

Об авторе

Аллен Тейлор — ветеран компьютерной индустрии с тридцатилетним стажем и автор более чем тридцати книг по компьютерной тематике. Читает лекции по базам данных, компьютерным сетям, передовым технологиям, астрономии и предпринимательству. Ведет интернет-курсы по разработке баз данных. Если вас интересует, чем занимается Аллен в настоящее время, зайдите на его сайт www.DatabaseCentral.Info. Можете связаться с ним и по электронной почте, написав письмо по адресу allen.taylor@ieee.org.

Содержание

Об авторе	17
Введение	18
Об этой книге	18
Для кого предназначена книга	18
Пиктограммы, используемые в книге	19
Что дальше	19
Ждем ваших отзывов!	20
Часть I. Знакомство с SQL	21
Глава 1. Основы реляционных баз данных	23
Работа с данными	23
Что такое база данных	24
Размер и сложность базы данных	25
Что такое СУБД	25
Плоские файлы	26
Модели баз данных	28
Реляционная модель	28
Компоненты реляционной базы данных	28
Отношения	29
Представления	30
Схемы, домены и ограничения	32
Объектная модель бросает вызов реляционной	33
Объектно-реляционная модель	33
Вопросы проектирования баз данных	34
Глава 2. Основы SQL	35
Что такое SQL	35
Немного истории	36
Инструкции SQL	37
Зарезервированные слова	39
Типы данных	39
Целочисленные типы	39
Числа с плавающей запятой	41
Символьные строки	42
Двоичные строки	44
Логические данные	45
Значения даты и времени	45
Интервалы	47
Тип XML	47
Тип ROW	49
Типы коллекций	50
Типы REF	51

Определяемые пользователем типы	52
Перечень типов данных	54
Пустые значения	55
Ограничения	56
Использование SQL в архитектуре клиент/сервер	57
Сервер	57
Клиент	58
Использование SQL в Интернете и локальных сетях	58
Глава 3. Компоненты SQL	61
Язык определения данных	61
Когда “Просто сделай это!” — не лучший совет	62
Создание таблиц	63
Создание представлений	64
Объединение таблиц в схемы	69
Заказ по каталогу	70
Инструкции DDL	71
Язык манипулирования данными	72
Выражения	73
Предикаты	75
Логические связи	76
Итоговые функции	77
Подзапросы	78
Язык управления данными	78
Транзакции	78
Пользователи и привилегии доступа	79
Ограничения ссылочной целостности угрожают вашим данным	82
Делегирование ответственности за безопасность	83
Часть II. Использование SQL для создания баз данных	85
Глава 4. Создание простой базы данных	87
Создание простой базы данных средствами СУБД	88
Что хранить в базе данных	88
Создание таблицы базы данных	88
Изменение структуры таблицы	95
Создание индекса	96
Удаление таблицы	99
Создание таблицы POWER средствами SQL	100
Создание SQL-запросов в Microsoft Access	100
Создание таблицы	102
Создание индекса	105
Изменение структуры таблицы	105
Удаление таблицы	106
Удаление индекса	106
Переносимость	106

Глава 5. Создание многотабличной базы данных	109
Проектирование базы данных	109
Действие 1: определение объектов	109
Действие 2: идентификация таблиц и столбцов	110
Действие 3: точное определение таблиц	111
Домены, символьные наборы, схемы сортировки и трансляции	114
Ускорение работы базы данных с помощью ключей	114
Работа с индексами	117
Что такое индекс	117
Зачем нужен индекс	118
Поддержка индекса	118
Обеспечение целостности данных	119
Логическая целостность	120
Доменная целостность	121
Ссылочная целостность	121
Когда вы думаете, что все хорошо	124
Потенциальные проблемы	125
Ограничения	126
Нормализация базы данных	129
Аномалии изменения и нормальные формы	129
Первая нормальная форма	131
Вторая нормальная форма	132
Третья нормальная форма	133
Доменно-ключевая нормальная форма (ДКНФ)	134
Ненормальная форма	135
Часть III. Хранение и извлечение данных	137
Глава 6. Манипулирование содержимым базы данных	139
Извлечение данных	139
Создание представлений	141
Создание представлений из таблиц	141
Создание представления с условием отбора	142
Создание представления с модифицированным атрибутом	143
Обновление представлений	144
Добавление новых данных	145
Добавление данных в виде отдельных записей	145
Добавление данных только в выбранные столбцы	146
Добавление в таблицу группы строк	147
Обновление существующих данных	149
Перемещение данных	152
Удаление устаревших данных	153
Глава 7. Обработка темпоральных данных	155
Моменты и периоды времени в стандарте SQL:2011	155
Использование таблиц с периодами прикладного времени	157
Назначение первичных ключей в таблицах с периодами прикладного времени	159

