должна быть уникальной;
- таблица может иметь несколько комбинаций столбцов, которые идентифицируют ее строки. Дизайнер базы данных выбирает одну из комбинаций и задает ее как первичный ключ.

При установке ограничения первичного ключа следует принять в расчет следующую информацию:

- простой ключ всегда может быть как ограничением столбца, так и ограничением таблицы; сложный ключ может быть только ограничением таблицы (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);

- вы задаете ограничение первичного ключа с помощью ключевых слов PRIMARY KEY в обозначении столбца в команде CREATE TABLE;
- если PRIMARY KEY является ограничением таблицы, вам необходимо задать названия столбца (столбцов). Если PRIMARY KEY является ограничением столбца, оно относится к соответствующему столбцу;
- SQL позволяет создать таблицу без первичного ключа (нарушая требования к модели таблицы). На практике вам всегда следует задавать первичный ключ для любой таблицы;
- для одной таблицы разрешено задавать не более одного первичного ключа;
- ограничения первичных ключей почти всегда имеют уникальные названия. Чтобы задать название первичного ключа, пользуйтесь предложением CONSTRAINT (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);
- ограничением на значение null для столбцов первичных ключей должно быть NOT NULL. Если вы не укажете это ограничение, СУБД автоматически задаст его для всех столбцов первичных ключей (см. раздел «Запрет значения null с помощью ограничения NOT NULL» ранее в этой главе);
- при добавлении строк с помощью команды INSERT следует указать значения для первичного ключа, в противном случае они будут сгенерированы автоматически в соответствии с типом данных столбца (см. примечание DBMS к разделу «Первичные ключи» в главе 2). За информацией о добавлении строк обращайтесь к разделу «Вставка строк с помощью команды INSERT» в главе 9;

- значения первичного ключа редко изменяются. Мы не рекомендуем вам изменять значения первичных ключей с помощью команды UPDATE (см. раздел «Изменение строк с помощью команды UPDATE» в главе 9);
- не используйте для других строк значение первичного ключа удаленной с помощью команды DELETE строки (см. раздел «Удаление строк с помощью команды DELETE» в главе 9);
- по всем вопросам, связанным с добавлением, изменением и удалением первичных ключей, связанных с внешними ключами, обращайтесь к разделу «Задание внешнего ключа с помощью ограничения FOREIGN KEY» далее в этой главе.

## Задание простого первичного ключа

Чтобы задать простой первичный ключ в качестве ограничения столбца, добавьте к описанию столбца в команде CREATE TABLE следующее ограничение столбца:

```
[CONSTRAINT constraint_name]
PRIMARY KEY
```

Или

Чтобы задать простой первичный ключ в качестве ограничения таблицы, добавьте после описания столбцов в команде CREATE TABLE следующее ограничение таблицы:

```
[CONSTRAINT constraint_name]
PRIMARY KEY (key_column)
```

*key\_column* — это название столбца первичного ключа. Для одной таблицы можно ввести только один первичный ключ. Информацию об общей структуре команды CREATE TABLE см. в разделе «Порядок создания таблиц» ранее в этой главе. Предложение CONSTRAINT является опциональным, а *constraint\_name* — это название ограничения (см. раздел «Основные

принципы работы с ограничениями» ранее в этой главе).

Листинги 10.8а, 10.8б и 10.8в отражают три эквивалентных способа задать простой первичный ключ для таблицы publishers.

Листинг 10.8а использует ограничение столбца, чтобы задать первичный ключ. Здесь показан самый доступный способ создания простого первичного ключа.

Листинг 10.8б использует неименованное ограничение таблицы, чтобы задать первичный ключ. Мы добавили к столбцу pub\_id ограничение NOT NULL, хотя это не обязательно, так как СУБД самостоятельно задает его (но не в MySQL; см. примечание DBMS далее в этой главе).

**Листинг 10.8а.** Задать простой первичный ключ для таблицы publishers с помощью ограничения столбца

```
Листинг
CREATE TABLE publishers
(
 pub_id CHAR(3) PRIMARY KEY,
 pub_name VARCHAR(20) NOT NULL ,
 city VARCHAR(15) NOT NULL ,
 state CHAR(2) NULL ,
 country VARCHAR(15) NOT NULL
);
```

**Листинг 10.8б.** Задать простой первичный ключ без названия для таблицы publishers с помощью ограничения таблицы

```
Листинг
CREATE TABLE publishers
(
 pub_id CHAR(3) NOT NULL,
 pub_name VARCHAR(20) NOT NULL,
 city VARCHAR(15) NOT NULL,
 state CHAR(2) NULL ,
 country VARCHAR(15) NOT NULL,
 PRIMARY KEY (pub_id)
);
```

**Листинг 10.8в.** Задать именованный простой первичный ключ для таблицы `publishers` с помощью ограничения таблицы

```

CREATE TABLE publishers
(
 pub_id CHAR(3) NOT NULL,
 pub_name VARCHAR(20) NOT NULL,
 city VARCHAR(15) NOT NULL,
 state CHAR(2) NULL,
 country VARCHAR(15) NOT NULL,
 CONSTRAINT publishers_pk
 PRIMARY KEY (pub_id)
);

```

**Листинг 10.9.** Задать сложный первичный ключ для таблицы `title_authors` с помощью именованного ограничения таблицы

```

CREATE TABLE title_authors
(
 title_id CHAR(3) NOT NULL,
 au_id CHAR(3) NOT NULL,
 au_order SMALLINT NOT NULL,
 royalty_share DECIMAL(5,2) NOT NULL,
 CONSTRAINT title_authors_pk
 PRIMARY KEY (title_id, au_id)
);

```

Листинг 10.8в использует именованное ограничение таблицы, чтобы задать первичный ключ. Здесь показан предпочтительный способ добавления первичного ключа; если вы решите впоследствии удалить или изменить ключ, можете использовать название `publishers_pk` (см. раздел «Изменение таблицы с помощью команды `ALTER TABLE`» далее в этой главе).

## Задание сложного первичного ключа

Добавьте к описанию столбца в команде `CREATE TABLE` следующее ограничение таблицы:

```

[CONSTRAINT constraint_name]
PRIMARY KEY (key_columns)

```

*key\_column* — это список столбцов первичного ключа, разделенных запятыми. Для одной таблицы можно ввести только первичный ключ. Информацию об общей структуре команды `CREATE TABLE` см. в разделе «Порядок создания таблиц» ранее в этой главе. Предложение `CONSTRAINT` является опциональным, а *constraint\_name* — это название ограничения (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе).

Листинг 10.9 задает сложный первичный ключ для таблицы `title_authors`. В него войдут столбцы `title_id` и `au_id`, а ключ будет называться `title_authors_pk`.

**С** Чтобы увидеть результат исполнения команды `CREATE TABLE`, изучите структуру таблицы с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.

**С** Чтобы задать вторичный ключ, обратитесь к разделу «Задание внешнего ключа с помощью ограничения `FOREIGN KEY`» далее в этой главе.

**С**

Чтобы задать столбец, который содержит уникальные значения, но не является первичным ключом, обратитесь к разделу «Присвоение уникальных значений с помощью ограничения UNIQUE» далее в этой главе.

**С**

Чтобы изменить или удалить существующее ограничение, обратитесь к разделу «Изменение таблицы с помощью команды ALTER TABLE» далее в этой главе.

**П**

Нельзя задавать более одного первичного ключа для одной таблицы. Например, вы не можете использовать подобную команду, чтобы задать сложный ключ для title\_authors:

```
CREATE TABLE title_authors(
 title_id CHAR(3) PRIMARY KEY,
 au_id CHAR(3) PRIMARY KEY,
 au_order SMALLINT NOT NULL,
 ...
);
```



СУБД MySQL требует, чтобы вы задали ограничение NOT NULL для столбцов первичного ключа, если он создается как ограничение таблицы, а не как ограничение столбцов (см. раздел «Запрет значения null с помощью ограничения NOT NULL» ранее в этой главе).

MySQL не поддерживает предложение CONSTRAINT для ограничений столбцов (но вы можете использовать его для ограничений таблиц).

Oracle воспринимает пустую строку (') как NULL (см. примечание **DBMS** в разделе «Значение null» главы 3).

## Задание внешнего ключа с помощью ограничения FOREIGN KEY

Про *внешние ключи* мы уже рассказывали в одноименном разделе главы 2. Вкратце повторим:

- внешний ключ используется для связи между двумя таблицами;
- внешний ключ – это столбец (или несколько столбцов) в таблице, значения в которой связаны со значениями в другой таблице или ссылаются на них;
- внешний ключ гарантирует, что строки в одной таблице имеют соответствующие строки в другой таблице, которая называется *родительской*;
- внешний ключ создает прямую связь с первичным ключом или группой значений в родительской таблице, поэтому его значения обязательно должны соответствовать уже существующим значениям в родительской таблице. Это правило называется *ссылочной целостностью*;
- в отличие от первичного ключа, столбцы внешнего ключа могут содержать значение null;
- в каждой таблице может быть несколько внешних ключей (или не быть вообще);
- как правило, значения внешнего ключа не являются уникальными в таблице;
- столбцы внешнего ключа в разных столбцах могут ссылаться на один столбец в родительской таблице;
- *простой внешний ключ* включает один столбец; *сложный* – несколько.

При установке ограничения внешнего ключа вам следует учитывать следующие моменты:

- простой ключ может быть как ограничением столбца, так и ограничением таблицы; сложный ключ может быть только ограничением таблицы (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);
- вы задаете внешний ключ с помощью ключевых слов FOREIGN KEY или REFERENCES в обозначении столбца в команде CREATE TABLE. Ключевые слова FOREIGN KEY и REFERENCES являются вариантами одного и того же ограничения; мы, как правило, будем использовать FOREIGN KEY;
- внешний ключ может иметь название столбца, отличное от родительского;
- тип данных внешнего ключа должен быть того же типа (или конвертируемым), что и родительский ключ (см. раздел «Преобразование типов данных с помощью функции CAST()» в главе 5);
- столбец внешнего ключа необязательно должен ссылаться только на столбец первичного ключа родительской таблицы; он может ссылаться на уникальный столбец в ней (см. раздел «Присвоение уникальных значений с помощью ограничения UNIQUE» далее в этой главе);
- таблица может включать сколько угодно внешних ключей (и даже ни одного);
- ограничения внешних ключей почти всегда имеют точные названия. Чтобы задать название ограничению, пользуйтесь предложением CONSTRAINT (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);

- ограничение на значение null для столбцов внешних ключей может быть как NULL, так и NOT NULL. Если вы не укажете это ограничение, СУБД автоматически задаст его как NULL для всех столбцов вторичных ключей (см. раздел «Запрет значения null с помощью ограничения NOT NULL» ранее в этой главе);
- внешние ключи упрощают изменение и удаление данных в таблицах, позволяя препятствовать некорректным операциям за счет контроля ссылочной целостности. Однако структура ссылок даже в средней базе данных может стать очень сложной. Неправильный дизайн базы данных способен вызвать появление медленных запросов, кольцевых ссылок, сложных операций по резервному сохранению и последующему восстановлению базы данных и ненужных каскадных удалений.

Чтобы сохранить ссылочную целостность, СУБД будет предотвращать создание неправильных строк (строк в подчиненной таблице, которые не имеют соответствующих строк в родительской таблице). Если вы будете изменять столбец внешнего ключа (который имеет ссылку на столбец PRIMARY KEY в родительской таблице) с помощью команд INSERT, UPDATE или DELETE, СУБД выполнит следующие проверки:

- при добавлении строки в подчиненную таблицу – соответствует ли новое значение внешнего ключа значению первичного ключа в родительской таблице. Если такого соответствия нет, СУБД не добавит строку;
- при изменении строки в подчиненной таблице – соответствует ли измененное значение внешнего ключа значению первичного ключа в родительской таблице. Если такого соответствия нет, СУБД не изменит строку;
- при удалении строки из подчиненной таблицы проверка ссылочной целостности не нужна;
- при добавлении строки в родительскую таблицу проверка ссылочной целостности не нужна;
- при изменении строки в родительской таблице – не соответствует ли какое-либо значение внешнего ключа значению первичного ключа в родительской таблице, которое будет изменено. Если такое соответствие существует, СУБД не изменит строку;
- при удалении строки из родительской таблицы – не соответствует ли какое-либо значение внешнего ключа значению первичного ключа в родительской таблице, которое будет удалено. Если такое соответствие существует, СУБД не удалит строку.

СУБД не выполняет проверку ссылочной целостности для строк внешнего ключа, которые имеют значение NULL.

### Задание простого внешнего ключа

Чтобы задать простой внешний ключ в качестве ограничения столбца, добавьте к описанию столбца в команде CREATE TABLE следующее ограничение столбца:

```
[CONSTRAINT constraint_name]
REFERENCES ref_table (ref_column)
```

Или

Чтобы задать простой внешний ключ в качестве ограничения таблицы, добавьте после описания столбцов в команде CREATE TABLE следующее ограничение таблицы:

```
[CONSTRAINT constraint_name]
FOREIGN KEY (key_column)
REFERENCES ref_table (ref_column)
```

*key\_column* – это название столбца внешнего ключа; *ref\_table* – название родительской таблицы, ссылку на которую



**Листинг 10.10.** Создать столбец вторичного ключа в таблице `titles` с помощью ограничения столбца

```

CREATE TABLE titles
(
 title_id CHAR(3) NOT NULL
 PRIMARY KEY,
 title_name VARCHAR(40) NOT NULL,
 type VARCHAR(10) NULL,
 pub_id CHAR(3) NOT NULL
 REFERENCES publishers(pub_id),
 pages INTEGER NULL,
 price DECIMAL(5,2) NULL,
 sales INTEGER NULL,
 pubdate DATE NULL,
 contract SMALLINT NOT NULL
);

```

**Листинг 10.11.** Задать простой вторичный ключ для таблицы `royalties` с помощью именованного ограничения таблицы

```

CREATE TABLE royalties
(
 title_id CHAR(3) NOT NULL,
 advance DECIMAL(9,2) NULL,
 royalty_rate DECIMAL(5,2) NULL,
 CONSTRAINT royalties_pk
 PRIMARY KEY (title_id),
 CONSTRAINT royalties_title_id_fk
 FOREIGN KEY (title_id)
 REFERENCES titles(title_id)
);

```

создает ограничение внешнего ключа; *ref\_column* – название столбца в *ref\_table*, который является ключом для ссылки. В таблице можно ввести несколько внешних ключей (или ни одного). Информация об общей структуре команды `CREATE TABLE` представлена в разделе «Порядок создания таблиц» ранее в этой главе. Предложение `CONSTRAINT` является опциональным, а *constraint\_name* – это имя ограничения (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе).

Листинг 10.10 использует ограничение столбца, чтобы создать внешний ключ в таблице `titles`. Здесь показан самый доступный способ создания простого внешнего ключа. После запуска этой команды СУБД проверит, существуют ли в столбце `pub_id` в `publishers` те значения, которые вы вводите в одноименный столбец в `titles`. Обратите внимание, что в столбце внешнего ключа значения `null` запрещены, поэтому необходимо ввести издательство для каждой книги.

Таблица `royalties` имеет связь с таблицей `titles`, поэтому листинг 10.11 задает столбец `title_id` как первичный и внешний ключи для связи `title_id` и `titles`. За информацией о связях обращайтесь к разделу «Связи» в главе 2.

Листинг 10.12 использует именованное ограничение таблицы, чтобы задать два внешних ключа. Здесь показан предпочтительный способ добавления внешних ключей; если вы решите впоследствии удалить или изменить ключи, можете использовать их названия (см. раздел «Изменение таблицы с помощью команды `ALTER TABLE`» далее в этой главе). Каждый столбец внешнего ключа является самостоятельным ключом, а не частью сложного ключа. Обратите внимание, что вместе внешние ключи составляют сложный первичный ключ таблицы.

## Задание сложного внешнего ключа

Добавьте после описания столбцов в команде CREATE TABLE следующее ограничение таблицы:

```
[CONSTRAINT constraint_name]
 FOREIGN KEY (key_columns)
 REFERENCES ref_table (ref_columns)
```

*key\_columns* — это список столбцов внешнего ключа, разделенных запятыми; *ref\_table* — название родительской таблицы, ссылку на которую создает ограничение внешнего ключа; *ref\_columns* — названия столбцов в *ref\_table*, которые являются ключами для ссылок. Списки *key\_columns* и *ref\_columns* должны включать одинаковое число столбцов в соответствующем порядке. В таблице можно ввести несколько внешних ключей (или ни одного). Информацию об общей структуре команды CREATE TABLE см. в разделе «Порядок создания таблиц» ранее в этой главе. Предложение CONSTRAINT является опциональным, а *constraint\_name* — это имя ограничения (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе). Информация об общей структуре команды CREATE TABLE приведена в разделе «Порядок создания таблиц» ранее в этой главе.

Наша база данных не содержит внешних ключей. Предположим, что мы создали таблицу `out_of_print`, чтобы хранить информацию о книгах каждого автора, которые вышли из печати. Таблица `title_authors` имеет сложный первичный ключ. Следующее ограничение показывает, как связать его с таблицей `out_of_print`:

```
CONSTRAINT out_of_print_fk
 FOREIGN KEY
 (title_id, au_id)
 REFERENCES
 title_authors (title_id, au_id)
```

**Листинг 10.12.** Задать простые вторичные ключи для таблицы `title_authors` с помощью именованных ограничений таблицы

```
Листинг
CREATE TABLE title_authors
(
 title_id CHAR(3) NOT NULL,
 au_id CHAR(3) NOT NULL,
 au_order SMALLINT NOT NULL,
 royalty_share DECIMAL(5,2) NOT NULL,
 CONSTRAINT title_authors_pk
 PRIMARY KEY (title_id, au_id),
 CONSTRAINT title_authors_title_id_fk
 FOREIGN KEY (title_id)
 REFERENCES titles(title_id),
 CONSTRAINT title_authors_au_id_fk
 FOREIGN KEY (au_id)
 REFERENCES authors(au_id)
);
```

**С**

Чтобы увидеть результат исполнения команды `CREATE TABLE`, изучите структуру таблицы с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.

**С**

Чтобы задать первичный ключ, обратитесь к разделу «Задание первичного ключа с помощью ограничения PRIMARY KEY» ранее в этой главе.

**С**

Чтобы изменить или удалить существующее ограничение, обратитесь к разделу «Изменение таблицы с помощью команды ALTER TABLE» далее в этой главе.

**П**

Выражение (*ref\_column*) или (*ref\_columns*) в предложении REFERENCES можно опустить, если столбец (или столбцы) является первичным ключом для *ref\_table*.

**С**

SQL позволяет указать действие, выполняемое СУБД, если вы попытаетесь изменить или удалить значение ключа (в родительской таблице), на которое указывает значение внешнего ключа. Чтобы указать действие, задайте предложение ON UPDATE или ON DELETE в ограничении FOREIGN KEY. Разные СУБД по-разному выполняют это действие; обратитесь к документации по вашей СУБД. Стандартное исполнение СУБД этих пунктов мы опишем в двух следующих примечаниях.

**П**

Ограничение FOREIGN KEY может обращаться к другому столбцу в той же таблице. Вспомните из раздела «Создание самообъединения» в главе 7, что таблица `employees` имеет ссылку на себя (эту таблицу мы создали только для иллюстрации; она не является частью базы данных). Таблица `employees` имеет три столбца: `emp_id`, `emp_name` и `boss_id`. Столбец `emp_id` — это первичный ключ, который идентифицирует сотрудника, а столбец

`boss_id` идентифицирует его начальника. Каждый начальник также является сотрудником, поэтому, чтобы быть уверенными в том, что информация о каждом начальнике при добавлении в таблицу совпадает с существующей информацией, мы создали столбец `boss_id` как вторичный ключ для `emp_id`:

```
CREATE TABLE employees
(
 emp_id CHAR(3) NOT NULL,
 emp_name CHAR(3) NOT NULL,
 boss_id CHAR(3) NULL,
 CONSTRAINT employees_pk
 PRIMARY KEY (emp_id),
 CONSTRAINT employees_fk
 FOREIGN KEY (boss_id)
 REFERENCES employees (emp_id)
);
```

**П**

Предложение ON UPDATE *action* задает действие, выполняемое СУБД, если вы измените значение ключа в строке (родительской таблицы), которое связано ссылками со вторичными ключами в других таблицах. *action* может иметь одно из четырех значений:

- CASCADE заменит значения внешних ключей в соответствии с новым значением первичного ключа;
- SET NULL заменит значения внешних ключей на NULL;
- SET DEFAULT заменит значения внешних ключей значениями по умолчанию (см. раздел «Присвоение значения по умолчанию с помощью ограничения DEFAULT» ранее в этой главе);
- NO ACTION выдаст ошибку для внешнего ключа. Эта установка задается по умолчанию.

**П**

Предложение ON DELETE *action* задает действие, которое выполнит СУБД, если вы удалите строку родительской таблицы, связанной ссылками с внешними ключами в других таблицах. *action* может иметь одно из четырех значений:

- CASCADE удалит строки, которые содержат значения внешних ключей, соответствующие удаленному первичному ключу;
- SET NULL заменит значения внешних ключей на NULL;
- SET DEFAULT заменит значения внешних ключей значениями по умолчанию (см. раздел «Присвоение значения по умолчанию с помощью ограничения DEFAULT» ранее в этой главе);
- NO ACTION выдаст ошибку для внешнего ключа. Эта установка задается по умолчанию.



Microsoft SQL Server не поддерживает тип данных DATE. Чтобы запустить листинг 10.10 в SQL Server, замените тип данных в столбце `pubdate` на `DATETIME`.

MySQL 4.0 и более ранних версий не поддерживает внешние ключи. Эти версии разрешают указывать `FOREIGN KEY` и `REFERENCES` в команде `CREATE TABLE` для совместимости со стандартом SQL, но MySQL не может поддерживать ссылочную целостность. Чтобы работать с внешними ключами, пользуйтесь типом таблиц InnoDB (это коммерческое дополнение к MySQL). Обратитесь к документации по MySQL или зайдите на Web-сайт [www.innodb.com](http://www.innodb.com).

MySQL не поддерживает предложение `CONSTRAINT` для ограничений столбцов (но вы можете использовать его для ограничений таблиц).

Oracle воспринимает пустую строку (") как NULL (см. примечание **DBMS** в разделе «Значение null» главы 3).

## Присвоение уникальных значений с помощью ограничения UNIQUE

Ограничение уникальности удостоверяет, что столбец (или несколько столбцов) не содержит повторяющихся значений. Ограничение уникальности похоже на первичный ключ, только уникальный столбец может содержать значения null и таких столбцов в таблице может быть несколько (за информацией об ограничении первичных ключей обращайтесь к разделу «Задание первичного ключа с помощью ограничения PRIMARY KEY» ранее в этой главе).

Предположим, что мы добавили столбец `isbn` (с ISBN книги) в таблицу `titles`. ISBN – это уникальный стандартизированный номер, который служит для точной идентификации книги. `titles` уже имеет первичный ключ (`title_id`), поэтому для уникальности каждого номера мы добавили к столбцу `isbn` ограничение уникальности.

При установке ограничения уникальности следует принять в расчет следующее:

- *простое ограничение уникальности* включает один столбец; *сложное* – несколько;
- значения сложного ограничения могут повторяться в одном столбце, но каждая комбинация значений должна быть уникальной;
- простое ограничение уникальности может быть ограничением столбца или таблицы; сложное ограничение всегда является ограничением таблицы (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);
- вы задаете ограничение уникальности с помощью ключевого слова `UNIQUE` в описании столбца в команде `CREATE TABLE`;

- ограничение таблицы `UNIQUE` требует ввода названия столбца (или столбцов). Ограничение столбца `UNIQUE` применяется к столбцу, для которого оно было задано;
- таблица может включать сколько угодно ограничений уникальности (в том числе ни одного);
- ограничение уникальности почти всегда имеет точное название. Чтобы задать название, пользуйтесь предложением `CONSTRAINT` (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);
- ограничение на значение null для столбцов уникальности может быть как `NULL`, так и `NOT NULL`. Если вы не укажете ограничение, СУБД автоматически задаст его как `NULL` для всех столбцов (см. раздел «Запрет значения null с помощью ограничения `NOT NULL`» ранее в этой главе).

### Задание простого ограничения уникальности

Чтобы задать простое ограничение уникальности в качестве ограничения столбца, добавьте к описанию столбца в команде `CREATE TABLE` следующее ограничение столбца:

```
[CONSTRAINT constraint_name] UNIQUE
```

Или

Чтобы задать простое ограничение уникальности в качестве ограничения таблицы, добавьте после описания столбцов в команде `CREATE TABLE` следующее ограничение таблицы:

```
[CONSTRAINT constraint_name]
UNIQUE (unique_column)
```

*unique\_column* – это название столбца, все значения в котором должны быть уникальными. В таблице можно ввести

несколько ограничений уникальности (в том числе ни одного). Информацию об общей структуре команды CREATE TABLE см. в разделе «Порядок создания таблиц» ранее в этой главе. Предложение CONSTRAINT является опциональным, а *constraint\_name* — это название ограничения (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе).

Листинги 10.13а и 10.13б демонстрируют два эквивалентных способа указать простое ограничение уникальности для таблицы titles.

Листинг 10.13а использует ограничение столбца, чтобы задать уникальный столбец. Здесь показан самый доступный способ создания простого ограничения уникальности.

Листинг 10.13б использует именованное ограничение таблицы, чтобы задать уникальный столбец. Здесь показан предпочтительный способ добавления ограничения уникальности; если впоследствии вы решите изменить или удалить его, можно будет использовать заданное имя (см. раздел «Изменение таблицы с помощью команды ALTER TABLE» далее в этой главе).

### Задание сложного ограничения уникальности

Добавьте после описания столбцов в команде CREATE TABLE следующее ограничение таблицы:

```
[CONSTRAINT constraint_name]
 UNIQUE (unique_columns)
```

*unique\_columns* — это список разделенных запятыми столбцов, для которых запрещены повторяющиеся значения. В таблице можно ввести несколько ограничений

**Листинг 10.13а.** Создать простое ограничение уникальности в столбце title\_name таблицы titles с помощью ограничения столбца

```

Листинг

CREATE TABLE titles
(
 title_id CHAR(3) PRIMARY KEY
 title_name VARCHAR(40) NOT NULL UNIQUE
 type VARCHAR(10) NULL
 pub_id CHAR(3) NOT NULL
 pages INTEGER NULL
 price DECIMAL(5,2) NULL
 sales INTEGER NULL
 pubdate DATE NULL
 contract SMALLINT NOT NULL
);

```

**Листинг 10.13б.** Создать простое ограничение уникальности в столбце title\_name таблицы titles с помощью именованного ограничения таблицы

```

Листинг

CREATE TABLE titles
(
 title_id CHAR(3) NOT NULL
 title_name VARCHAR(40) NOT NULL
 type VARCHAR(10) NULL
 pub_id CHAR(3) NOT NULL
 pages INTEGER NULL
 price DECIMAL(5,2) NULL
 sales INTEGER NULL
 pubdate DATE NULL
 contract SMALLINT NOT NULL
 CONSTRAINT titles_pk
 PRIMARY KEY (title_id),
 CONSTRAINT titles_title_name_unique
 UNIQUE (title_name)
);

```

**Листинг 10.14.** Задать сложное ограничение уникальности для столбцов `au_fname` и `au_lname` таблицы `authors` с помощью именованного ограничения таблицы

```

Листинг
TABLE authors
(
 au_id CHAR(3) NOT NULL ,
 au_fname VARCHAR(15) NOT NULL ,
 au_lname VARCHAR(15) NOT NULL ,
 phone VARCHAR(12) NULL ,
 address VARCHAR(20) NULL ,
 city VARCHAR(15) NULL ,
 state CHAR(2) NULL ,
 zip CHAR(5) NULL ,
 CONSTRAINT authors_pk
 PRIMARY KEY (au_id),
 CONSTRAINT authors_au_name_unique
 UNIQUE (au_fname, au_lname)
);

```



Microsoft SQL Server не поддерживает тип данных `DATE`. Чтобы запустить листинги 10.13а и 10.13б в SQL Server, замените тип данных в столбце `pubdate` на `DATETIME`.

MySQL не поддерживает предложение `CONSTRAINT` для ограничений столбцов (но вы можете использовать его для ограничений таблиц).

Oracle воспринимает пустую строку (") как `NULL` (см. примечание **DBMS** в разделе «Значение null» в главе 3).

SQL допускает не более одного значения `null` в уникальном столбце, для которого разрешены эти значения. Microsoft SQL Server следует стандарту, но Microsoft Access, Oracle, MySQL и PostgreSQL допускают много значений `null` в уникальных столбцах, для которых не задано ограничение `NOT NULL`.

уникальности (или ни одного). Информацию об общей структуре команды `CREATE TABLE` см. в разделе «Порядок создания таблиц» ранее в этой главе. Предложение `CONSTRAINT` является опциональным, а *constraint\_name* — это название ограничения (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе).

Листинг 10.14 задает сложное ограничение уникальности для таблицы `authors`. Это ограничение обеспечивает уникальность имени и фамилии каждого автора.

**С**

Чтобы увидеть результат исполнения команды `CREATE TABLE`, изучите структуру таблицы с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.

**П**

Столбец внешнего ключа может указывать на уникальный столбец (см. раздел «Задание внешнего ключа с помощью ограничения `FOREIGN KEY`» ранее в этой главе).

**С**

Чтобы изменить или удалить существующее ограничение, обратитесь к разделу «Изменение таблицы с помощью команды `ALTER TABLE`» далее в этой главе.

**С**

Вы можете создать уникальный индекс вместо ограничения уникальности (см. раздел «Создание индекса с помощью команды `CREATE INDEX`» в главе 11). Чтобы узнать, работает ли ваша СУБД с индексами или ограничениями, обратитесь к документации.

## Проверка значений столбца с помощью ограничения CHECK

До настоящего времени вы знали только об ограничениях по типу данных, размеру и диапазону столбца для данных, которые вы добавляете. Вы можете использовать *ограничение check* для дополнительного ограничения значений в столбце (столбцах). Обычно ограничения *check* используются для:

- проверки минимального или максимального значения. Например, не дают уровню продаж принимать значение ниже нуля;
- проверки специальных значений. Например, не дают задать для столбца *science* никакие значения, кроме *biology*, *chemistry* и *physics*;
- проверки диапазона значений. Например, проверяют, находится ли значение авторского гонорара между 2 и 20 процентами.

Ограничение *check* напоминает внешний ключ, так как все значения, нужные для ограничения, располагаются в столбце (см. раздел «Задание внешнего ключа с помощью ограничения FOREIGN KEY» ранее в этой главе). Разница заключается в том, как эти ограничения определяют допустимые значения. Ограничение внешнего ключа считывает список допустимых значений из другой таблицы, в то время как ограничение *check* определяет эти значения с помощью логического выражения. Например, это ограничение не допускает, чтобы зарплата какого-либо сотрудника была более \$50 000:

```
CHECK (salary <= 50000)
```

При установке ограничения *check* вам нужно учитывать следующие моменты:

- ограничение *check*, которое относится к одному столбцу, может быть как ограничением столбца, так и ограничением таблицы; ограничение *check*, которое относится к нескольким столбцам, всегда является ограничением таблицы (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);
- вы задаете ограничение *check* с помощью ключевого слова **CHECK** в обозначении столбца в команде **CREATE TABLE**;
- столбец может включать сколько угодно ограничений *check* (или ни одного);
- если вы создаете несколько ограничений *check* для столбца, выполняйте это осторожно, чтобы они не конфликтовали друг с другом. Не рассчитывайте на то, что СУБД самостоятельно расположит ограничения в нужном порядке или исправит все конфликты;
- ограничения *check* почти всегда имеют точные названия. Чтобы задать название ограничению, пользуйтесь предложением **CONSTRAINT** (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе);
- условием для ограничения *check* может быть любое условие **WHERE**, например сравнение (**=**, **<>**, **<=**, **>**, **>=**), **LIKE**, **BETWEEN**, **IN** или **IS NULL**. Вы можете соединять разные условия с помощью операторов **AND**, **OR** и **NOT**. За информацией об условиях обращайтесь к разделу «Фильтрация строк с помощью предложения **WHERE**» в главе 4;
- условие ограничения *check* может относиться к любому столбцу в таблице, но не может относиться к столбцам в других таблицах;



**Листинг 10.15.** Задать различные ограничения check столбца и таблицы для таблицы titles

```

CREATE TABLE titles
(
 title_id CHAR(3) NOT NULL,
 title_name VARCHAR(40) NOT NULL,
 type VARCHAR(10) NULL
 CONSTRAINT type_chk
 CHECK (type IN ('biography',
 'children', 'computer',
 'history', 'psychology')),
 pub_id CHAR(3) NOT NULL,
 pages INTEGER NULL
 → CHECK (pages > 0),
 price DECIMAL(5,2) NULL,
 sales INTEGER NULL,
 pubdate DATE NULL,
 contract SMALLINT NOT NULL,
 CONSTRAINT titles_pk
 PRIMARY KEY (title_id),
 CONSTRAINT titles_pub_id_fk
 FOREIGN KEY (pub_id)
 REFERENCES publishers(pub_id),
 CONSTRAINT title_id_chk
 CHECK (
 (SUBSTRING(title_id FROM 1 FOR 1) = 'T')
 AND
 (CAST(SUBSTRING(title_id FROM 2 FOR 2)
 AS INTEGER) BETWEEN 0 AND 99)),
 CONSTRAINT price_chk
 CHECK (price >= 0.00
 AND price < 100.00),
 CONSTRAINT sales_chk
 CHECK (sales >= 0),
 CONSTRAINT pubdate_chk
 CHECK (pubdate >= DATE '1950-01-01'),
 CONSTRAINT title_name_contract_chk
 CHECK (title_name <> ''
 AND contract >= 0),
 CONSTRAINT revenue_chk
 CHECK (price * sales >= 0.00)
);

```

- вы можете добавить ограничение check после заполнения таблицы, но лучше это сделать заранее, чтобы устранить возможные ошибки.

## Задание ограничения check

Чтобы задать ограничение check в качестве ограничения столбца или таблицы, добавьте к описанию столбца в команде CREATE TABLE следующее ограничение таблицы:

```
[CONSTRAINT constraint_name]
CHECK (condition)
```

*condition* – это логическое условие, которое проверяется СУБД, если вы изменяете содержимое таблицы с помощью команд INSERT, UPDATE или DELETE. Если в процессе изменения это условие становится истинным или неизвестным (значение равно NULL), изменение разрешается, если ложным – отменяется и выдается сообщение об ошибке. Информацию об общей структуре команды CREATE TABLE см. в разделе «Порядок создания таблиц» ранее в этой главе. Предложение CONSTRAINT является опциональным, а *constraint\_name* – это название ограничения (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе).

Листинг 10.15 показывает различные ограничения check столбца и таблицы для таблицы titles. Ограничение title\_id\_chk проверяет, что каждое значение первичного ключа принимает вид Tnn, где nn – это формат значений от 00 до 99 включительно.

**С**

Чтобы увидеть результат исполнения команды CREATE TABLE, изучите структуру таблицы с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.

С

Чтобы изменить или удалить существующее ограничение, обратитесь к разделу «Изменение таблицы с помощью команды ALTER TABLE» далее в этой главе.



Чтобы запустить листинг 10.15 в Microsoft Access, замените два ограничения столбца (для столбцов `type` и `pages`) ограничениями таблицы, поместив их после описания столбцов, первое выражение получения подстроки — `Mid(title_id, 1, 1)`; выражение `CAST — CInt (Mid(title_id, 2, 2))`; уберите из команды даты слово `DATE` и дополните ее символами `#` вместо кавычек (`#1950-01-01#`).

П

В соответствии со стандартными требованиями SQL условие не может включать функцию получения времени (`CURRENT_DATE`, `CURRENT_TIME` или `CURRENT_TIMESTAMP`), а также функцию получения имени пользователя (`CURRENT_USER`, `SESSION_USER` или `SYSTEM_USER`), но некоторые СУБД их разрешают. Microsoft Access, Microsoft SQL Server и PostgreSQL (но не Oracle) разрешают, например, следующее ограничение `check`:

```
CHECK (ship_time >= CURRENT_TIMESTAMP)
```

В Access вместо `CURRENT_TIMESTAMP` следует использовать `Now()` (см. разделы «Извлечение значений текущих даты и времени» и «Отображение информации о пользователе» в главе 5).

П

Microsoft SQL Server и Oracle позволяют создавать *пользовательские типы данных*, которые представляют собой стандартные типы данных (`CHARACTER`, `INTEGER` и т.д.) с различными ограничениями. Например, вы можете создать тип данных `marital_status`, состоящий из одного символа `CHARACTER`, который допускает только значения `S`, `M`, `W`, `D` или `NULL` (для холостых, женатых, вдовцов, разведенных или неизвестного типа). Преимущество использования такого типа данных заключается в том, что вы можете создать его один раз и затем использовать для разных таблиц, а не повторять описание ограничений. Обратитесь к документации по вашей СУБД.

Чтобы запустить листинг 10.15 в Oracle, вместо двух выражений получения подстроки используйте `SUBSTR(title_id, 1, 1)` и `SUBSTR(title_id, 2, 2)`.

MySQL 4.0 и более ранних версий не поддерживает ограничение `check`. Эти версии допускают наличие ключевого слова `check` в командах `CREATE TABLE` для совместимости со стандартом SQL, но MySQL не выполняет необходимых действий для данных ограничений. Если вы зададите ограничение `check`, это должно быть ограничение таблицы, но не столбца.

Чтобы запустить листинг 10.15 в PostgreSQL, вместо значений с плавающей точкой `0.00` и `100.00` воспользуйтесь `CAST(0.00 AS DECIMAL)` и `CAST(100.00 AS DECIMAL)` (см. раздел «Преобразование типов данных с помощью функции `CAST()`» в главе 5).

В Microsoft SQL Server вы можете задать ограничение `title_id_chk` как `CHECK (title_id LIKE '[T][0-9][0-9]')`; обращайтесь к SQL Help.

Oracle воспринимает пустую строку (`' '`) как `NULL` (см. примечание **DBMS** в разделе «Значение `null`» главы 3).

## Создание временной таблицы с помощью команды CREATE TEMPORARY TABLE

Все таблицы, которые мы создавали раньше, были постоянными (их еще называют *базовыми*). Эти таблицы всегда хранят данные, если вы не удалите их с помощью команды DROP. SQL также позволяет вам создавать *временные таблицы* для недолгого хранения данных или промежуточных результатов. Обычно временные таблицы используются для:

- хранения результатов сложного запроса. Это позволяет впоследствии использовать их для других запросов и значительно ускорять вычисления;
- создания образа (или *копии*) таблицы в определенный момент времени. Для того чтобы записать точное время, вы можете добавить столбец со значением по умолчанию CURRENT\_TIMESTAMP;
- хранения результата подзапроса. См. примечание DBMS в разделе «Создание внешних объединений с помощью команды OUTER JOIN» главы 7 (например, листинг 7.32).

СУБД автоматически удаляет временную таблицу после завершения сессии или транзакции (данные в таблице удаляются тоже). *Сессия* – это время вашего подключения к СУБД (между входом и выходом), в течение которого СУБД принимает и исполняет команды. *Транзакция* – это набор команд SQL, которые выполняются в качестве единой группы (см. главу 13).

При создании временной таблицы следует принять в расчет следующие моменты:

- два типа временной таблицы (глобальная и локальная) различаются по доступу. *Глобальная временная таблица* доступна для всех активных сессий; *локальная временная таблица* – только для сессии, которая ее создала;
- временные таблицы отвечают тем же требованиям в отношении названий таблиц, столбцов, типов данных и т.д., что и базовые таблицы;
- вы задаете временную таблицу с помощью стандартной команды CREATE TABLE, добавив перед ключевым словом TABLE параметр GLOBAL TEMPORARY или LOCAL TEMPORARY;
- изначально во временной таблице нет строк. Для изменения таблицы вы можете использовать команды INSERT, UPDATE и DELETE так же, как для базовой таблицы (см. главу 9);
- если вы создадите большую временную таблицу, можете самостоятельно удалить ее и освободить таким образом память, а не ждать, пока это сделает СУБД (см. раздел «Удаление таблицы с помощью команды DROP TABLE» далее в этой главе);
- если вы создадите временную таблицу с таким же названием, что и базовая, то она будет скрывать базовую до тех пор, пока вы ее не удалите;
- команда CREATE TEMPORARY TABLE позволяет администраторам базы данных предоставлять пользователям рабочее пространство и при этом не разрешать им работать с потенциально опасными командами CREATE TABLE, ALTER TABLE и DROP TABLE.

## Создание временной таблицы

Введите:

```
CREATE {LOCAL | GLOBAL} TEMPORARY
→ TABLE table
(
 column1 data_type1 [constraint1],
 column2 data_type2 [constraint2],
 ...
 columnN data_typeN [constraintN],
 [, table_constraints]
);
```

*table* – это название временной таблицы, которую вы создаете. Ключевое слово **LOCAL** показывает, что таблица будет локальной. **GLOBAL** означает, что таблица будет глобальной (см. листинги 10.16 и 10.17).

*column1*, *column2*, ..., *columnN* – это названия столбцов в таблице *table*; *data\_type1*, *data\_type2*, ..., *data\_typeN* задают тип данных для соответствующих столбцов.