Применение ограничений ссылочной целостности к таблицам с периодами прикладного времени	160
Формирование запросов к таблицам с периодами прикладного времени	160
Работа с системно-версионными таблицами	161
Назначение первичных ключей в системно-версионных таблицах	163
Применение ссылочных ограничений к системно-версионным таблицам	164
Формирование запросов к системно-версионным таблицам	164
Ослеживание данных с помощью битемпоральных таблиц	165
Глава 8. Обработка значений	167
Значения	167
Значения типа записи	168
Литеральные значения	168
Переменные	169
Специальные переменные	171
Ссылки на столбцы	172
Выражения со значением	173
Строковые выражения	173
Числовые выражения	174
Выражения со значением даты/времени	174
Интервальные выражения	174
Условные выражения	175
Функции	175
Статистические вычисления с помощью итоговых функций	175
Функции преобразований	178
Глава 9. Использование сложных выражений	189
Условные выражения CASE	189
Использование выражения CASE с условиями отбора	190
Использование выражения CASE со значениями	192
Специальное выражение CASE — NULLIF	194
Еще одна специальная форма выражения CASE — COALESCE	195
Преобразование типов данных с помощью выражения CAST	196
Использование выражения CAST в SQL	197
Использование выражения CAST при взаимодействии SQL и языка приложения	197
Выражения со значением типа записи	198
Глава 10. Выбор нужных данных	201
Уточняющие предложения	201
Предложение FROM	203
Предложение WHERE	203
Предикаты сравнения	204
Предикат BETWEEN	205
Предикаты IN и NOT IN	206
Предикаты LIKE и NOT LIKE	207
Предикат SIMILAR	208
Предикат NULL	209
Предикаты ALL, SOME и ANY	210

Предикат EXISTS	212
Предикат UNIQUE	213
Предикат DISTINCT	213
Предикат OVERLAPS	213
Предикат MATCH	214
Правила ссылочной целостности и предикат MATCH	215
Логические связи	218
Логическая связка AND	218
Логическая связка OR	219
Логическая связка NOT	219
Предложение GROUP BY	219
Предложение HAVING	221
Предложение ORDER BY	222
Использование инструкции FETCH для ограничения выборки	224
Использование оконных функций для создания результирующего множества	225
Разделение окна на участки с помощью функции NTILE	226
Навигация в пределах окна	226
Вложение оконных функций	228
Выполнение расчетов по группам строк	229
Глава 11. Использование реляционных операторов	231
Оператор UNION	231
Операция UNION ALL	233
Операция CORRESPONDING	233
Оператор INTERSECT	234
Оператор EXCEPT	236
Операторы объединения	236
Простое объединение	236
Объединение, основанное на равенстве	238
Перекрестное объединение	240
Естественное объединение	240
Условное объединение	241
Объединение по именам столбцов	241
Внутреннее объединение	242
Внешнее объединение	242
Объединение слиянием	246
Предложения ON и WHERE	251
Глава 12. Вложенные запросы	253
Назначение подзапросов	254
Вложенные запросы, возвращающие наборы строк	254
Вложенные запросы, возвращающие одно значение	258
Использование подзапросов вместе с предикатами ALL, SOME и ANY	260
Вложенные запросы как средство проверки на существование	261
Другие коррелированные подзапросы	263
Инструкции UPDATE, DELETE и INSERT	266
Регистрация изменений с помощью конвейерных DML-операций	268