Ограничения столбцов и таблиц для временных таблиц отличаются в зависимости от СУБД. Обратитесь к документации по вашей СУБД. Общая информация об ограничениях представлена в разделе «Основные принципы работы с ограничениями» ранее в этой главе.

**С** Чтобы увидеть результат исполнения команды **CREATE TABLE**, изучите структуру таблицы с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.

**С** Чтобы изменить временную таблицу, обратитесь к разделу «Изменение таблицы с помощью команды **ALTER TABLE**» далее в этой главе.

**С** Чтобы создать временную копию существующей таблицы, обратитесь к разделу «Создание новой таблицы на основе существующей с помощью команды **SELECT INTO**» далее в этой главе.

**Листинг 10.16.** Локальная временная таблица доступна только для вас. Она удаляется, когда вы завершите сессию СУБД

```
Листинг
CREATE LOCAL TEMPORARY TABLE editors
(
 ed_id CHAR(3) ,
 ed_fname VARCHAR(15) ,
 ed_lname VARCHAR(15) ,
 phone VARCHAR(12) ,
 pub_id CHAR(3)
);
```

**Листинг 10.17.** Глобальная временная таблица доступна для вас и других пользователей. Она удаляется, когда вы завершите сессию СУБД, и все другие задания перестают обращаться к ней

```
Листинг
CREATE GLOBAL TEMPORARY TABLE editors
(
 ed_id CHAR(3) ,
 ed_fname VARCHAR(15) ,
 ed_lname VARCHAR(15) ,
 phone VARCHAR(12) ,
 pub_id CHAR(3)
);
```



Глобальная временная таблица может использоваться для передачи данных между пользователями.



Microsoft Access не поддерживает временные таблицы.

В Microsoft SQL Server уберите строку {LOCAL | GLOBAL} TEMPORARY и укажите имя временной таблицы, добавив перед ее названием один или два символа #. Если вы обращаетесь к названию временной таблицы, необходимо добавлять один или два символа #. Чтобы создать локальную временную таблицу в SQL Server, введите:

```
CREATE TABLE #table (...);
```

Чтобы создать глобальную временную таблицу в SQL Server, введите:

```
CREATE TABLE ##table (...);
```

В Oracle обозначение временной таблицы «видно» для всех сессий, но ее данные доступны только для сессии, из которой они добавляются. Чтобы создать временную таблицу в Oracle, введите:

```
CREATE GLOBAL TEMPORARY
```

```
→ TABLE table (...);
```

MySQL поддерживает только локальные временные таблицы; пропустите ключевое слово LOCAL. Чтобы создать временную таблицу в MySQL, введите:

```
CREATE TEMPORARY TABLE table (...);
```

PostgreSQL поддерживает только локальные временные таблицы; ключевое слово LOCAL является опциональным. Ваша СУБД может поддерживать опциональное предложение ON COMMIT, которое задается стандартом SQL. Команда ON COMMIT PRESERVE ROWS сохраняет все изменения во временной таблице при исполнении COMMIT, а команда ON COMMIT DELETE ROWS удаляет таблицу после исполнения COMMIT. За информацией о команде COMMIT обращайтесь к главе 13.

СУБД по-разному работает с временными таблицами (это касается доступа, ограничений, внешних ключей (ссылочной целостности), индексов и представлений). Обратитесь к документации по вашей СУБД.

## Создание новой таблицы на основе существующей с помощью команды SELECT INTO

Команда `SELECT INTO` создает новую таблицу и заполняет ее результатами исполнения команды `SELECT`. Эта команда аналогична созданию пустой таблицы с помощью `CREATE TABLE` и заполнению ее с помощью `INSERT SELECT` (см. раздел «Вставка строк с помощью команды `INSERT`» в главе 9). Обратите внимание, что команда `SELECT INTO` экспортирует строки из существующей таблицы, в то время как команда `INSERT SELECT` импортирует строки в существующую таблицу. Обычно команда `SELECT INTO` используется для:

- архивирования отдельных строк;
- создания резервных копий данных;
- создания копии таблицы в определенный момент времени;
- быстрого копирования структуры таблиц без ее данных;
- создания данных для тестирования;
- копирования таблицы для безопасной проверки команд `INSERT`, `UPDATE` и `DELETE`.

При использовании команды `SELECT INTO` нужно учитывать следующие моменты:

- вы можете выбирать строки для новой таблицы с помощью стандартных предложений `WHERE`, `JOIN`, `GROUP BY` и `HAVING` команды `SELECT` либо с помощью любой опции `SELECT`, описанной в главах 4–8;
- эта команда добавляет строки в одну таблицу, независимо от того, сколько таблиц-источников использует команда `SELECT`;

- свойства столбцов и выражений в списке предложений команды `SELECT` определяют структуру новой таблицы;
- если вы включите в список пунктов команды `SELECT` столбец с производными (рассчитанными) данными, то значения в столбцах новой таблицы будут соответствовать значениям, которые были рассчитаны на момент выполнения команды `SELECT INTO` (см. раздел «Создание производных столбцов» в главе 5);
- название новой таблицы должно отличаться от названия существующей;
- чтобы использовать эту команду, вы должны иметь права на выполнение команды `CREATE TABLE`, полученные у администратора вашей базы данных.

### Создание новой таблицы на основе существующей

Введите:

```
SELECT columns
 INTO new_table
 FROM existing_table
 [WHERE search_condition];
```

*new\_table* – это название новой таблицы, которую вы создаете; *existing\_table* – название таблицы-источника; *columns* – список разделенных запятыми выражений или названий столбцов из *existing\_table*.

СУБД рассчитывает выражения в *columns*, чтобы определить структуру *new\_table*. Столбцы в *new\_table* создаются в порядке, заданном *columns*. Каждый столбец в *new\_table* имеет такое же название, тип данных и значение, что и соответствующий столбец в *columns*. Предложение `WHERE` определяет, какие строки из *existing\_table* будут добавлены в *new\_table*. Вы также можете

**Листинг 10.18.** Копировать структуру и данные существующей таблицы `authors` в новую таблицу `authors2`

```

SELECT *
 INTO authors2
 FROM authors;

```

**Листинг 10.19.** Скопировать только структуру (но не данные) существующей таблицы `publishers` в новую таблицу с названием `publishers2`

```

SELECT *
 INTO publishers2
 FROM publishers
 WHERE 1 = 2;

```

**Листинг 10.20.** Создать глобальную временную таблицу `titles2`, которая содержит заголовки и информацию о продажах книг, выпущенных издательством P01

```

SELECT title_name, sales
 INTO GLOBAL TEMPORARY TABLE titles2
 FROM titles
 WHERE pub_id = 'P01';

```

**Листинг 10.21.** Создать новую таблицу `author_title_names`, которая содержит имена авторов, не живущих в штатах Нью-Йорк и Калифорния, а также названия их книг

```

SELECT a.au_fname, a.au_lname, t.title_name
 INTO author_title_names
 FROM authors a
 INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
 INNER JOIN titles t
 ON ta.title_id = t.title_id
 WHERE a.state NOT IN ('CA', 'NY');

```

указать предложения `GROUP BY`, `HAVING` и `JOIN`. `search_condition` — это любое условие поиска `WHERE` (см. раздел «Фильтрация строк с помощью предложения `WHERE`» в главе 4).

Листинг 10.18 копирует структуру и данные существующей таблицы `authors` в новую таблицу `authors2`.

Листинг 10.19 использует условие `WHERE` (всегда являющееся ложным), чтобы скопировать только структуру (но не данные) существующей таблицы `publishers` в новую таблицу с названием `publishers2`.

Листинг 10.20 создает глобальную временную таблицу `titles2`, которая содержит заголовки и информацию о продажах книг, выпущенных издательством P01 (см. раздел «Создание временной таблицы с помощью команды `CREATE TEMPORARY TABLE`» ранее в этой главе).

Листинг 10.21 использует объединения, чтобы создать новую таблицу `author_title_names`, которая содержит имена авторов, не живущих в штатах Нью-Йорк и Калифорния, а также названия их книг.

**С**

Чтобы увидеть результат исполнения команды `SELECT INTO`, введите команду `SELECT`, например `SELECT * FROM new_table`. Чтобы изучить структуру таблицы, воспользуйтесь одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.

**С**

Чтобы изменить таблицу, обратитесь к разделу «Изменение таблицы с помощью команды `ALTER TABLE`» далее в этой главе.

**С**

Чтобы добавить строки в существующую таблицу, используйте команду `INSERT` (см. раздел «Вставка строк с помощью команды `INSERT`» в главе 9).

## П

Еще один распространенный способ использования команды `SELECT INTO` заключается в том, чтобы создать временную таблицу, содержащую данные на текущий день, например:

```
SELECT *
 INTO GLOBAL TEMPORARY TABLE
 todays_sales
 FROM orders
 WHERE order_date = CURRENT DATE;
```



Стандарт SQL по-другому определяет команду `SELECT INTO`. Эта команда выбирает значение в списке переменных в программе, а не создает новую таблицу. Версия команды `SELECT INTO` в ORACLE работает аналогичным образом.

Microsoft Access не поддерживает временные таблицы. Чтобы запустить листинг 10.20 в Access, удалите ключевые слова `GLOBAL TEMPORARY TABLE`. Чтобы запустить листинг 10.21 в Access, напечатайте:

```
SELECT a.au_fname, a.au_lname,
 t.title_name
 INTO author_title_names
 FROM titles t
 INNER JOIN (authors a
 INNER JOIN title_authors ta
 ON a.au_id = ta.au_id)
 ON t.title_id = ta.title_id
 WHERE a.state NOT IN ('NY', 'CA');
```

Microsoft SQL Server использует символ `##`, чтобы создать глобальную временную таблицу. Чтобы запустить листинг 10.20 в Microsoft SQL Server, замените `INTO GLOBAL TEMPORARY TABLE titles2` на `INTO ##titles2`.

В Oracle команда `SELECT INTO` используется только для того, чтобы задать переменные в PL/SQL. Команда `CREATE TABLE AS SELECT` в Oracle (еще ее называют CTAS) семантически эквивалентна `SELECT INTO`. Приведем структуру команды:

```
CREATE TABLE new_table AS
 SELECT columns
 FROM existing_table
 [WHERE search_condition];
```

Чтобы запустить листинги 10.18–10.21 в Oracle, введите (листинг 10.18):

```
CREATE TABLE authors2 AS
 SELECT *
 FROM authors;
```

(листинг 10.19):

```
CREATE TABLE publishers2 AS
 SELECT *
 FROM publishers
 WHERE 1 = 2;
```

(листинг 10.20):

```
CREATE GLOBAL TEMPORARY TABLE
 titles2 AS
 SELECT title_name, sales
 FROM titles
 WHERE 1 = pub_id = 'P01';
```

(листинг 10.21):

```
CREATE TABLE author_title_names AS
 SELECT a.au_fname, a.au_lname,
 t.title_name
 FROM authors a
 INNER JOIN title_authors ta
 ON a.au_id = ta.au_id
 INNER JOIN titles t
 ON ta.title_id = t.title_id
 WHERE a.state NOT IN ('CA', 'NY');
```

В Oracle 8i используйте в листинге 10.21 предложение `WHERE` вместо `JOIN`:

```
CREATE TABLE author_title_names AS
 SELECT a.au_fname, a.au_lname,
 t.title_name
 FROM authors a, title_authors ta,
 titles t
 WHERE a.au_id = ta.au_id
 AND ta.title_id = t.title_id
 AND a.state NOT IN ('CA', 'NY');
```

MySQL не поддерживает команду `SELECT INTO`, в листингах 10.18–10.21 используйте команды `CREATE TABLE` и `INSERT SELECT`. В PostgreSQL удалите из листинга 10.20 ключевое слово `GLOBAL`. PostgreSQL также поддерживает команду `CREATE TABLE AS`, которая является семантическим эквивалентом `SELECT INTO`.



## Изменение таблицы с помощью команды ALTER TABLE

Используйте команду ALTER TABLE, чтобы изменить таблицу путем добавления, изменения или удаления столбцов и пометок.



Использование команды ALTER TABLE существенно различается для разных СУБД. Чтобы определить, что именно вы можете изменять, и условия, при которых изменения будут разрешены, обратитесь к документации по вашей СУБД. В зависимости от функций СУБД вы можете использовать команду ALTER TABLE, чтобы:

- добавить или удалить столбец;
- изменить тип данных в столбце;
- добавить, изменить или удалить ограничения на значение null или значения по умолчанию для столбца;
- добавить, изменить или удалить ограничения таблицы или столбца (такие, как первичный ключ, внешний ключ, ограничение уникальности и ограничение check);
- переименовать столбец;
- переименовать таблицу.

### Порядок изменения таблицы

Введите:

```
ALTER TABLE table
 alter_table_action;
```

*table* – это название таблицы, которую вы изменяете; *alter\_table\_action* – описание действия для выполнения, начинается с ключевого слова ADD, ALTER или DROP.

Например, эти команды можно использовать, чтобы изменить или удалить столбцы либо ограничения:

```
ADD [COLUMN] column_data_type
DROP COLUMN column
ADD constraint_definition
DROP CONSTRAINT constraint_name
```

Листинги 10.22 и 10.23, соответственно, добавляют и удаляют из таблицы `authors` столбец `email_address`.

Если команда `ALTER TABLE` в СУБД не поддерживает нужное вам действие (например, не может удалить или переименовать столбец или ограничение), вы можете вручную создать и заполнить таблицу.

## Создание и заполнение таблицы

1. Используйте команду `CREATE TABLE`, чтобы создать новую таблицу с новыми описаниями столбцов, ограничениями столбцов и таблиц (обратитесь к разделу «Создание новой таблицы с помощью команды `CREATE TABLE`» ранее в этой главе).
2. С помощью команды `INSERT SELECT` скопируйте строки (из соответствующих столбцов) из старой таблицы в новую (см. раздел «Вставка строк с помощью команды `INSERT`» в главе 9).
3. С помощью `SELECT * FROM new_table` проверьте, что в новую таблицу добавлены нужные строки (см. раздел «Выбор столбцов с помощью предложений `SELECT` и `FROM`» в главе 4).
4. Чтобы удалить старую таблицу, используйте команду `DROP TABLE` (см. раздел «Удаление таблицы с помощью команды `DROP TABLE`» далее в этой главе).
5. Замените новое название таблицы старым (см. примечание **DBMS** в этом разделе).
6. При необходимости повторно создайте индексы (см. раздел «Создание индекса с помощью команды `CREATE INDEX`» в главе 11). Также необходимо восстановить все другие свойства, которые были удалены вместе со старой таблицей, например права доступа и триггеры.

**Листинг 10.22.** Добавить в таблицу `authors` столбец `email_address`

```
Листинг
ALTER TABLE authors
ADD email_address CHAR(25);
```

**Листинг 10.23.** Удалить из таблицы `authors` столбец `email_address`

```
Листинг
ALTER TABLE authors
DROP COLUMN email_address;
```

**С**

Чтобы увидеть результат исполнения команды ALTER TABLE, изучите структуру с помощью одной из команд, описанных в разделе «Отображение названий столбцов в таблице» главы 9.

**С**

Чтобы изменить или удалить ограничение, используйте название, которое вы ввели в предложение CONSTRAINT при его создании (см. раздел «Основные принципы работы с ограничениями» ранее в этой главе). Если вы не задали названия для ограничения, используйте название, которое было автоматически сгенерировано СУБД.

**П**

СУБД обычно накладывает на пустые таблицы меньше ограничений на модификацию, чем на заполненные. Например, если вы добавляете новый столбец в таблицу, в которой уже есть одна или несколько строк, он не может иметь ограничение NOT NULL (а новый столбец в пустой таблице может иметь любое ограничение).



В PostgreSQL команда ALTER TABLE позволяет вам добавить или переименовать столбец, но не удалить его, поэтому листинг 10.23 не будет работать в PostgreSQL. Чтобы удалить столбец, обратитесь к подразделу «Создание и заполнение таблицы» в этом разделе.

Каждая СУБД по-разному переименовывает таблицу. В Microsoft Access вам следует выделить имя таблицы в окне **Database** (База данных) и вызвать для нее контекстное меню, а затем выбрать пункт **Rename** (Переименовать). В Microsoft SQL Server выполните команду EXEC sp\_rename 'old\_name', 'new\_name'; В Oracle – команду RENAME old\_name TO new\_name; , в MySQL – команду RENAME TABLE old\_name TO new\_name; , в PostgreSQL – команду ALTER TABLE old\_name RENAME TO new\_name;.

## Удаление таблицы с помощью команды DROP TABLE

Используйте команду `DROP TABLE`, чтобы удалить таблицу из базы данных. При этом вам следует помнить, что:

- вы можете удалить базовую или временную таблицу;
- удаленная таблица не подлежит восстановлению. Вы не сможете отменить команду `DROP TABLE`;
- будет удалена структура таблицы, данные, индексы, ограничения, права доступа и т.д.;
- удаление таблицы – не то же самое, что удаление всех ее строк. Вы можете удалить из таблицы все строки с помощью команды `DELETE FROM table`, но не саму таблицу (см. раздел «Удаление строк с помощью команды `DELETE`» в главе 9);
- не будут удалены представления, которые ссылаются на таблицу (см. главу 12);
- у вас появятся сложности с внешними ключами или представлениями, которые ссылаются на удаленную таблицу, если вы не измените или не удалите их.

### Удаление таблицы

Введите:

```
DROP TABLE table
```

*table* – это название таблицы, которую вы желаете удалить (см. листинг 10.24).

Листинг 10.24. Удалить таблицу `royalties`

```
Листинг
DROP TABLE royalties;
```



Единственный способ восстановить удаленную таблицу – заново создать ее и восстановить данные, воспользовавшись последней резервной копией.



Некоторые СУБД требуют, чтобы до удаления таблицы вы удалили или изменили отдельные свойства. Например, в Microsoft SQL Server с помощью команды `DROP TABLE` нельзя удалить таблицу, на которую ссылается другая таблица посредством внешнего ключа, до тех пор, пока не будет удален сам ключ или таблица. Стандарт SQL позволяет задать опцию удаления как `RESTRICT` или `CASCADE`. `RESTRICT` (безопасное удаление) не дает удалить таблицу, на которую ссылаются представления или другие ограничения. `CASCADE` (опасное удаление) удаляет все связанные объекты вместе с таблицей. Чтобы определить, поддерживает ли ваша СУБД эту функцию, обратитесь к документации.

Вспомните из раздела «Таблицы, столбцы и строки» в главе 2, что строки сохраняются в таблице без соблюдения порядка. Это позволяет СУБД быстро выполнять такие команды, как `INSERT`, `UPDATE` и `DELETE`, но отрицательный эффект заключается в том, что поиск и сортировка данных затрудняются. Предположим, что вы запустили на выполнение следующий запрос:

```
SELECT *
FROM authors
WHERE au_lname = 'Hull';
```

Чтобы выполнить этот запрос, СУБД должна осуществить поиск по всей таблице `authors`, сравнивая значения в столбце `au_lname` каждой строки со строкой `Hull`. Выполнить поиск в таблице в небольшой базе данных просто, но таблицы в крупных базах данных могут включать миллионы строк.

В СУБД есть функция под названием *индекс* (Index), которая предназначена для того же, что и ее эквивалент в книге или библиотеке, — для быстрого поиска данных. Проще говоря, индекс представляет собой список, где каждое значение в индексированном столбце (или нескольких столбцах) хранится вместе с адресом (физическим расположением) строк, содержащих

это значение. Вместо того чтобы искать отдельные строки по всей таблице, СУБД сканирует только индекс в поисках нужных адресов. Поиск по индексу обычно выполняется гораздо быстрее, чем по таблице, но есть ряд особенностей, которые следует учитывать. Мы расскажем о них далее в этой главе.

## Создание индекса с помощью команды `CREATE INDEX`

Индексы достаточно сложны; их эффективность и воздействие на производительность системы зависят от того, как работает программа оптимизации СУБД. В этом разделе мы расскажем об общих принципах работы с индексами, но рекомендуем вам внимательно изучить документацию по вашей СУБД. Обычно индексы следует использовать для столбцов, которые:

- часто используются для поиска;
- часто используются для сортировки;
- регулярно используются для объединений.

Индексы не следует использовать для столбцов, которые:

- содержат только небольшое число значений (например, `gender` (пол) или `state` (штат));
- редко используются в запросах;
- входят в маленькую таблицу с небольшим числом строк.

При создании индекса вам следует помнить, что:

- команды SQL для работы с индексами изменяют объекты базы данных, поэтому для работы с ними вам необходимо получить разрешение администратора базы данных;
- индекс не изменяет данные, а лишь упрощает и ускоряет доступ к ним;
- в таблице может быть сколько угодно индексов (в том числе ни одного);
- в идеале вам следует создавать индексы для таблицы одновременно с самой таблицей. На практике процесс создания индекса требует времени. Только самые важные индексы создаются одновременно с таблицей. Другие индексы добавляются по мере того, как в них возникает необходимость. Большинство СУБД располагает инструментами для проверки эффективности индексов;
- не создавайте больше индексов, чем нужно. После того как вы добавляете, изменяете или удаляете строки в таблице (см. главу 9), СУБД должна обновить (а может, и реорганизовать) индекс. С учетом увеличения количества индексов возрастает время, которое СУБД тратит на обновление индексов при изменении таблицы;
- ваша СУБД будет автоматически поддерживать и использовать индексы после их создания. Ни пользователям, ни программистам не понадобится

выполнять какие-либо действия, чтобы отразить изменения во всех индексах;

- использование индексов одинаково как для пользователя, так и для программиста. Отсутствие или присутствие индекса не требует внесения изменений в структуру какой-либо команды SQL;
- индекс может ссылаться на один или несколько столбцов в таблице. Индекс, который ссылается на один столбец, является *простым*; индекс, который ссылается на несколько столбцов, — *сложным*. Столбцы в сложном индексе обязательно должны соседствовать в таблице;
- порядок следования столбцов в сложном индексе имеет значение. Сложный индекс относится только к группе столбцов, для которой он задан, но не к каждому столбцу по отдельности или к тем же столбцам в другом порядке;
- вы можете создать несколько сложных индексов, которые используют одинаковые столбцы, если укажете разные комбинации столбцов. Например, два следующих выражения задают разные комбинации для одной таблицы:  

```
CREATE INDEX au_name_idx1
ON authors (au_fname, au_lname);
CREATE INDEX au_name_idx2
ON authors (au_lname, au_fname);
```
- кроме ускорения поиска индекс может служить средством обеспечения уникальности значения. *Уникальный индекс* требует уникальности значения столбца (или комбинации значений для нескольких столбцов), на который он ссылается;

- если вы попытаетесь создать уникальный индекс для столбцов, в которых уже существуют повторяющиеся значения, ваша СУБД выдаст сообщение об ошибке и не создаст индекс;

- СУБД автоматически создаст уникальный индекс, если вы зададите первичный ключ или ограничение уникальности;
- СУБД может автоматически создавать индексы для внешних ключей, а может и не создавать их. Если она их не создает, вам следует сделать это самостоятельно, поскольку большинство объединений включает внешний ключ;
- все СУБД работают с индексами, хотя индексы не входят в основную модель (но и не нарушают ни один из ее принципов);
- так как индексы не описаны в стандарте SQL-92, команды SQL для работы с индексами различаются в разных СУБД. Стандартная команда CREATE INDEX работает одинаково во многих СУБД.

## Порядок создания индекса

Введите:

```
CREATE [UNIQUE] INDEX index
ON table (index_columns);
```

*index* — это название индекса, который вы создаете. Названия индексов в таблице должны быть уникальными. В Oracle и PostgreSQL названия индексов должны быть уникальными в базе данных.

*table* — это название таблицы, для которой будет создан индекс, а *index\_columns* — это список названий столбцов для индекса, разделенных запятыми.

Чтобы создать уникальный индекс, укажите опцию UNIQUE. Эта опция заставит СУБД выполнить поиск повторяющихся значений в *index\_columns*. Если *table* содержит строки с повторами в *index\_columns*, СУБД не создаст индекс. Если вы попробуете

вставить в *index\_columns* повторяющиеся значения или заменить существующие значения повторяющимися, СУБД выдаст ошибку и отменит действие.

Листинг 11.1 создает простой индекс с названием *pub\_id\_idx* по столбцу *pub\_id* таблицы *titles*. *pub\_id* является вторичным ключом и может использоваться для индекса, так как:

- изменения в ограничениях первичного ключа проверяются с помощью ограничений вторичного ключа в соответствующих таблицах;
- столбцы вторичного ключа часто используются в объединениях, если данные из связанных таблиц комбинируются в запросах путем сопоставления столбцов вторичного ключа в одной таблице со столбцами первичного ключа или другими столбцами с уникальными значениями в другой таблице.

Листинг 11.2 создает простой уникальный индекс с названием *title\_name\_idx* по столбцу *title\_name* таблицы *titles*. СУБД создаст этот индекс только при условии, что в столбце *title\_name* нет повторяющихся значений. Этот индекс также запрещает добавление повторяющихся значений с помощью команд INSERT и UPDATE.

Листинг 11.3 создает сложный индекс с названием *state\_city\_idx* по столбцам *title* и *city* таблицы *authors*. СУБД будет использовать этот индекс, если вы начнете сортировать строки по столбцам *state*, *city*. Этот индекс бесполезен для сортировки и поиска только по столбцу *state*, только по столбцу *city* или по столбцам *city*, *state*; для этих целей вам понадобятся отдельные индексы.

**С**

Не употребляйте термины «индекс» и «ключ» как эквивалентные (хотя в литературе вы увидите, что их так используют). Индекс представляет собой физический механизм, который используется СУБД для улучшения производительности системы. Ключ является логическим (основанным на данных) принципом, который применяется СУБД для сохранения ссылочной целостности и обновления данных в разных режимах.

**П**

Вместо уникального индекса вы можете создать ограничение уникальности; см. раздел «Присвоение уникальных значений с помощью ограничения UNIQUE» в главе 10.

**П**

Индексы представляют собой файлы, которые хранятся на диске, то есть занимают место (а иногда и много места). Но если вы будете использовать индексы правильно, то сэкономите много времени, так как избавите систему от необходимости последовательно считывать большие таблицы.

**П**

Последовательный поиск по таблице (в которой отсутствуют индексы) называется *сканированием таблицы*.

**П**

Для программистов: большинство индексов построены по принципу В-дерева. Отдельные СУБД позволяют указывать структуру данных и алгоритм, который будет использоваться при создании индекса.



Если задана опция `UNIQUE`, Microsoft SQL Server будет рассматривать значения `null` как повторяющиеся и не допустит более одного такого значения в столбцах с уникальным индексом. СУБД Microsoft Access, Oracle, MySQL и PostgreSQL допускают в таких столбцах производное количество значений `null`.

**Листинг 11.1.** Создать простой индекс в столбце `pub_id` таблицы `titles`

```
Листинг
CREATE INDEX pub_id_idx
ON titles (pub_id);
```

**Листинг 11.2.** Создать простой уникальный индекс в столбце `title_name` таблицы `titles`

```
Листинг
CREATE UNIQUE INDEX title_name_idx
ON titles (title_name);
```

**Листинг 11.3.** Создать сложный индекс в столбцах `title` и `city` таблицы `authors`

```
Листинг
CREATE INDEX state_city_idx
ON authors (state, city);
```



## Удаление индекса с помощью команды DROP INDEX

Чтобы удалить индекс, пользуйтесь командой DROP INDEX. Так как индекс логически и физически независим от данных в соответствующей таблице, вы можете удалить его в любое время, и это не повлияет на таблицу (или на другие индексы). Если вы удалите индекс, все SQL-запросы и другие приложения продолжат работу, но доступ к данным, для которых вы задавали индекс, будет более медленным.

Основные причины для удаления индекса:

- он больше не нужен;
- замедление работы СУБД при исполнении команд INSERT, UPDATE и DELETE превышает то время, которое вы выигрываете благодаря использованию индекса.

В стандарте SQL-92 не предусмотрены индексы, поэтому команды для работы с ними различаются для разных СУБД. Мы расскажем, как удалить индексы для каждой СУБД, описанной в этой книге. Если вы работаете с СУБД, не рассматриваемой здесь, обратитесь к документации, чтобы удалить индекс.

В Oracle и PostgreSQL названия индексов в базе данных должны быть уникальными, поэтому при удалении индекса вам не нужно указывать название таблицы. В СУБД Microsoft Access, Microsoft SQL Server и MySQL названия индексов в таблице должны быть уникальными, но могут повторяться в других таблицах, поэтому при удалении индекса вы должны указать и таблицу. В примерах этого раздела удаляется индекс, который был создан в листинге 11.1.

## Удаление индекса в СУБД Microsoft Access или MySQL

Введите:

```
DROP INDEX index
ON table;
```

*index* — это название индекса, который вы желаете удалить; *table* — название таблицы, к которой относится данный индекс (см. листинг 11.4а).

## Удаление индекса в СУБД Microsoft SQL Server

Введите:

```
DROP INDEX table.index;
```

*index* — это название индекса, который вы желаете удалить; *table* — название таблицы, к которой относится данный индекс (см. листинг 11.4б).

## Удаление индекса в СУБД Oracle или PostgreSQL

Введите:

```
DROP INDEX index;
```

*index* — это название индекса, который вы желаете удалить (см. листинг 11.4в).

**Листинг 11.4а.** Удалить индекс `pub_id_idx` (Microsoft Access или MySQL)

```
Листинг
DROP INDEX pub_id_idx
ON titles;
```

**Листинг 11.4б.** Удалить индекс `pub_id_idx` (Microsoft SQL Server)

```
Листинг
DROP INDEX titles.pub_id_idx;
```

**Листинг 11.4в.** Удалить индекс `pub_id_idx` (Oracle или PostgreSQL)

```
Листинг
DROP INDEX pub_id_idx;
```

# ПРЕДСТАВЛЕНИЯ

---

*Представление (view)* – это команда SELECT, считывающая таблицу, данные в которой заимствованы из других таблиц (такие таблицы называются *таблицами низшего уровня*).

При работе с представлениями вам следует помнить, что:

- таблицы низшего уровня могут быть базовыми, временными или другими представлениями;
- представление по-другому называют *виртуальной или заимствованной таблицей*. Это позволяет отличить его от базовой или временной таблицы;
- СУБД сохраняет представление только в виде команды SELECT, но не в виде набора значений. Это позволяет избежать дублирования данных;
- если в команде SELECT присутствует ссылка на представление, то оно принимает вид физической таблицы. Эта таблица существует только во время исполнения команды, а затем сразу удаляется;
- представление состоит из набора столбцов с названиями и строк с данными, поэтому вы можете применить его практически везде, где вы хотели бы использовать настоящую таблицу;

- вы можете свободно выполнять запросы через представления. В некоторых случаях можете изменять данные в представлениях. При этом изменения в данных будут применены и к таблицам низшего уровня;
- независимо от того, на какое число таблиц низшего уровня ссылается представление или как эти таблицы комбинируются, представление всегда является одной таблицей (см. примечания и советы в разделе «Таблицы, столбцы и строки» главы 2);

## Создание представления с помощью команды CREATE VIEW

Вы можете оформить представление в виде презентации, которая состоит из одного окна для показа нескольких базовых таблиц. Окно может отображать всю базовую таблицу, часть ее или комбинацию из нескольких таблиц (или их частей). Представление также может отображать данные в этих базовых таблицах посредством других представлений («окна в окнах»). SQL-программисты используют

представления, чтобы отобразить данные для пользователей в приложениях баз данных. Работа с представлениями дает следующие преимущества:

- упрощенный доступ к данным. Представления скрывают сложные данные и упрощают команды, поэтому пользователям легче выполнять действия с представлениями, чем с базовыми таблицами. Если вы создадите сложное представление (например, включающее несколько базовых таблиц, объединений и подзапросов), пользователи смогут обращаться к нему и при этом не тратить время на то, чтобы разобраться в сложных связях между таблицами (и даже не знать, что эти таблицы находятся в работе);
- автоматическое обновление. При обновлении базовой таблицы все представления, которые ссылаются на нее, автоматически отражают это изменение. Например, если вы добавите в таблицу `authors` строку с информацией о новом авторе, все представления, связанные с `authors`, автоматически изменятся. Это позволяет сэкономить дисковое пространство и избежать дублирования данных, так как без представлений СУБД пришлось бы хранить все заимствованные данные, чтобы сохранить их целостность;
- повышенную безопасность. Один из самых распространенных способов использования представлений – это сокрытие данных от пользователей путем фильтрации таблиц низшего уровня. Предположим, что таблица `employees` включает столбцы `salary` и `commission`. Если вы создадите представление, которое пропускает эти два столбца, администратор базы данных может дать пользователям разрешение только на

доступ к представлению, а не к таблице низшего уровня. При этом данные будут скрыты от тех, кому не следует их знать;

- логическую независимость данных. Базовые таблицы отображают реальное представление базы данных. Если вы используете SQL, чтобы создать программу для работы с базой данных, вам нужно, чтобы пользователи видели не всю базу данных, а виртуальные представления, которые, в свою очередь, зависят от программы. Виртуальные представления скрывают части базы данных (целые таблицы или отдельные строки и столбцы), не относящиеся к программе. Поэтому пользователи работают с виртуальным представлением, которое заимствуется из реального (в виде базовых таблиц) и при этом не зависит от него.

Виртуальное представление делает программу независимой от логических изменений в структуре базы данных. Предположим, что многие программы работают с таблицей `titles`. Книги время от времени отправляются в печать, поэтому дизайнер базы данных принял решение уменьшить нагрузку на систему путем выделения книг, выходящих из печати, в отдельную таблицу. Он разделил таблицу `titles` на две: `in_print_titles` и `out_of_print_titles`. При этом все программы перестают правильно работать, так как они обращаются к таблице `titles`, теперь не существующей. Но если эти программы будут обращаться к представлению `titles`, а не к настоящей таблице, вы сможете изменить его таким образом, чтобы оно выглядело в виде группы таблиц `in_print_titles` и `out_of_print_titles` (см. раздел «Комбинирование строк с помощью оператора `UNION`» в главе 7). Программы воспримут

две новые таблицы как единое целое и продолжат работу с ней, как будто не было никакого разделения (нужно отметить, что вы не можете использовать представления, чтобы обезопасить программы от всех изменений. Например, представления не могут компенсировать удаленные таблицы или столбцы).

При создании представлений необходимо учитывать следующие факторы:

- команды SQL для работы с представлениями изменяют объекты и данные базы данных, поэтому для их использования следует получить разрешение администратора вашей базы данных;
- названия представлениям даются по тем же правилам, что и названия таблицам;
- названия представлений в базе данных должны быть уникальными (они не могут иметь такое же название, как другая таблица или представление);
- столбцы в представлениях получают названия столбцов по умолчанию, заданные для таблиц низшего уровня. С помощью предложения AS вы можете указать для столбцов представления другие названия; см. раздел «Создание псевдонимов столбцов с помощью предложения AS» в главе 4;
- если столбец в представлении будет иметь такое же название, как другой столбец в нем, вам придется присвоить ему другое название (так как описание представления включает объединение, а столбцы из нескольких разных таблиц низшего уровня имеют одинаковое название);
- столбец в представлении может быть простой ссылкой на столбец, буквенным обозначением или выражением, которое включает выражения или функции;
- если в некоторых СУБД столбец заимствуется из арифметического выражения, встроенной функции или буквенного значения, вам придется точно указать название столбца в представлении;
- столбец в представлении получает тот же тип данных, что и столбец или выражение, из которого он заимствуется;
- не существует ограничений по количеству представлений, которые вы можете создать. Как правило, вы будете создавать представления для групп данных, с которыми работает большое число пользователей;
- некоторые СУБД не разрешают создавать представления для временных таблиц;
- представления можно задать с помощью практически любой команды SELECT, хотя использование предложения ORDER BY обычно запрещено;
- можно располагать представления на разных уровнях, то есть команда SELECT, описывающая представление, может считывать данные из другого представления. Все представления на разных уровнях должны обращаться к базовым таблицам (иначе вы ничего не увидите). Максимальное количество уровней для представлений различается для разных СУБД;
- можно использовать представления для упрощения сложных запросов. Сохранив запрос, который выполняет сложные расчеты, в виде представления, вы сможете повторять расчеты при очередном обращении к нему;
- представление может содержать запрос, который нельзя выразить другим способом. Например, вы можете создать представление, которое объединяет сгруппированное представление с базовой таблицей или соединяет представление, сформированное с помощью команды UNION, с базовой таблицей;

- описание представления не может ссылаться на себя, поскольку оно еще не создано;
- представления могут отображать данные в другом формате, чем в таблицах низшего уровня;
- в отличие от базовой таблицы, представление не разрешает использование индексов и ограничений;
- когда вы создаете представление с помощью команды `SELECT *`, SQL конвертирует знак «\*» в список всех столбцов. Это преобразование выполняется только один раз в процессе создания представления (а не при исполнении команды), поэтому при добавлении столбца в таблицу низшего уровня (с помощью команды `ALTER TABLE`) описание представления не изменится и его надо будет обновить вручную;
- так как представления не хранят данные, СУБД должна выполнять формирующую их команду каждый раз, когда вы на них ссылаетесь. Сложные представления (особенно с несколькими уровнями) могут существенно повлиять на производительность.

## Создание представления

Введите:

```
CREATE VIEW view [(view_columns)]
AS select_statement;
```

*view* — это название представления, которое вы желаете создать. Это название в базе данных должно быть уникальным.

*view\_columns* — это опциональный список разделенных запятыми названий для столбцов в представлении. Количество столбцов в *view\_columns* должно соответствовать количеству столбцов в предложении `SELECT` команды *select\_statement* (если вы

укажете название хотя бы одного столбца, вам придется перечислить и остальные столбцы). Следует указывать *view\_columns*, если столбец в *select\_statement* заимствуется из арифметического выражения, функции или буквенного обозначения; если несколько столбцов в представлении получают одинаковое название (обычно из-за использования объединений); либо для того, чтобы присвоить столбцу название, отличное от названия столбца, из которого он заимствуется. Если вы пропускаете список *view\_columns*, представление заимствует названия столбцов из команды *select\_statement*. Вы также можете задавать названия столбцов в *select\_statement* с помощью предложений `AS`. Название каждого столбца в представлении должно быть уникальным.

*select\_statement* — это команда `SELECT`, идентифицирующая столбцы и строки в таблице или таблицах, на которых формируется представление. *select\_statement* может быть достаточно сложной командой и включать несколько таблиц или других представлений. Обычно использование предложения `ORDER BY` запрещено. За дополнительной информацией о команде `SELECT` обращайтесь к главам 4–8. За информацией о запретах разных СУБД касательно использования команды `SELECT` для создания представлений обращайтесь к документации по вашей СУБД (см. листинги 12.1–12.5).

**С** Чтобы выбирать строки и столбцы при запросе по представлению, обратитесь к разделу «Считывание данных из представления» далее в этой главе.

**С** Чтобы изменить значения базовой таблицы через представление, обратитесь к разделу «Изменение данных через представление» далее в этой главе.

**Листинг 12.1.** Создать представление, которое скрывает личную информацию об авторах (номера телефонов и адреса)

```

CREATE VIEW au_names
AS
SELECT au_id, au_fname, au_lname
FROM authors;

```

**Листинг 12.2.** Создать представление, которое отображает список всех авторов, проживающих в городах, где расположены издательства. Обратите внимание, что мы используем в представлении названия столбцов `au_city` и `pub_city`. Мы переименовали эти столбцы, чтобы избежать затруднений, которые возникли бы, если бы оба столбца заимствовали название `city` из таблиц низшего уровня

```

CREATE VIEW cities
(au_id, au_city, pub_id, pub_city)
AS
SELECT a.au_id, a.city, p.pub_id, p.city
FROM authors a
INNER JOIN publishers p
ON a.city = p.city;

```

**Листинг 12.3.** Создать представление, которое отображает список прибыли (= цена × продажи), сгруппированный по типу книг для издательства. Последующее использование запросов к этому представлению будет простым, так как мы задали название для результата арифметического выражения, а не позволили СУБД присвоить название по умолчанию

```

CREATE VIEW revenues
(Publisher, BookType, Revenue)
AS
SELECT pub_id, type, SUM(price * sales)
FROM titles
GROUP BY pub_id, type;

```

**С**

Чтобы удалить представление, обратитесь к разделу «Удаление представления с помощью команды DROP VIEW» далее в этой главе.

**П**

Нельзя использовать команду SELECT INTO с представлением; см. раздел «Создание новой таблицы на основе существующей с помощью команды SELECT INTO» в главе 10.

**П**

Вы не можете создавать временные представления. Представления и временные таблицы отличаются по длительности существования. Представление существует, пока выполняется команда SQL, обращающаяся к нему; временная таблица существует на протяжении сессии. См. раздел «Создание временной таблицы с помощью команды CREATE TEMPORARY TABLE» в главе 10.

**С**

SQL не имеет команды ALTER VIEW. Если таблицы низшего уровня или представления были изменены после создания, удалите их и создайте заново (Microsoft SQL Server и Oracle поддерживают команду ALTER VIEW).