Глава 13. Рекурсивные запросы	271
Что такое рекурсия	271
Хьюстон, у нас проблема	272
Сбой недопустим	272
Что такое рекурсивный запрос	274
Где можно применить рекурсивный запрос	274
Лобовое решение	275
Экономия времени с помощью рекурсивного запроса	276
Где еще можно использовать рекурсивные запросы	278
Часть IV. Управление операциями	279
Глава 14. Обеспечение безопасности базы данных	281
Язык управления данными	281
Уровни доступа пользователей	282
Администратор базы данных	282
Владельцы объектов базы данных	283
Понятие публичности	283
Предоставление полномочий пользователям	284
Роли	285
Вставка данных	286
Просмотр данных	286
Модификация табличных данных	287
Удаление устаревших строк из таблицы	287
Использование ссылок на связанные таблицы	287
Использование доменов	288
Инициирование выполнения инструкций SQL	290
Предоставление уровневых полномочий	290
Право на предоставление полномочий	291
Отзыв полномочий	292
Совместное использование инструкций GRANT и REVOKE	294
Глава 15. Защита данных	295
Угрозы целостности данных	295
Нестабильность платформы	296
Аппаратный сбой	296
Одновременный доступ	296
Уменьшение уязвимости данных	299
Использование SQL-транзакций	299
Транзакция по умолчанию	301
Уровни изоляции	301
Неявная инструкция начала транзакции	303
Инструкция SET TRANSACTION	303
Инструкция COMMIT	304
Инструкция ROLLBACK	304
Блокирование объектов базы данных	304
Резервное копирование данных	305
Точки сохранения и субтранзакции	305
Ограничения в транзакциях	307

Глава 16. Использование SQL в приложениях	311
SQL в приложении	311
Следите за звездочкой	312
Сильные и слабые стороны SQL	312
Сильные и слабые стороны процедурных языков	313
Проблемы, возникающие при совместном использовании SQL с процедурными языками	313
Вставка инструкций SQL в процедурные языки	314
Внедрение кода SQL	314
Модульный язык	316
Объектно-ориентированные инструменты быстрой разработки	318
Использование SQL в приложении Microsoft Access	319
Часть V. Практическое использование SQL	323
Глава 17. Доступ к данным с помощью ODBC и JDBC	325
ODBC	325
Интерфейс ODBC	326
Компоненты ODBC	326
ODBC в среде клиент/сервер	327
ODBC и Интернет	327
Серверные расширения	328
Клиентские расширения	329
ODBC и локальные сети	330
JDBC	330
Глава 18. Работа с XML-данными	333
Как XML соотносится с SQL	333
Тип данных XML	334
Когда использовать тип данных XML	334
Когда не стоит использовать тип данных XML	335
Преобразование данных из формата SQL в формат XML и наоборот	335
Преобразование наборов символов	336
Преобразование идентификаторов	336
Преобразование типов данных	337
Преобразование таблиц	337
Обработка пустых значений	338
Создание схемы XML	338
Функции SQL для выполнения действий над XML-данными	339
Оператор XMLDOCUMENT	340
Оператор XMLELEMENT	340
Оператор XMLFOREST	340
Функция XMLCONCAT	340
Функция XMLAGG	341
Функция XMLCOMMENT	341
Функция XMLPARSE	342
Функция XMLPI	342