Если вы запустите команду CREATE VIEW в Microsoft Access, представление отобразится в виде объекта запроса в окне **Database** (База данных). Чтобы запустить листинг 12.4 в Access, замените все символы «||» знаком «+» (см. примечание **DBMS** в разделе «Объединение строк с помощью оператора ||» главы 5). Чтобы запустить листинг 12.5 в Access, введите:

```

CREATE VIEW au_titles
(LastName, Title)
AS
SELECT an.au_lname, t.title_name
FROM au_names an
INNER JOIN (titles t
INNER JOIN title_authors ta
ON t.title_id = ta.title_id)
ON an.au_id = ta.au_id
WHERE an.au_id IN ('A02', 'A05');

```

Microsoft SQL Server не поддерживает точку с запятой после команды CREATE VIEW (что странно). Чтобы запустить листинги 12.1–12.5 в SQL Server, уберите точку с запятой из каждой команды. Кроме того, чтобы запустить листинг 12.4 в SQL Server, замените все символы «||» знаком «+», а все функции TRIM(x) – выражением LTRIM (RTRIM(x)) (см. примечания DBMS в разделе «Объединение строк с помощью оператора ||» и «Удаление пробелов с помощью функции TRIM()» главы 5).

Oracle 9i запустит листинги 12.2 и 12.5 без изменений. Чтобы запустить листинги 12.2 и 12.5 в Oracle 8i, используйте синтаксис WHERE вместо JOIN.

Введите (листинг 12.2):

```
CREATE VIEW cities
(au_id, au_city, pub_id, pub_city)
AS
SELECT a.au_id, a.city,
 p.pub_id, p.city
FROM authors a, publishers p
WHERE a.city = p.city;
```

и (листинг 12.5)

```
CREATE VIEW au_titles
(LastName, Title)
AS
SELECT a.au_lname, t.title_name,
FROM title_authors ta,
 au_names an, titles t
WHERE ta.au_id = an.au_id
 AND t.title_id = ta.title_id
 AND an.au_id in ('A02', 'A05');
```

СУБД MySQL 4.0 и более ранних версий не поддерживает представления и не сможет запустить листинги 12.1–12.5. Чтобы скрыть данные, с помощью системы привилегий MySQL запретите доступ к столбцам.

SQL позволяет задать опциональное предложение WITH [CASCADED | LOCAL] CHECK OPTION при создании представления. Это предложение применяется только к изменяемым представлениям и позволяет добавлять, заменять или удалять только те данные, которые считываются в представлении. См. раздел «Изменение данных через представление» далее в этой главе.

**Листинг 12.4.** Создать представление, которое упрощает распечатку почтовых адресов авторов. Обратите внимание, что мы задали названия столбцов в предложении SELECT, а не в предложении CREATE VIEW

```
Листинг
CREATE VIEW mailing_labels
AS
SELECT
 TRIM(au_fname || ' ' || au_lname)
 AS "address1",
 TRIM(address)
 AS "address2",
 TRIM(city) || ', ' || TRIM(state) ||
 → ' ' || TRIM(zip)
 AS "address3"
FROM authors;
```

**Листинг 12.5.** Создать представление, которое отображает список фамилий авторов A02 и A05, а также книг, написанных ими (в том числе в соавторстве). Обратите внимание, что эта команда использует несколько уровней представлений – ссылается на представление au\_names, созданное в листинге 12.1

```
Листинг
CREATE VIEW au_titles (LastName, Title)
AS
SELECT an.au_lname, t.title_name
FROM title_authors ta
INNER JOIN au_names an
ON ta.au_id = an.au_id
INNER JOIN titles t
ON t.title_id = ta.title_id
WHERE an.au_id in ('A02', 'A05');
```



Например, если в представлении показаны только авторы из штата Нью-Йорк, вы не сможете в этом режиме добавить, изменить или удалить информацию о тех авторах, которые не живут в штате Нью-Йорк. Опции `CASCADED` и `LOCAL` применяются только к многоуровневым представлениям. `CASCADED` проверяет текущее представление и все представления, на которые оно ссылается. `LOCAL` проверяет только текущее представление, даже если оно ссылается на другие. Microsoft SQL Server и Oracle поддерживают `CHECK OPTION` (обратитесь к документации).

**Листинг 12.6.** Отобразить все строки и столбцы из представления `au_titles`. Результат исполнения показан на рис. 12.1




```
SELECT *
FROM au_titles;
```

| LastName  | Title                         |
|-----------|-------------------------------|
| Kells     | Ask Your System Administrator |
| Heydemark | How About Never?              |
| Heydemark | I Blame My Mother             |
| Heydemark | Not Without My Faberge Egg    |
| Heydemark | Spontaneous, not Annoying     |

**Рис. 12.1.** Результат листинга 12.6

**Листинг 12.7.** Отобразить список городов в представлении `cities`. Результат исполнения показан на рис. 12.2



```
SELECT DISTINCT au_city
FROM cities;
```

| au_city       |
|---------------|
| New York      |
| San Francisco |

**Рис. 12.2.** Результат листинга 12.7

## Считывание данных из представления

При создании представления на экране не отображается ничего. Команда `CREATE VIEW` всего лишь дает указание СУБД сохранить представление в виде команды `SELECT`. Чтобы увидеть данные в представлении, надо выполнить команду `SELECT` точно так же, как вы запрашиваете данные из обычной таблицы. Вы можете:

- с помощью предложения `SELECT` изменить порядок отображенных столбцов;
- с помощью операторов и функций выполнять расчеты;
- изменять заголовки столбцов с помощью `AS`;
- фильтровать строки с помощью `WHERE`;
- группировать строки с помощью `GROUP BY`;
- фильтровать сгруппированные строки с помощью `HAVING`;
- объединять представления с другими представлениями или с таблицами посредством `JOIN`;
- сортировать результат с помощью `ORDER BY`.

## Считывание данных из представления

Введите:

```
SELECT columns
FROM view
[JOIN joins]
[WHERE search_condition]
[GROUP BY group_columns]
[HAVING search_condition]
[ORDER BY sort_columns];
```

`view` — это название представления для запроса. Предложения команды работают с представлениями так же, как с таблицами.

За информацией о предложениях SELECT, FROM, ORDER BY и WHERE обратитесь к главе 4; за информацией о GROUP BY и HAVING — к главе 6; за информацией о JOIN — к главе 7.

Листинги 12.6–12.11 и рис. 12.1–12.6 показывают, как считывать данные из представлений, созданных листингами 12.1–12.5 в разделе «Создание представления с помощью команды CREATE VIEW».

C

Чтобы изменить значения базовой таблицы через представление, обратитесь к разделу «Изменение данных через представление» далее в этой главе.

C

Чтобы удалить представление, обратитесь к разделу «Удаление представления с помощью команды DROP VIEW» далее в этой главе.



Чтобы запустить листинг 12.9 в Microsoft Access, возьмите названия столбцов представления в двойные кавычки и скобки:

```
SELECT ["address3"]
FROM mailing_labels
WHERE ["address1"] LIKE '%Kell%';
```

Чтобы запустить листинг 12.9 в Oracle, возьмите названия столбцов представления в двойные кавычки:

```
SELECT "address3"
FROM mailing_labels
WHERE "address1" LIKE '%Kell%';
```

MySQL 4.0 и более ранних версий не поддерживает представления и не сможет запустить листинги 12.6–12.11.

**Листинг 12.8.** Отобразить список типов книг, доход от продаж которых превышает 1 миллион долларов США. Результат исполнения показан на рис. 12.3

```

Листинг
SELECT BookType,
 AVG(Revenue) AS "AVG(Revenue)"
FROM revenues
GROUP BY BookType
HAVING AVG(Revenue) > 1000000;
```

| BookType   | AVG(Revnuue) |
|------------|--------------|
| -----      | -----        |
| biographi  | 18727318.50  |
| computer   | 1025396.65   |
| psychology | 2320933.76   |

**Рис. 12.3.** Результат листинга 12.8

**Листинг 12.9.** Отобразить третью строку почтового адреса каждого автора, в имени которого есть слово Kell. Результат исполнения показан на рис. 12.4

```

Листинг
SELECT address3
FROM mailing_labels
WHERE address1 LIKE '%Kell%';
```

| address3            |
|---------------------|
| -----               |
| New York, NY 10014  |
| Palo Alto, CA 94305 |

**Рис. 12.4.** Результат листинга 12.9

**Листинг 12.10.** Отобразить имена всех авторов, которые написали в соавторстве хотя бы одну книгу. Результат исполнения показан на рис. 12.5

```

SELECT DISTINCT an.au_fname, an.au_lname
FROM au_names an
INNER JOIN title_authors ta
ON an.au_id = ta.au_id
WHERE ta.au_order > 1;

```

| au_fname | au_lname |
|----------|----------|
| -----    | -----    |
| Hallie   | Hull     |
| Klee     | Hull     |

**Рис. 12.5.** Результат листинга 12.10

**Листинг 12.11.** Отобразить список авторов, которые живут в Калифорнии. Результат исполнения показан на рис. 12.6

```

SELECT au_fname, au_lname
FROM au_names
WHERE state = 'CA';

```

ERROR: Invalid column name 'state'.

**Рис. 12.6.** Результат листинга 12.11

## Изменение данных через представление

*Изменяемое представление* – это такое представление, к которому вы можете применять команды INSERT, UPDATE и DELETE, чтобы преобразовать данные в таблицах низшего уровня. Любые изменения, вносимые в модифицируемое представление, всегда применяются к базовым таблицам. Команды INSERT, UPDATE и DELETE работают с представлениями так же, как с таблицами (см. главу 9).

*Неизменяемое представление* не поддерживает команды INSERT, UPDATE и DELETE, поскольку изменения не будут внесены в базовые таблицы. Такие представления предназначены только для чтения. Чтобы изменить данные, которые отображаются в неизменяемом представлении, вам следует напрямую изменить таблицы низшего уровня.

Каждая строка в изменяемом представлении связана с одной строкой в базовой таблице низшего уровня. Для того чтобы изменять данные в представлении, команда SELECT, которая задает представление, должна отвечать следующим требованиям:

- представление должно быть задано для одной таблицы (без объединений);
- команда SELECT не может сопровождаться предложениями GROUP BY и HAVING;
- команда SELECT DISTINCT запрещена;
- запросы UNION, INTERSECT и EXCEPT запрещены;
- предложение SELECT может задавать только ссылки на простые столбцы, но не на столбцы с расчетами, функциями и т.д.;
- если таблица низшего уровня является представлением, оно тоже должно быть изменяемым.

Запреты, относящиеся к изменениям, строги, но служат для обеспечения безопасности. Некоторые СУБД упрощают эти запреты, поскольку существует много других видов изменяемых представлений; СУБД может расширить набор изменяемых представлений с помощью изучения ограничений ссылочной целостности в базе данных. Помимо поддержки вышеописанных изменяемых представлений ваша СУБД может допускать создание изменяемых представлений, базирующихся на следующих запросах:

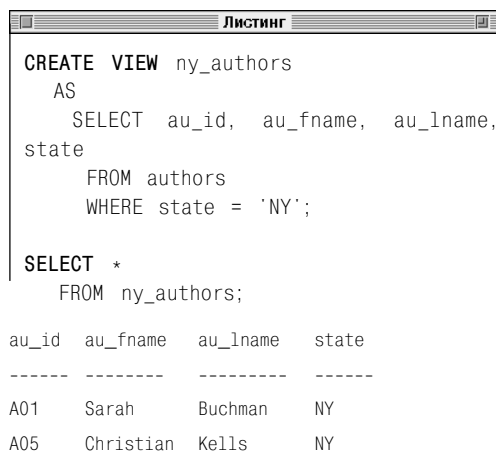
- внутренних объединениях «один-к-одному»;
- внешних объединениях «один-к-одному»;
- внутренних объединениях «один-ко-многим»;
- внешних объединениях «один-ко-многим»;
- объединениях «многие-ко-многим»;
- запросах UNION и EXCEPT.

В этом разделе мы приведем пример изменяемого представления, которое ссылается только на одну таблицу низшего уровня (в соответствии со стандартом SQL). За информацией о том, поддерживает ли ваша СУБД изменяемые представления, которые ссылаются на несколько таблиц низшего уровня, и если да, то каким образом они влияют на эти таблицы, обратитесь к документации по вашей СУБД.

### Добавление строки в представлении

Рассмотрим представление `ny_authors`, которое состоит из информации об авторах, проживающих в штате Нью-Йорк (см. листинг 12.12 и рис. 12.7). `ny_authors` ссылается только на базовую таблицу `authors`.

**Листинг 12.12.** Создать и отобразить представление `ny_authors`, которое состоит из информации только об авторах, проживающих в штате Нью-Йорк. Результат исполнения листинга показан на рис. 12.7



**Рис. 12.7.** Результат листинга 12.12: режим просмотра `ny_authors`

**Листинг 12.13.** Добавить новую строку в представление `ny_authors`

```

Листинг
INSERT INTO ny_authors
VALUES('A08','Don','Dawson','NY');
```

**Листинг 12.14.** Добавить новую строку в представление `ny_authors`. СУБД отменит вставку, если при создании представления вы использовали опцию `WITH CHECK OPTION`

```

Листинг
INSERT INTO ny_authors
VALUES('A09','Jill','LeFlore','CA');
```

Листинг 12.13 добавляет новую строку в представление. СУБД вставляет новую строку в таблицу `authors`. Строка содержит значение `A08` в столбце `au_id`, `Don` в `au_fname`, `Dawson` в `au_lname` и `NY` в `state`. Другие столбцы в строке (`phone`, `address`, `city` и `zip`) задаются как `NULL` (или значения по умолчанию, если есть ограничение `DEFAULT`).

Листинг 12.14, как и листинг 12.13, добавляет новую строку в представление. Однако новый автор из штата Калифорния, а не из Нью-Йорка, что делает ложным условие `WHERE` в описании представления. Добавит СУБД новую строку или отменит действие? Ответ на вопрос зависит от того, как создавалось представление. В нашем примере строка будет добавлена, так как команда `CREATE VIEW` (см. листинг 12.12) не имеет предложения `WITH CHECK OPTION`, поэтому СУБД не должна сохранять соответствие с начальной установкой представления. За информацией о предложении `WITH CHECK OPTION` обратитесь к примечанию **DBMS** в разделе «Создание представления с помощью команды `CREATE VIEW`» этой главы. СУБД отменила бы вставку строки, если бы представление `ny_authors` было задано следующим образом:

```

CREATE VIEW ny_authors
AS
SELECT au_id, au_fname, au_lname,
 state
FROM authors
WHERE state = 'NY'
WITH CHECK OPTION;
```

## Изменение строки в представлении

Листинг 12.15 изменяет существующую строку в представлении. СУБД преобразует строку автора `A01` в таблице `authors` путем замены Сары Бухман (`Sarah Buchman`)

на Ясмин Хаукамли (Yasmin Howcomely). Значения в других столбцах этой строки (au\_id, phone, address, city, state и zip) не изменяются.

Предположим, что листинг 12.15 имеет следующий вид:

```
UPDATE ny_authors
SET au_fname = 'Yasmin',
 au_lname = 'Howcomely',
 state = 'CA'
WHERE au_id = 'A01';
```

Эта команда встречается с той же проблемой, что и листинг 12.14: при внесении изменения строка Yasmin больше не будет соответствовать условиям, заданным для представления. СУБД примет или отвергнет изменения в зависимости от того, была ли задана опция WITH CHECK OPTION при создании представления. Если она была задана, строка не может быть изменена.

## Удаление строки в представлении

Листинг 12.16 удаляет строку в представлении. СУБД удалит строку с информацией об авторе A05 из таблицы authors (это касается всех столбцов в строке, а не только тех, которые отображены в представлении). В свою очередь, строка будет удалена из самого представления ny\_authors.

Разумеется, при изменении представления могут возникнуть нарушения в системе ссылок. СУБД отменит удаление, если при удалении будет нарушено ограничение, сохраняющее ссылочную целостность. См. раздел «Задание внешнего ключа с помощью ограничения FOREIGN KEY» в главе 10. Удаление строки будет выполнено только при условии, что ни одно из ограничений FOREIGN KEY в связанных таблицах не будет нарушено. Некоторые изменения могут быть выполнены с помощью опции

**Листинг 12.15.** Изменить существующую строку в представлении ny\_authors

```
Листинг
UPDATE ny_authors
SET au_fname = 'Yasmin',
 au_lname = 'Howcomely'
WHERE au_id = 'A01';
```

**Листинг 12.16.** Удалить строку в представлении ny\_authors

```
Листинг
DELETE FROM ny_authors
WHERE au_id = 'A05';
```

CASCADE (если она была задана) в ограничении FOREIGN KEY, а не с помощью описания представления.

Например, в листинге 12.16 СУБД отменит изменения, если предварительно не изменить или не удалить значения внешнего ключа в таблице `title_authors`, которые указывают на автора A05 в `authors`.

**С** Чтобы удалить представление, обратитесь к разделу «Удаление представления с помощью команды `DROP VIEW`» далее в этой главе.

**П** Изменяемое представление должно содержать ключ, с помощью которого базовая таблица проверяет соответствие каждой строки в представлении одной строке в базовой таблице.

**П** Любой столбец, который не входит в представление, должен иметь разрешение на содержание значения `null` или ограничение `DEFAULT` в базовой таблице, чтобы СУБД могла создать целую строку для добавления.

**П** Измененные значения должны соответствовать установкам столбцов базовой таблицы, таким как тип данных, возможность принимать значение `null` и другие ограничения.

**П** Некоторые столбцы, получаемые как результат арифметических выражений, являются (теоретически) изменяемыми. Например, в представлении с вычисляемым столбцом `bonus = 0.1 * salary` вы можете попробовать изменить значение `bonus` и использовать обратную функцию (`bonus/0.1`), чтобы изменить значение столбца `salary` в базовой таблице. Однако ваши усилия пропадут даром, так как SQL не будет реализовывать изменения в вычисляемых столбцах.

**П** При усложнении изменяемых представлений вы заметите, что одно действие, выполненное с представлением, может повлечь за собой другие действия. Например, команда `UPDATE` может потребовать добавления новых строк в базовую таблицу с помощью команды `INSERT`.



Чтобы запустить листинг 12.12 в Microsoft SQL Server, уберите точку с запятой из команды `CREATE VIEW` и запустите две команды по отдельности.

MySQL 4.0 и более ранних версий не поддерживает представления и не запустит листинги 12.12–12.16.

Чтобы допустить работу с представлением, следует использовать систему правил PostgreSQL. Команда `CREATE RULE`, которая задает правила, является расширением PostgreSQL, а не частью ANSI SQL (обратитесь к документации по СУБД PostgreSQL).

Чтобы узнать, как СУБД работает с представлениями для столбцов, тип данных в которых автоматически создает идентификатор строк, обратитесь к документации. См. примечание **DBMS** в разделе «Первичные ключи» главы 2.

## Удаление представления с помощью команды DROP VIEW

Чтобы удалить представление, пользуйтесь командой `DROP VIEW`. Так как представление физически независимо от таблиц низшего уровня, вы можете удалить его в любое время и это не повлияет на сами таблицы. Все программы SQL, приложения и другие представления, которые ссылаются на удаленное представление, перестанут нормально функционировать.

### Удаление представления

Введите:

```
DROP VIEW view;
```

*view* — это название представления, которое вы желаете удалить (см. листинг 12.17).

**П**

При удалении представления другие представления, которые ссылаются на него, удалены не будут. Поэтому вам придется по отдельности удалять каждое представление с помощью команды `DROP VIEW`; см. раздел «Удаление таблицы с помощью команды `DROP TABLE`» в главе 10.

**С**

В SQL нет команды `ALTER VIEW`. Если вы изменили таблицу низшего уровня или представление после его создания, следует удалить его и создать заново (Microsoft SQL Server и Oracle поддерживают команду `ALTER VIEW`).



MySQL 4.0 и более ранних версий не поддерживает представления и не сможет запустить листинг 12.17.

Листинг 12.17. Удалить представление `ny_authors`

```
Листинг
DROP VIEW ny_authors;
```



# ТРАНЗАКЦИИ

---

## 13

```
UPDATE saving_accounts
 SET balance = balance - 500.00
 WHERE account_number = 1009;
```

```
UPDATE checking_accounts
 SET balance = balance + 500.00
 WHERE account_number = 6482;
```

**Рис. 13.1.** Если клиент банка переводит деньги со сберегательного счета на чековый, нужно исполнить две команды SQL

*Транзакция* (Transaction) – это последовательность из одной или нескольких команд SQL, которые исполняются в виде единого логического целого. СУБД рассматривает транзакцию как неделимую группу и исполняет либо все ее команды, либо ни одной.

Давайте проиллюстрируем важность транзакций на примере. Предположим, что клиент переводит 500 долларов США со своего сберегательного счета на свой чековый счет. Эта операция состоит из двух отдельных действий, которые выполняются последовательно:

1. Уменьшить сберегательный счет на 500 долларов США.
2. Увеличить чековый счет на 500 долларов США.

На рис. 13.1 показаны две команды SQL для данной транзакции. Представим теперь, что СУБД откажет (по причине сбоя питания, нарушения в работе системы, неисправности аппаратных средств) после исполнения первой команды, но до того, как будет исполнена вторая команда. Баланс на счетах нарушится, а вы не будете об этом знать. Это может привести к очень серьезным и неприятным последствиям.

Чтобы избежать этого, с помощью транзакции следует гарантировать исполнение двух команд SQL, что позволит сохранить правильный баланс на счетах. Если что-то мешает исполнению одной из команд транзакции, СУБД отменяет все команды транзакции, выполненные до этого. При отсутствии ошибки все команды будут исполнены.

## Выполнение транзакций

Чтобы изучить принципы выполнения транзакций, следует иметь представление о некоторых понятиях:

- *выполнение* (Commit). Выполнение транзакции реализует в базе данных все изменения, произошедшие с начала исполнения транзакции. После исполнения операции все изменения будут видны другим пользователям и останутся в базе данных даже при сбое системы;
- *откат* (Roll back). Откат транзакции отменяет все изменения, которые возникли в результате исполнения ее команд. После отката транзакции данные сохраняют свой начальный вид, как будто она никогда не была исполнена;
- *журнал транзакций*. Файл журнала транзакций (или просто *журнал*) – это запись всех изменений, которые были внесены в базу данных посредством транзакции. В журнал транзакций записываются начало каждой транзакции, изменения, внесенные в данные, а также информация, с помощью которой можно отменить или повторить изменения (если это понадобится впоследствии). При исполнении транзакций в базе данных журнал постоянно увеличивается.

Хотя СУБД поддерживает физическую целостность каждой транзакции, вы должны начинать и завершать транзакции таким образом, чтобы сохранять логическую целостность данных, в соответствии с требованиями решаемой задачи. Операция должна состоять только из тех команд SQL, которые необходимы для выполнения одного изменения (не больше и не меньше). Перед транзакцией и после нее данные во всех связанных таблицах должны быть в рабочем состоянии.

При разработке и исполнении транзакций необходимо учитывать следующие факторы:

- команды SQL, работающие в транзакциях, изменяют данные, поэтому вам придется получить разрешение администратора базы данных, чтобы запустить их;
- обработка транзакций относится только к командам INSERT, UPDATE и DELETE (см. главу 9). Вы не сможете отменить результат выполнения команд SELECT, CREATE, ALTER и DROP, если включите их в транзакцию;
- каждая команда INSERT, UPDATE и DELETE должна выполняться как часть транзакции;
- выполненная транзакция становится постоянной. Это значит, что все изменения сохраняются даже в случае сбоя системы;
- система восстановления данных СУБД зависит от транзакции. Когда вы включаете СУБД после сбоя, она проверяет журнал транзакций, чтобы выяснить, были ли они выполнены. Если СУБД найдет незавершенные транзакции, она отменит их на основании журнала. Вам следует повторить все отмененные транзакции (хотя некоторые СУБД автоматически повторяют незавершенные транзакции);

- функция сохранения/восстановления резервных копий СУБД зависит от транзакций. Функция резервного копирования регулярно сохраняет копии базы данных вместе с журналами транзакций на резервном диске. Предположим, что при сбое системы рабочий диск был поврежден, поэтому данные и журнал транзакций стали нечитаемыми. Вы можете воспользоваться функцией восстановления, которая загрузит последнюю резервную копию базы данных и затем исполнит все выполненные транзакции в журнале с момента сохранения резервной копии до последней транзакции перед сбоем.

При этом база данных восстанавливается в том виде, в котором она была до сбоя (вам опять-таки придется повторить все незаконченные транзакции);

- по очевидным причинам вам следует хранить базу данных и журнал транзакций на отдельных физических дисках.

Для исполнения транзакции ее границы (начальная и конечная точки) должны быть четко обозначены. Эти границы позволяют СУБД исполнить все команды в виде единой операции. В соответствии со стандартом SQL транзакция всегда начинается с первой команды SQL и заканчивается командой COMMIT или ROLLBACK.

### **Контроль параллелизма**

Некоторые пользователи считают, что компьютеры выполняют несколько действий одновременно. На самом деле эти действия выполняются не одновременно, а последовательно. Иллюзия одновременного исполнения возникает потому, что микропроцессор работает с отрезками времени, намного меньшими, чем человек может представить. В СУБД контроль параллелизма представляет собой набор правил, которые предотвращают нарушения целостности данных, вызванные тем, что несколько пользователей одновременно обращаются к данным или изменяют их.

СУБД использует блокировку (самую распространенную методику), чтобы обеспечить целостность при исполнении операций, а также целостность базы данных. Блокировка запрещает доступ к данным при считывании и записи; таким образом, она не дает пользователям считать данные, которые изменяются другими пользователями. При этом несколько пользователей не смогут одновременно изменять одни и те же данные. Без блокировки данные могут стать логически неверными и команды, использующие такие данные, будут выдавать неправильные результаты. Механизмы блокировки очень сложны; обратитесь к документации по вашей СУБД.

Прозрачность параллелизма – это механизм, благодаря которому при исполнении транзакции создается впечатление, что это единственная операция, которая исполняется с базой данных. СУБД изолирует изменения, выполненные транзакцией, от изменений, выполненных другими транзакциями. Транзакция никогда не использует данные в промежуточном состоянии; она работает с данными либо до изменения, либо после завершения других транзакций. Этот уровень изоляции позволяет перезагрузить начальные данные и выполнить серию транзакций, которые приведут данные в то состояние, в котором они были после исполнения оригинальных транзакций.



Oracle совместим со стандартом ANSI. В других СУБД вы можете (или должны) начинать операцию с помощью определенной команды. СУБД обычно использует ключевое слово `BEGIN`, чтобы точно обозначить начало транзакции. `BEGIN` не является частью стандарта SQL, поэтому его использование отличается для разных СУБД (обратитесь к документации по вашей СУБД).

## Как начать транзакцию

В Microsoft Access или Microsoft SQL Server введите `BEGIN TRANSACTION`;

или

в MySQL или PostgreSQL введите `BEGIN`;

## Порядок выполнения транзакции

Введите `COMMIT`;

## Отмена транзакции

Введите `ROLLBACK`;

Команды `SELECT` в листинге 13.1 показывают, что транзакции `UPDATE` исполняются СУБД, а затем отменяются командой `ROLLBACK`. Результат исполнения листинга показан на рис. 13.2.

Листинг 13.2 демонстрирует более практическое применение транзакций. Мы хотим удалить издательство P04 из таблицы `publishers` без нарушения ссылочной целостности. Поскольку некоторые значения вторичного ключа в `titles` указывают на издательство P04, сначала нам нужно удалить связанные строки из таблиц `titles`, `titles_authors` и `royalties`. Мы используем транзакцию, чтобы убедиться, что все команды `DELETE` будут исполнены. Если не все команды были исполнены успешно, то данные останутся неизменными (за информацией о проверках ссылочной

**Листинг 13.1.** Команды `UPDATE` (как и команды `INSERT` и `DELETE`) в группе команд транзакции никогда не являются конечными. Результат исполнения листинга показан на рис. 13.2

```

Листинг
SELECT SUM(pages), AVG(price) FROM titles;

BEGIN;
UPDATE titles SET pages = 0;
UPDATE titles SET price = price * 2;
SELECT SUM(pages), AVG(price) FROM titles;
ROLLBACK;

SELECT SUM(pages), AVG(price) FROM titles;
```

| SUM(pages) | AVG(price) |
|------------|------------|
| 5107       | 18.3975    |

| SUM(pages) | AVG(price) |
|------------|------------|
| 0          | 36.7750    |

| SUM(pages) | AVG(price) |
|------------|------------|
| 5107       | 18.3975    |

**Рис. 13.2.** Результат листинга 13.1. Результаты исполнения команд `SELECT` показывают, что СУБД отменила транзакцию

**Листинг 13.2.** Используйте транзакцию, чтобы удалить издательство P04 из таблицы `publishers`, а также связанные с ней строки в других таблицах

```

Листинг

BEGIN;

DELETE FROM title_authors
 WHERE title_id IN
 (SELECT title_id
 FROM titles
 WHERE pub_id = 'P04');

DELETE FROM royalties
 WHERE title_id IN
 (SELECT title_id
 FROM titles
 WHERE pub_id = 'P04');

DELETE FROM titles
 WHERE pub_id = 'P04';

DELETE FROM publishers
 WHERE pub_id = 'P04';

COMMIT;

```

целостности обратитесь к разделу «Задание внешнего ключа с помощью ограничения FOREIGN KEY» в главе 10).

**С**

Всегда завершайте транзакцию с помощью команды `COMMIT` или `ROLLBACK`. Отсутствие конечной команды может привести к возникновению больших транзакций с непредсказуемыми результатами, к прерыванию исполнения программы или к восстановлению последней незавершенной транзакции.

**П**

Вы можете располагать транзакции на нескольких уровнях. Максимальное количество уровней зависит от конкретной СУБД.

**П**

Большинство СУБД по умолчанию работают в режиме автоматического исполнения, если только вы не ввели ключевое слово для транзакции. В *режиме автоматического исполнения* каждая команда исполняется как отдельная транзакция. Если команда была завершена успешно, СУБД исполняет ее; если СУБД обнаруживает ошибку, команда будет отменена.

**П**

Работая с большими транзакциями, вы можете задать промежуточные маркеры (*точки сохранения*), чтобы разделить транзакцию на несколько частей. Точки сохранения позволяют отменять изменения от текущей точки в транзакции до предыдущей точки (при условии, что транзакция пока не была выполнена). Представьте себе сессию, в которой вы ввели несколько неисполненных команд `INSERT`, `UPDATE` и `DELETE`, а затем поняли, что последние изменения неверны или не нужны. С помощью точек сохранения вы можете избежать восстановления всех команд. Microsoft SQL Server и Oracle поддерживают точки сохранения.



Чтобы запустить листинги 13.1 и 13.2 в Microsoft Access, вместо команды `BEGIN` используйте `BEGIN TRANSACTION`. В Access вы не можете исполнять транзакции посредством пользовательского интерфейса Access SQL View или DAO; для этих целей нужен провайдер Jet OLE DB и ADO.

Чтобы запустить листинги 13.1 и 13.2 в Microsoft SQL Server, замените `BEGIN` на `BEGIN TRANSACTION`.

Транзакции в Oracle начинаются без ввода ключевого слова. Чтобы запустить листинги 13.1 и 13.2 в Oracle, пропустите ключевое слово `BEGIN`.

MySQL 4.0 и более ранних версий поддерживает транзакции в таблицах InnoDB и BDB, но не в родных таблицах (обратитесь к документации по MySQL).

# ПРИЛОЖЕНИЕ

---



В большинстве примеров данной книги используется база данных `books` (см. раздел «Наша типовая база данных» в главе 2). Чтобы работать с примерами, необходимо создать в вашей СУБД базу данных `books`.

В этом приложении мы расскажем, как создать такую базу данных, построить ее таблицы и заполнить их данными. Все нужные файлы вы можете загрузить с Web-страницы данной книги по адресу [www.peachpit.com/vqs/sql](http://www.peachpit.com/vqs/sql) или с сайта издательства ДМК Пресс [www.dmkpress.ru](http://www.dmkpress.ru). Файлы хранятся в архиве с расширением `zip`. Чтобы разархивировать их, используйте специальную утилиту, например WinZip (Windows – [www.winzip.com](http://www.winzip.com)), `gunzip` или `unzip` (Mac OS – [www.stuffit.com](http://www.stuffit.com)). После разархивирования прочитайте файл `readme.txt`.

---

## Создание типовой базы данных books

Если вы используете Microsoft Access, просто откройте файл books.mdb. Если вы используете другую СУБД, вам необходимо создать базу с именем books, а затем запустить SQL-скрипт, чтобы построить ее таблицы и заполнить их данными. В листинге A.1 показан скрипт ANSI SQL под названием create\_books.sql, который создает таблицы и добавляет в них строки. Чтобы создать образец базы данных в СУБД, требуется выполнить следующие действия:

1. Создать базу данных books.
2. Если нужно, изменить create\_books.sql таким образом, чтобы работать с вашей СУБД.
3. Запустить create\_books.sql для базы данных books.

Мы опишем процесс для каждой СУБД, рассматриваемой в этой книге. Как всегда, по умолчанию мы допускаем, что вы получили соответствующее разрешение у администратора базы данных и имеете

к ней доступ (см. советы в разделе «Выполнение программ SQL» в главе 1).

Все СУБД поддерживают команду CREATE DATABASE, которая создает новую базу данных, но мы опишем, как использовать другие, более простые инструменты. Если вы предпочитаете работать с командой CREATE DATABASE, обратитесь к документации по вашей СУБД. Эта команда не является стандартной командой SQL, поэтому ее структура и возможности различаются в разных СУБД (в стандарте SQL-92 ближайшей командой к CREATE DATABASE является CREATE SCHEMA).

### Открытие базы данных books в программе Microsoft Access

1. Выберите пункты меню **File** ⇒ **Open** (Файл ⇒ Открыть), а затем **books.mdb** и щелкните по кнопке **Open** (Открыть).
2. Чтобы изучить таблицы, нажмите кнопку **Tables** (Таблицы) под кнопкой **Objects** (Объекты) в окне **Database** (База данных).

**Листинг A.1.** SQL-скрипт create\_books.sql создает все таблицы в базе данных books и заполняет их данными. Этот скрипт написан на ANSI SQL для максимальной совместимости, но вам может понадобиться внести в него некоторые изменения, чтобы запустить его на вашей СУБД

Листинг

```
DROP TABLE authors;
CREATE TABLE authors
(
 au_id CHAR(3) NOT NULL ,
 au_fname VARCHAR(15) NOT NULL ,
 au_lname VARCHAR(15) NOT NULL ,
 phone VARCHAR(12) NULL ,
 address VARCHAR(20) NULL ,
 city VARCHAR(15) NULL ,
 state CHAR(2) NULL ,
 zip CHAR(5) NULL ,
 CONSTRAINT authors_pk PRIMARY KEY (au_id)
);
```



**Листинг А.1** (продолжение)

Листинг

```
DROP TABLE publishers;
CREATE TABLE publishers
(
 pub_id CHAR(3) NOT NULL ,
 pub_name VARCHAR(20) NOT NULL ,
 city VARCHAR(15) NOT NULL ,
 state CHAR(2) NULL ,
 country VARCHAR(15) NOT NULL ,
 CONSTRAINT publishers_pk PRIMARY KEY (pub_id)
);

DROP TABLE titles;
CREATE TABLE titles
(
 title_id CHAR(3) NOT NULL ,
 title_name VARCHAR(40) NOT NULL ,
 type VARCHAR(10) NULL ,
 pub_id CHAR(3) NOT NULL ,
 pages INTEGER NULL ,
 price DECIMAL(5,2) NULL ,
 sales INTEGER NULL ,
 pubdate DATE NULL ,
 contract SMALLINT NOT NULL ,
 CONSTRAINT titles_pk PRIMARY KEY (title_id)
);

DROP TABLE title_authors;
CREATE TABLE title_authors
(
 title_id CHAR(3) NOT NULL ,
 au_id CHAR(3) NOT NULL ,
 au_order SMALLINT NOT NULL ,
 royalty_share DECIMAL(5,2) NOT NULL ,
 CONSTRAINT title_authors_pk PRIMARY KEY (title_id, au_id)
);

DROP TABLE royalties;
CREATE TABLE royalties
(
 title_id CHAR(3) NOT NULL ,
 advance DECIMAL(9,2) NULL ,
 royalty_rate DECIMAL(5,2) NULL ,
 CONSTRAINT royalties_pk PRIMARY KEY (title_id)
);
```

## Листинг А.1 (продолжение)

## ЛИСТИНГ

```

INSERT INTO authors VALUES('A01','Sarah','Buchman','718-496-7223',
 '75 West 205 St','Bronx','NY','10468');
INSERT INTO authors VALUES('A02','Wendy','Heydemark','303-986-7020',
 '2922 Baseline Rd','Boulder','CO','80303');
INSERT INTO authors VALUES('A03','Hallie','Hull','415-549-4278',
 '3800 Waldo Ave, #14F','San Francisco','CA','94123');
INSERT INTO authors VALUES('A04','Klee','Hull','415-549-4278',
 '3800 Waldo Ave, #14F','San Francisco','CA','94123');
INSERT INTO authors VALUES('A05','Christian','Kells','212-771-4680',
 '114 Horatio St','New York','NY','10014');
INSERT INTO authors VALUES('A06','','Kellsey','650-836-7128',
 '390 Serra Mall','Palo Alto','CA','94305');
INSERT INTO authors VALUES('A07','Paddy','O''Furniture','941-925-0752',
 '1442 Main St','Sarasota','FL','34236');

INSERT INTO publishers VALUES('P01','Abatis Publishers','New York','NY','USA');
INSERT INTO publishers VALUES('P02','Core Dump Books','San Francisco','CA','USA');
INSERT INTO publishers VALUES('P03','Schadenfreude Press','Hamburg',NULL,'Germany');
INSERT INTO publishers VALUES('P04','Tenterhooks Press','Berkeley','CA','USA');

INSERT INTO titles VALUES('T01','1977!','history','P01',
 107,21.99,566,DATE '2000-08-01',1);
INSERT INTO titles VALUES('T02','200 Years of German Humor','history','P03',
 14,19.95,9566,DATE '1998-04-01',1);
INSERT INTO titles VALUES('T03','Ask Your System Administrator','computer','P02',
 1226,39.95,25667,DATE '2000-09-01',1);
INSERT INTO titles VALUES('T04','But I Did It Unconsciously','psychology','P04',
 510,12.99,13001,DATE '1999-05-31',1);
INSERT INTO titles VALUES('T05','Exchange of Platitudes','psychology','P04',
 201,6.95,201440,DATE '2001-01-01',1);
INSERT INTO titles VALUES('T06','How About Never?','biography','P01',
 473,19.95,11320,DATE '2000-07-31',1);
INSERT INTO titles VALUES('T07','I Blame My Mother','biography','P03',
 333,23.95,1500200,DATE '1999-10-01',1);
INSERT INTO titles VALUES('T08','Just Wait Until After School','children','P04',
 86,10.00,4095,DATE '2001-06-01',1);
INSERT INTO titles VALUES('T09','Kiss My Boo-Boo','children','P04',
 22,13.95,5000,DATE '2002-05-31',1);
INSERT INTO titles VALUES('T10','Not Without My Faberge Egg','biography','P01',
 NULL,NULL,NULL,NULL,0);
INSERT INTO titles VALUES('T11','Perhaps It's a Glandular Problem','psychology','P04',
 826,7.99,94123,DATE '2000-11-30',1);
INSERT INTO titles VALUES('T12','Spontaneous, Not Annoying','biography','P01',
 507,12.99,100001,DATE '2000-08-31',1);
INSERT INTO titles VALUES('T13','What Are The Civilian Applications?','history','P03',
 802,29.99,10467,DATE '1999-05-31',1);

```

**Листинг А.1** (окончание)

| Листинг                   |                               |
|---------------------------|-------------------------------|
| INSERT INTO title_authors | VALUES('T01', 'A01', 1, 1.0); |
| INSERT INTO title_authors | VALUES('T02', 'A01', 1, 1.0); |
| INSERT INTO title_authors | VALUES('T03', 'A05', 1, 1.0); |
| INSERT INTO title_authors | VALUES('T04', 'A03', 1, 0.6); |
| INSERT INTO title_authors | VALUES('T04', 'A04', 2, 0.4); |
| INSERT INTO title_authors | VALUES('T05', 'A04', 1, 1.0); |
| INSERT INTO title_authors | VALUES('T06', 'A02', 1, 1.0); |
| INSERT INTO title_authors | VALUES('T07', 'A02', 1, 0.5); |
| INSERT INTO title_authors | VALUES('T07', 'A04', 2, 0.5); |
| INSERT INTO title_authors | VALUES('T08', 'A06', 1, 1.0); |
| INSERT INTO title_authors | VALUES('T09', 'A06', 1, 1.0); |
| INSERT INTO title_authors | VALUES('T10', 'A02', 1, 1.0); |
| INSERT INTO title_authors | VALUES('T11', 'A03', 2, 0.3); |
| INSERT INTO title_authors | VALUES('T11', 'A04', 3, 0.3); |
| INSERT INTO title_authors | VALUES('T11', 'A06', 1, 0.4); |
| INSERT INTO title_authors | VALUES('T12', 'A02', 1, 1.0); |
| INSERT INTO title_authors | VALUES('T13', 'A01', 1, 1.0); |
|                           |                               |
| INSERT INTO royalties     | VALUES('T01', 10000, 0.05);   |
| INSERT INTO royalties     | VALUES('T02', 1000, 0.06);    |
| INSERT INTO royalties     | VALUES('T03', 15000, 0.07);   |
| INSERT INTO royalties     | VALUES('T04', 20000, 0.08);   |
| INSERT INTO royalties     | VALUES('T05', 100000, 0.09);  |
| INSERT INTO royalties     | VALUES('T06', 20000, 0.08);   |
| INSERT INTO royalties     | VALUES('T07', 1000000, 0.11); |
| INSERT INTO royalties     | VALUES('T08', 0, 0.04);       |
| INSERT INTO royalties     | VALUES('T09', 0, 0.05);       |
| INSERT INTO royalties     | VALUES('T10', NULL, NULL);    |
| INSERT INTO royalties     | VALUES('T11', 100000, 0.07);  |
| INSERT INTO royalties     | VALUES('T12', 50000, 0.09);   |
| INSERT INTO royalties     | VALUES('T13', 20000, 0.06);   |

Чтобы запустить команды SQL с books, обратитесь к разделу «Microsoft Access» в главе 1.

### Создание базы данных books в программе Microsoft SQL Server

1. Выберите пункты меню **Start** ⇒ **Programs** ⇒ **Microsoft SQL Server** ⇒ **Enterprise Manager** (Пуск ⇒ Программы ⇒ Microsoft SQL Server ⇒ Enterprise Manager).

2. Перейдите в папку Databases (Базы данных) на сервере, с которым вы работаете (см. рис. А.1).
3. Выберите пункты меню **Tools** ⇒ **Wizards** (Сервис ⇒ Мастер).
4. В разделе **Database** (База данных) выберите пункт **Create Database Wizard** (Мастер создания базы данных) и нажмите на кнопку **OK** (см. рис. А.2).
5. Нажмите на кнопку **Next** (Далее), чтобы пропустить окно **Welcome** (Приветствие).

6. Введите в поле **Database Name** (Имя базы данных) слово **books**, затем нажмите на кнопку **Next** (Далее).

Если вы желаете изменить местоположение базы данных и файлов журнала операций, воспользуйтесь кнопками поиска (см. рис. А.3).

7. Примение название по умолчанию, заданное для файла базы данных, а также начальный размер (см. рис. А.4). Нажмите на кнопку **Next** (Далее).
8. Воспользуйтесь заданными по умолчанию установками размеров файлов в базе данных (см. рис. А.5). Нажмите на кнопку **Next** (Далее).
9. Согласитесь с заданными по умолчанию размерами файла журнала транзакций и начальным размером (см. рис. А.6). Нажмите на кнопку **Next** (Далее).
10. Согласитесь с заданными по умолчанию установками увеличения размеров файла журнала транзакций (см. рис. А.7). Нажмите на кнопку **Next** (Далее).
11. Нажмите на кнопку **Finish** (Закончить), чтобы подтвердить выбор (см. рис. А.8).
12. Нажмите на кнопку **OK**, чтобы закрыть окно подтверждения.
13. Нажмите на кнопку **NO** (Нет), чтобы отказаться от создания плана обслуживания. Теперь на панели **Details** (Свойства) в окне **Enterprise Manager** отобразится база данных **books**.
14. Чтобы запустить скрипт `create_books.sql` в SQL Server, используйте тип данных столбца `pubdate` в таблице `titles` не `DATE`, а `DATETIME` и удалите ключевое слово `DATE` в буквенных обозначениях дат во всех командах `INSERT INTO titles`. Например, замените `DATE '2000-08-01'` на `'2000-08-01'`.

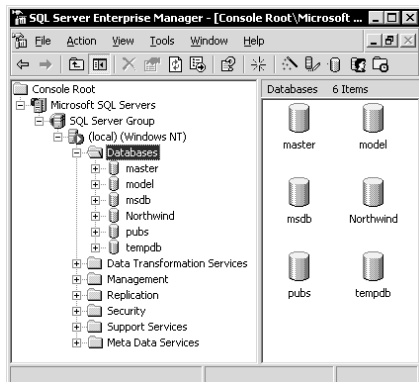


Рис. А.1. SQL Server отображает список баз данных на панели **Details** (Свойства)

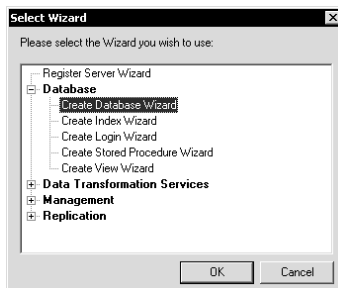


Рис. А.2. Мастер Create Database Wizard последовательно выполняет все действия, которые необходимы для создания базы данных

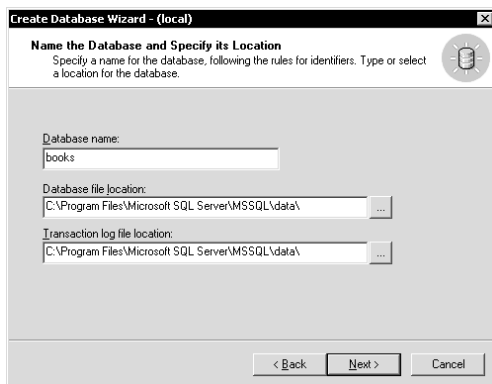
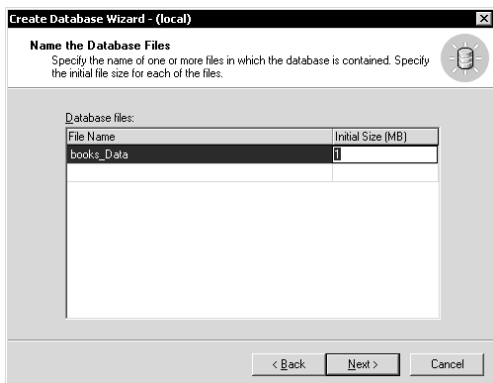
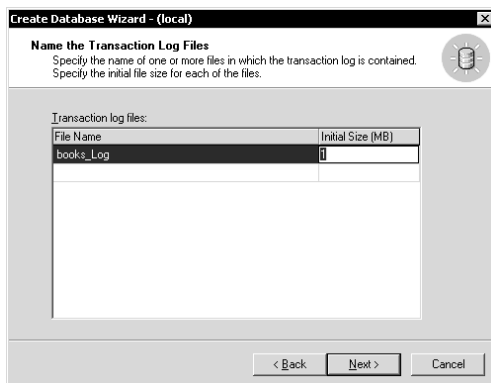


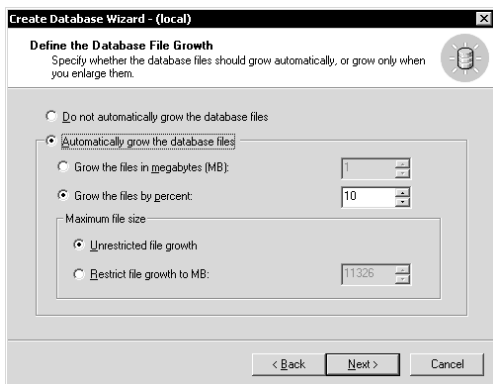
Рис. А.3. Введите название базы данных **books**



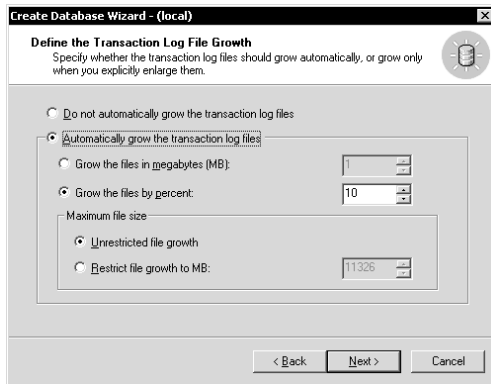
**Рис. А.4.** Согласитесь с названием по умолчанию, заданным для файла базы данных, а также начальным размером



**Рис. А.6.** Примите заданные по умолчанию размеры файла журнала транзакций и начальный размер



**Рис. А.5.** Для этой базы данных разрешите SQL Server автоматически увеличивать размеры файла базы данных, а не делайте это самостоятельно



**Рис. А.7.** Для этой базы данных разрешите SQL Server автоматически увеличивать размеры файла журнала транзакций, а не делайте это самостоятельно

15. Выберите пункты меню **Start** ⇒ **Programs** ⇒ **Microsoft SQL Server** ⇒ **Query Analyzer** (Пуск ⇒ Программы ⇒ Microsoft SQL Server ⇒ Query Analyzer).
16. Укажите сервер и режим аутентификации, затем нажмите на кнопку **OK**.
17. Выберите в поле на панели инструментов базу данных **books**.

18. Выберите пункты меню **File** ⇒ **Open** (Файл ⇒ Открыть), затем в диалоговом окне **Open** (Открыть) файл **create\_books.sql** и щелкните по кнопке **OK**.
19. Выберите пункты меню **Query** ⇒ **Execute** (Запрос ⇒ Выполнить) либо нажмите клавишу **F5** (см. рис. А.9).

20. Чтобы запустить SQL-скрипты и интерактивные команды с books, обратиться к разделу «Microsoft SQL Server» в главе 1.

## Создание базы данных books в Oracle

1. Запустите программу Database Configuration Assistant. Эта процедура отличается для разных платформ. Например, в Windows вам следует выбрать пункты меню **Start** ⇒ **Programs** ⇒ **Oracle** ⇒ **OraHome91** ⇒ **Configuration and Migration Tools** ⇒ **Database Configuration Assistant** (Пуск ⇒ Программы ⇒ Oracle ⇒ OraHome91 ⇒ Configuration and Migration Tools ⇒ Database Configuration Assistant).
2. Нажмите на кнопку **Next** (Далее), чтобы пропустить окно **Welcome** (Приветствие).
3. Выберите пункт **Create a Database** (Создать базу данных) – см. рис. А.10, затем нажмите на кнопку **Next** (Далее).
4. Выберите пункт **General Purpose** (Общего назначения) – см. рис. А.11, затем нажмите на кнопку **Next** (Далее).
5. Введите в полях **Global Database Name** (Глобальное имя базы данных) и **SID** (System Identifier – системный идентификатор) слово books (см. рис. А.12), затем нажмите на кнопку **Next** (Далее).
6. Выберите режим **Dedicated Server Mode** (Персонализированный режим) – см. рис. А.13, затем нажмите на кнопку **Next** (Далее).
7. Выберите пункт **Typical** (Типичные), примите другие установки инициализации, заданные по умолчанию (см. рис. А.14), затем нажмите на кнопку **Next** (Далее).



Рис. А.8. SQL Server готов к созданию базы данных

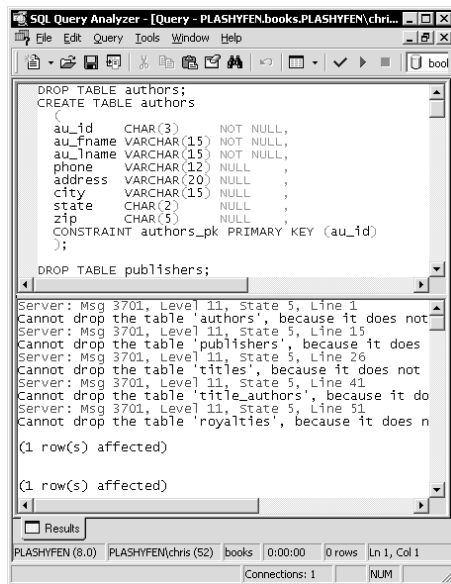


Рис. А.9. Используйте программу Query Analyzer, чтобы создать таблицы и добавить в них данные. Игнорируйте предупреждения, которые появляются при первом запуске скрипта

8. Примите заданные по умолчанию установки **Database Storage** (Хранилище базы данных) – см. рис. А.15, затем нажмите на кнопку **Next** (Далее).
9. Выберите пункт **Create Database** (Создать базу данных) – см. рис. А.16, затем нажмите на кнопку **Finish** (Закончить).
10. Нажмите на кнопку **OK**, чтобы пропустить окно **Summary** (Итого) и создать базу данных.
11. Нажмите на кнопку **Exit** (Выход) в сообщении, которое появится, когда Oracle завершит создание базы данных.
12. Для запуска скрипта `create_books.sql` в Oracle у автора A06 в столбце `au_fname` таблицы `authors` замените пустую строку (") символом пробела (' '). Это изменение не позволяет Oracle интерпретировать имя автора A06 как NULL (см. примечание **DBMS** в разделе «Значение null» главы 3).
13. Запустите SQL\*Plus. Эта процедура отличается для разных платформ. Например, в Windows вам следует выбрать пункты **Start** ⇒ **Programs** ⇒ **Oracle** ⇒ **OraHome91** ⇒ **Application Development** ⇒ **SQL Plus** (Пуск ⇒ Программы ⇒ Oracle ⇒ OraHome91 ⇒ Application Development ⇒ SQL Plus).
14. Введите имя пользователя, пароль и имя базы `books`, затем нажмите на кнопку **OK**.
15. В строке SQL наберите:

```
@create_books.sql
```

Затем нажмите клавишу **Enter** (см. рис. А.17).

Вы можете добавить абсолютное или относительное имя пути (см. примечание в разделе «Выполнение программ SQL» главы 1).

Монитор SQL\*Plus отобразит результаты. Игнорируйте сообщения Oracle о том, что он не может удалить несуществующие таблицы. Команды `DROP TABLE` нужны для того, чтобы при следующем запуске скрипта `create_books.sql` восстановить таблицы в первоначальном виде.

16. Чтобы запустить SQL-скрипты и интерактивные команды с `books`, обратитесь к разделу «Oracle» в главе 1.

## Создание базы данных books в MySQL

1. В командной строке операционной системы (оболочки) наберите `mysqladmin create books`, затем нажмите клавишу **Enter**. MySQL создаст базу данных `books`.

2. Введите в оболочке:

```
mysql -f books < create_books.sql
```

Опция `-f` заставляет `mysql` работать, даже если будут обнаружены ошибки SQL (см. рис. А.18). Вам не нужно изменять скрипт `create_books.sql`, чтобы запустить его в MySQL. Вы можете добавить абсолютное или относительное имя пути (см. примечание в разделе «Выполнение программ SQL» главы 1).

`mysql` отобразит результаты. Игнорируйте сообщения MySQL о том, что программа не может удалить несуществующие таблицы. Команды `DROP TABLE` нужны для того, чтобы при последующем запуске скрипта `create_books.sql` восстановить таблицы в первоначальном виде.

3. Чтобы запустить SQL-скрипты и интерактивные команды с `books`, обратитесь к разделу «MySQL» в главе 1.

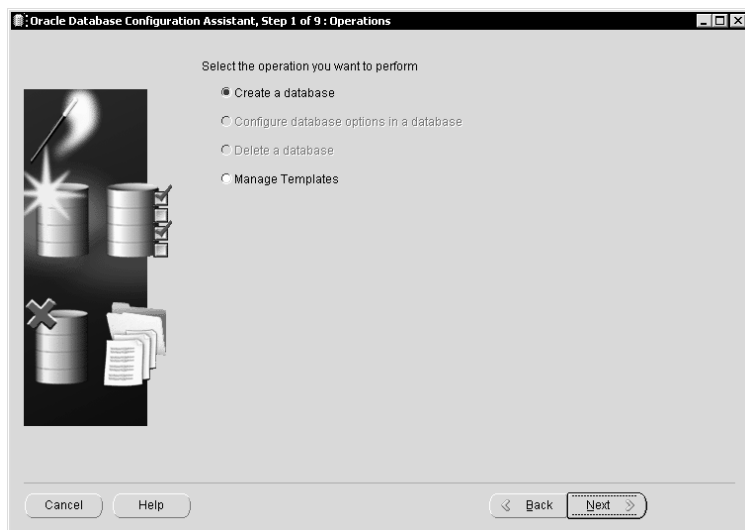


Рис. А.10. Программа Database Configuration Assistant последовательно выполняет все действия для создания базы данных

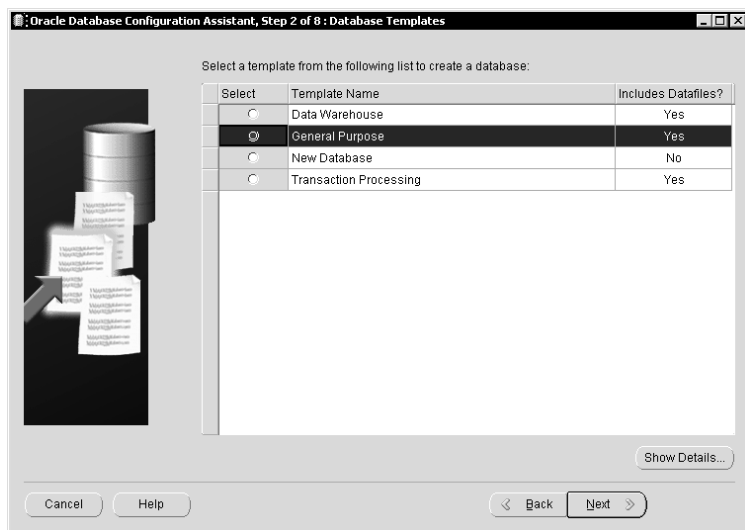


Рис. А.11. Мы выбрали пункт **General Purpose** (Общего назначения). Вместо него вы можете выбрать пункт **New Database** (Новая база данных), но при этом создание базы данных займет у Oracle больше времени



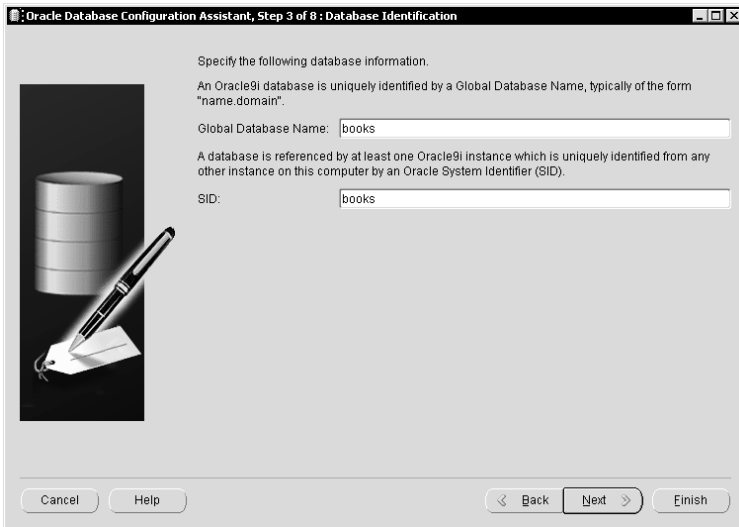


Рис. А.12. Присвойте название базе данных books

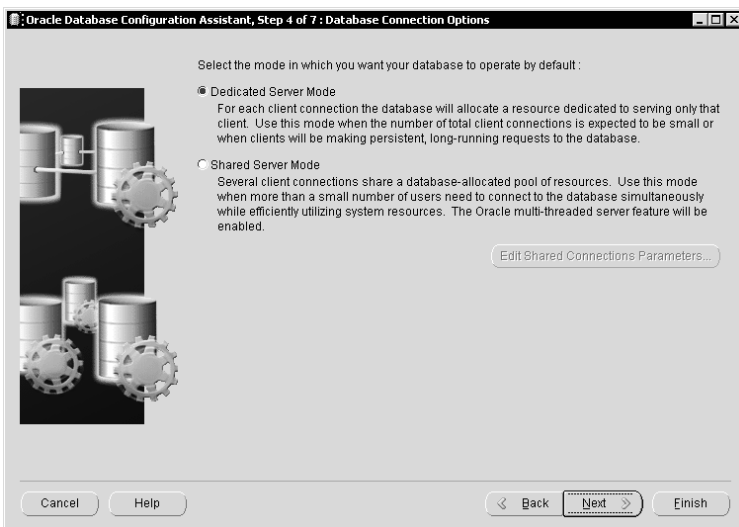
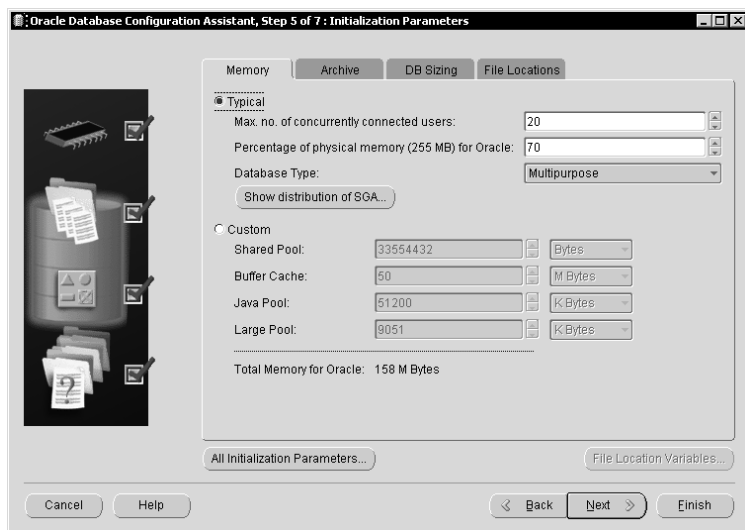
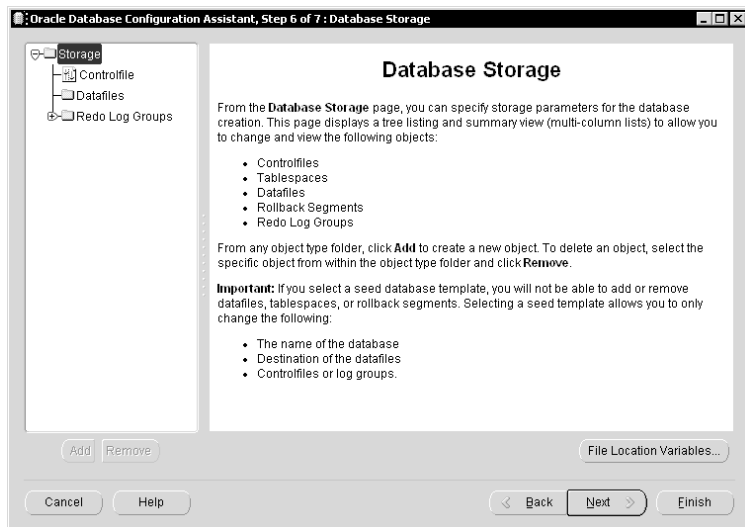


Рис. А.13. Следует использовать режим **Dedicated Server Mode**, если база данных предназначена для небольшого числа клиентов



**Рис. А.14.** Установка **Typical** используется для создания базы данных с небольшим объемом данных



**Рис. А.15.** Согласитесь с заданными по умолчанию установками **Database Storage**

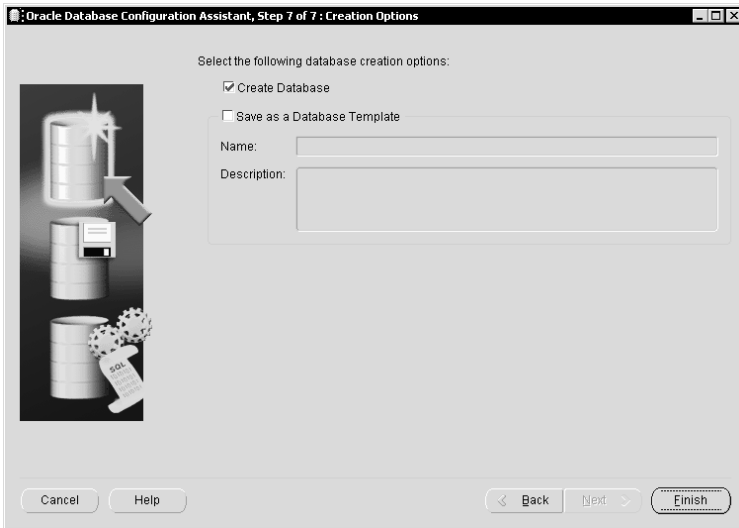


Рис. А.16. Oracle готов к созданию базы данных

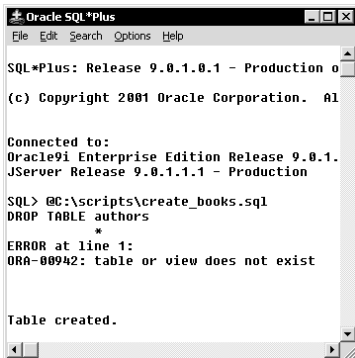


Рис. А.17. Используйте программу SQL\*Plus, чтобы создать таблицы и добавить в них данные. Игнорируйте предупреждения, которые появляются при первом запуске скрипта

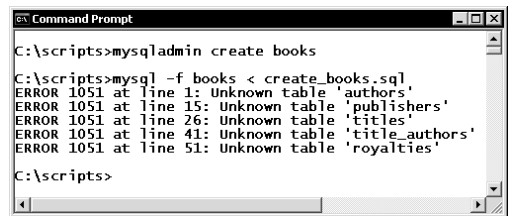


Рис. А.18. Создавая базу данных books в MySQL, игнорируйте предупреждающие сообщения, которые появляются при первом запуске скрипта

## C

Если вы запустили MySQL с удаленного компьютера в сети, необходимо указать хост (имя сервера), имя пользователя и пароль, чтобы подключиться к серверу. Для этого введите `mysql -h host -u user -p dbname`.

*host* — это название сервера, *user* — имя пользователя, а *dbname* — название базы данных, которая будет использоваться. MySQL попросит вас ввести пароль. Обратитесь к администратору базы данных, чтобы узнать, какие настройки соединения следует ввести.

## Создание базы данных books в PostgreSQL

1. В командной строке операционной системы (оболочки) наберите `createdb books`, затем нажмите клавишу **Enter**. PostgreSQL создаст базу данных `books`.

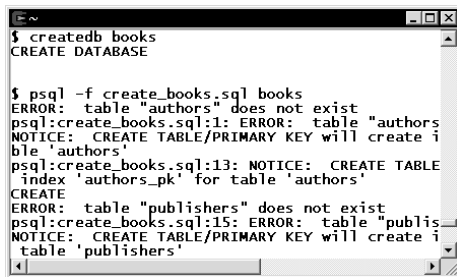
2. Введите в оболочке:

```
psql -f create_books.sql books
```

Опция `-f` задает название файла SQL-скрипта (см. рис. А.19). Вам не нужно изменять скрипт `create_books.sql`, чтобы запустить его в PostgreSQL. Можно добавить абсолютное или относительное имя пути (см. примечание в разделе «Выполнение программ SQL» главы 1).

`psql` отобразит результаты. Игнорируйте сообщения PostgreSQL о том, что программа не может удалить несуществующие таблицы. Команды `DROP TABLE` нужны для того, чтобы при последующем запуске скрипта `create_books.sql` восстановить таблицы в первоначальном виде.

3. Чтобы запустить SQL-скрипты и интерактивные команды с `books`, обратитесь к разделу «PostgreSQL» главы 1.



**Рис. А.19.** Создавая базу данных `books` в PostgreSQL, игнорируйте предупреждающие сообщения, которые появляются при первом запуске скрипта

## C

Если вы запустили PostgreSQL с удаленного компьютера в сети, необходимо указать хост, имя пользователя и пароль, чтобы подключиться к серверу. Для этого введите:

```
psql -h host -U user -W dbname
```

*host* — это название хоста, *user* — имя пользователя, а *dbname* — название базы данных, которая будет использоваться. PostgreSQL попросит ввести пароль. Обратитесь к администратору базы данных, чтобы узнать, какие настройки соединения следует ввести.

# ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

---

## А

Альтернативный ключ 50  
Аргумент 137

## Б

База данных  
    нормализация 57  
    типовая 23, 63  
Битовый тип данных 79  
Блокировка 427

## В

Внешний ключ  
    простой 383  
    сложный 383  
Временная таблица  
    глобальная 395  
    локальная 395  
Выражение 74  
    CASE 179  
    DECODE() 183

## Г

Глагол 73  
Группа, повторяющаяся 58

## Д

Действительный числовой тип данных 82  
Декомпозиция  
    без потерь 57  
    с сохранением зависимостей 57

Детерминант 60  
Домен 44, 75

## З

Зависимость  
    полная функциональная 60  
    транзитивная 60  
    частичная функциональная 58  
Заголовок столбца  
    alias 96  
    по умолчанию 96  
    псевдоним 96  
Замкнутость  
    таблиц в реляционной модели 45  
Заполнение таблицы 357  
Запрос  
    внешний 300  
    внутренний 300  
Значение 44  
    null 53  
    по умолчанию 376  
    элементарное 58

## И

Идентификатор 70  
Имя столбца, уточненное 220  
Индекс 405  
    простой 406  
    сложный 406  
    удаление 409  
    уникальный 406  
Интервальный тип данных 88

**К**

Календарный тип данных 84

Каталог

базы данных

стандарт ANSI 47

системный, СУБД 46

Ключ

альтернативный 50

внешний 51, 383

свойства 51

вторичный 383

простой 383

сложный 383

первичный 47

создание 379

простой 379

сложный 379

Ключевое слово

CHECK 392

ESCAPE 125

Команда

ALTER TABLE 401

COMMIT 427

CREATE DATABASE 432

CREATE INDEX 407

CREATE TABLE 368

CREATE VIEW 414

DELETE 363

DROP INDEX 409

DROP TABLE 404

DROP VIEW 424

INSERT 352

ROLLBACK 427

SELECT 92

ALL 100

AS 96

DISTINCT 99

FROM 93

ORDER BY 101

WHERE 109

SQL 70

UPDATE 358

Комментарии 69, 74

**Л**

Лексема команды SQL 70

Литерал 76

**М**

Метаданные 46

Метасимвол 122

непереключенный 125

переключенный 125

Модель реляционная 41

Мощность таблицы 47

**Н**

Нормализация 57

Нормальная форма

первая (1НФ) 58

вторая (2НФ) 58

третья (3НФ) 60

четвертая, пятая 57

**О**

Объединение 224

JOIN

CROSS 225, 233

FULL OUTER 225

INNER 225, 241

LEFT OUTER 225, 265

NATURAL 225, 236

RIGHT OUTER 225

SELF 225, 279

UNION 267

внешнее 265

синтаксис JOIN 228

синтаксис WHERE 228

табличная операция 219

Объектный тип 43

Ограничение 368

check 392

NOT NULL 374

NULL 374

PRIMARY KEY 379

наименование 370

столбца 369  
таблицы 369  
уникальности 389  
    простое 389  
    сложное 389  
Операнд 137  
Оператор 137  
    UNION 286  
    бинарный арифметический 140  
    конкатенации 145  
    логический 114  
        AND 114  
        NOT 116  
        OR 115  
    отрицания 140, 143  
    перегружаемый 166  
    сравнения 110  
        BETWEEN 128  
        IN 131  
        IS NULL 134  
        LIKE 122  
    тождества 140, 143  
    унарный арифметический 140  
Операция  
    бинарная 44  
    обработки наборов записей 227  
    проекции 95  
    унарная 44  
Отличие 297  
Очередность операторов 143

## **П**

Первичный ключ 47  
    простой 48  
    составной 48  
Перегрузка оператора 166  
Переключающий символ 124  
Пересечение 295  
Повторяющаяся группа 58  
Подзапрос 299  
Порядок таблицы 47  
Правило ассоциативности 143  
Предикат 109

Предложение  
    AS 96, 222  
    CONSTRAINT 370  
    DISTINCT 202  
    FROM 70, 93  
    GROUP BY 206  
    HAVING 215  
    ORDER 70  
    ORDER BY 101  
        ASC 102  
        DESC 102  
    SELECT 70  
    SQL 70  
    USING 232  
    WHERE 70  
Представление  
    неизменяемое 419  
    создание 411  
    удаление 424  
Преобразование сужающее 176  
Преобразование типов неявное 173  
Прозрачность параллелизма 427  
Просмотр структуры таблицы  
    в Microsoft Access 349  
    в Microsoft SQL Server 350  
    в MySQL 350  
    в Oracle 350  
    в PostgreSQL 351  
Псевдонимы таблиц 222

## **Р**

Режим  
    автоматического исполнения 429  
Реляционная модель 41

## **С**

Свойства  
    внешнего ключа 51  
    столбца таблицы базы данных 44  
Связь  
    ведущий-ведомый 56  
    многие-ко-многим 55  
    один-к-одному 54  
    один-ко-многим 55

- предок-потомок 56
  - рефлексивная 279
  - Сессия 395
  - Символ регистровый 153
  - Синтаксис SQL 69
  - Система кодирования 79
  - Сканирование таблицы 408
  - Слово
    - ключевое (зарезервированное) 70
  - Сортировка
    - восходящая 101
    - нисходящая 101
    - по нескольким столбцам 102
    - по одному столбцу 101
    - по столбцам, заданным относительным местоположением 104
  - Ссылочная целостность 51, 383
  - Столбец
    - агрегатный 206
    - группирующий 206
    - обнуляемый 91
    - производный 138
  - Строка
    - висячая 51
    - пустая 78
  - Строки и столбцы
    - неупорядоченные 45
  - Строковый тип данных 77
  - СУБД 11
    - Microsoft Access 27
    - режим запросов ANSI SQL 26
    - Microsoft SQL Server 30
    - SQL Query Analyzer 30
    - MySQL 36
    - Oracle 33
    - PostgreSQL 38
  - Схема
    - базы данных 47
    - в смысле стандарта ANSI 47
- T**
- Таблица
    - базовая 395
    - виртуальная 411
    - временная 395
      - глобальная 395
      - локальная 395
    - займствованная 411
    - изменение структуры 401
    - копия 395
    - низшего уровня 411
    - образ 395
    - пустая 44
    - реляционной базы данных 43
    - ссылающаяся на себя 51
    - удаление 404
  - Таблица истинности 114
  - Таблицы
    - пользователя 46
    - системные 46
  - Термины
    - нереляционных моделей 43
    - реляционной модели 43
    - стандарта ANSI SQL 43
  - Тип данных 75
    - битовый 79
      - BIT 79
      - BIT VARYING 79
    - действительный числовой 82
      - DOUBLE PRECISION 83
      - FLOAT 83
      - REAL 83
    - мантисса 82
    - порядок 82
  - интервальный 88
    - интервал day-time 88
    - интервал year-month 88
  - календарный 84
    - DATE 84, 85
    - TIME 84, 85
    - TIME WITH TIME ZONE 84, 85
    - TIMESTAMP 84, 85
    - TIMESTAMP WITH TIME ZONE 84, 85
    - поля типа DATETIME 86
  - пользовательский 394
  - строковый 77
    - CHARACTER 77
    - CHARACTER VARYING 77
-



NATIONAL CHARACTER 77  
 NATIONAL CHARACTER VARYING 77  
 точный числовой 80  
   DECIMAL 81  
   INTEGER 81  
   NUMERIC 81  
   SMALLINT 81  
   масштаб 80  
   точность 80  
 Типовая база данных  
   таблица authors 64  
   таблица publishers 65  
   таблица royalties 68  
   таблица title\_authors 67  
   таблица titles 66  
 Точка сохранения 429  
 Точный числовой тип данных 80  
 Транзакция 425  
   commit 426  
   log 426  
   roll back 426  
   выполнение 426  
   журнал 426  
   откат 426  
   точка сохранения 429

## У

Упорядочивание  
   лексикографическое 76  
 Условие  
   объединенное 114  
   поиска 109  
   сравнения 110

## Ф

Форма  
   вторая нормальная (2НФ) 58  
   третья нормальная (3НФ) 60  
 Функция 137  
   ADD\_MONTHS() 168  
   BIT\_LENGTH() 160  
   CAST() 173  
   CHAR\_LENGTH 160  
   CHARACTER\_LENGTH() 159  
   COALESCE() 184

CONCAT() 148  
 CURRENT\_DATE() 169  
 CURRENT\_TIME() 169  
 CURRENT\_TIMESTAMP() 169  
 CURRENT\_USER 171  
 DATE\_ADD() 168  
 DATE\_SUB() 168  
 DATEDIFF() 168  
 DATEPART() 168  
 EXTRACT() 166  
 GETDATE() 170  
 INSTR() 163  
 LEN() 160  
 LENGTH() 160  
 LOCATE() 164  
 LOWER() 153  
 LPAD() 158  
 NULLIF() 185  
 OCTET\_LENGTH() 160  
 POSITION() 161  
 RPAD() 158  
 SESSION\_USER 172  
 SUBSTRING() 149  
 SYS\_CONTEXT 172  
 SYSDATE 170  
 SYSTEM\_USER 172  
 TRIM() 155  
   BOTH, опция 155  
   LEADING, опция 155  
   TRAILING, опция 155  
 UPPER() 153  
 агрегатная 188  
   AVG() 198  
   COUNT() 200  
   MAX() 194  
   MIN() 192  
   SUM() 196  
   опция DISTINCT 202  
 аргумент 137  
 перегружаемая 166

## Ш

Шаблон  
   значений 122  
   переключенный 125

**Я**

## Язык

- встроенный 14
- естественный 12
- интерактивный 14
- непроцедурный 12
- неформальный 12
- программирования 11
- процедурный 12
- со свободным форматом предложений 73
- формальный 11

**A**

- ADD\_MONTHS() 168
- ALL 100
- ALTER TABLE 401
- AND 114
- ANSI 15
- ANSI SQL 1992 15
- AS 96, 222
- ASC 102
- AVG() 198

**B**

- BETWEEN 128
- BIT\_LENGTH() 160

**C**

- CASE 179
- CAST() 173
- CHAR\_LENGTH 160
- CHARACTER\_LENGTH() 159
- CHECK 392
- COALESCE() 184
- COMMIT 427
- CONCAT() 148
- CONSTRAINT 370
- COUNT() 200
- CREATE DATABASE 432
- CREATE INDEX 407
- CREATE TABLE 368

- CREATE VIEW 414
- CURRENT\_DATE() 169
- CURRENT\_TIME() 169
- CURRENT\_TIMESTAMP() 169
- CURRENT\_USER 171

**D**

- DATE\_ADD() 168
- DATE\_SUB() 168
- DATEDIFF() 168
- DATEPART() 168
- DECODE() 183
- DEFAULT 376
- DELETE 363
- DESC 102
- DISTINCT 99, 202
- DROP INDEX 409
- DROP TABLE 404
- DROP VIEW 424

**E**

- ESCAPE 125
- EXCEPT 297
- EXTRACT() 166

**F**

- FROM 93

**G**

- GETDATE() 170
- GROUP BY 206

**H**

- HAVING 215

**I**

- IN 131
- Index 405
- INNER JOIN 241
- INSERT 352
- INSTR() 163
- INTERSECT 295
- IS NULL 134

---

**J**

JOIN  
  CROSS 225, 233  
  FULL OUTER 225  
  INNER 225, 241  
  LEFT OUTER 225, 265  
  NATURAL 225, 236  
  RIGHT OUTER 225  
  SELF 225, 279  
  синтаксис объединения 228

**L**

LEFT OUTER JOIN 265  
LEN() 160  
LENGTH() 160  
LIKE 122  
LOCATE() 164  
LOWER() 153  
LPAD() 158

**M**

MAX() 194  
MIN() 192

**N**

NATURAL JOIN 236  
NOT 114  
NOT NULL 374  
NULL 374  
NULLIF() 185

**O**

OCTET\_LENGTH() 160  
OR 114  
ORDER BY 101  
  ASC 102  
  DESC 102

**P**

POSITION() 161  
PRIMARY KEY 379

**R**

ROLLBACK 427  
RPAD() 158

**S**

SELECT 92  
SELF JOIN 279  
SESSION\_USER 172  
SQL  
  семантика 12  
  синтаксис 12  
SUBSTRING() 149  
SUM() 196  
SYS\_CONTEXT 172  
SYSDATE 170  
SYSTEM\_USER 172

**T**

Transaction 425  
  commit 426  
  log 426  
  roll back 426  
TRIM() 155  
  BOTH, опция 155  
  LEADING, опция 155  
  TRAILING, опция 155

**U**

UNION 286  
UPDATE 358  
UPPER() 153  
USING 232

**V**

View 411

**W**

WHERE  
  в предложении SELECT 109  
  синтаксис объединения 228

Крис Фиайли

## SQL

Главный редактор *Мовчан Д. А.*  
dm@dmk-press.ru

Перевод с английского *Хаванов А. В.*

Научный редактор *Нилов М. В.*

Выпускающий редактор *Космачева Н. А.*

Верстка *Пискунова Л. П.*

Графика *Салимонов Р. В.*

Дизайн обложки *Дудатий А. М.*

Подписано в печать 30.06.2003. Формат 70×100  $\frac{1}{16}$ .

Гарнитура «Миниатюра». Печать офсетная.

Усл. печ. л. 37,05. Тираж 1000 экз. Зак. №

Издательство «ДМК Пресс»

Web-сайт издательства: [www.dmk-press.ru](http://www.dmk-press.ru)

Internet-магазин: [www.aliants-kniga.ru](http://www.aliants-kniga.ru)

Отпечатано на ордена Трудового Красного Знамени

ГУП Чеховский полиграфический комбинат

Министерства Российской Федерации по делам печати,  
телерадиовещания и средств массовых коммуникаций

142300, г. Чехов Московской области

Тел. (272) 71-336, факс (272) 62-536