Функция XMLQUERY	342
Функция XMLCAST	343
Предикаты	343
Предикат DOCUMENT	343
Предикат CONTENT	343
Предикат XMLEXISTS	344
Предикат VALID	344
Преобразование данных XML в таблицы SQL	345
Преобразование нестандартных типов данных в XML	346
Домены	346
Индивидуальные типы UDT	347
Записи	347
Массивы	348
Мультимножества	349
Содружество SQL и XML	350
Часть VI. Расширенные возможности SQL	351
Глава 19. Обработка наборов данных с помощью курсоров	353
Объявление курсора	354
Выражение запроса	354
Предложение ORDER BY	355
Разрешение обновления	356
Чувствительность	356
Перемещаемость	357
Открытие курсора	358
Извлечение данных из отдельных строк	359
Синтаксис	359
Ориентация перемещаемого курсора	360
Позиционные инструкции DELETE и UPDATE	360
Закрытие курсора	361
Глава 20. Процедурное программирование и хранимые модули	363
Составные инструкции	363
Атомарность	364
Переменные	365
Курсоры	365
Состояния	365
Обработка состояний	366
Необрабатываемые состояния	368
Присвоение	368
Управляющие блоки	369
Конструкция IF...THEN...ELSE...END IF	369
Конструкция CASE...END CASE	369
Цикл LOOP...END LOOP	370
Инструкция LEAVE	371
Цикл WHILE...DO...END WHILE	371

Цикл REPEAT...UNTIL...END REPEAT	372
Цикл FOR...DO...END FOR	372
Инструкция ITERATE	372
Хранимые процедуры	373
Хранимые функции	374
Полномочия	375
Хранимые модули	375
Глава 21. Обработка ошибок	377
Переменная SQLSTATE	377
Директива WHENEVER	378
Области диагностики	379
Заголовок области диагностики	380
Информационная область диагностики	381
Пример нарушения ограничения	383
Добавление новых ограничений в уже созданную таблицу	384
Интерпретация информации, возвращаемой переменной SQLSTATE	384
Обработка исключений	385
Глава 22. Триггеры	387
Область применения триггеров	387
Создание триггера	387
Триггеры инструкций и строк	388
Когда срабатывает триггер	388
Иницилируемая SQL-инструкция	389
Пример определения триггера	389
Срабатывание последовательности триггеров	389
Ссылки на старые и новые значения	390
Срабатывание нескольких триггеров в одной таблице	391
Часть VII. Великолепные десятки	393
Глава 23. Десять самых распространенных ошибок	395
Уверенность в том, что клиенты знают, чего хотят	395
Игнорирование масштабов проекта	396
Учет только технических факторов	396
Отсутствие обратной связи с клиентами	396
Использование только своей любимой среды разработки	396
Использование только своей любимой системной архитектуры	397
Проектирование таблиц баз данных отдельно друг от друга	397
Отказ от консультаций с другими специалистами	397
Игнорирование бета-тестирования	398
Отказ от создания документации	398
Глава 24. Десять советов по извлечению данных	399
Проверяйте структуру базы данных	399
Испытайте запросы на тестовой базе данных	399

Дважды проверяйте запросы, включающие операторы JOIN	400
Трижды проверяйте запросы с подзапросами	400
Подводите итоги, используя предложение GROUP BY	400
Внимательно относитесь к ограничениям из предложения GROUP BY	400
Используйте круглые скобки с операторами AND, OR и NOT	401
Контролируйте полномочия на получение данных	401
Регулярно выполняйте резервное копирование своих баз данных	401
Тщательно обрабатывайте ошибочные состояния	402
Приложение. Зарезервированные слова SQL:2011	403
Предметный указатель	405

Введение

Добро пожаловать в мир разработки баз данных, где царит SQL — структурированный язык запросов! Существует множество систем управления базами данных (СУБД), предназначенных для разных платформ. Различия между этими системами велики, однако их объединяет одно: все они поддерживают SQL как средство доступа к данным и управления ими. Если вы знаете этот язык, то сможете создавать реляционные базы данных и извлекать из них полезную информацию.

Об этой книге

Системы управления базами данных играют важнейшую роль в большинстве организаций. Люди часто думают, что создание и обслуживание таких систем — удел профессионалов в области баз данных, которым открыты знания, недоступные простым смертным. Книга, которую вы читаете, развеет этот миф. Прочитав книгу, вы:

- ✓ узнаете об истории баз данных;
- ✓ получите представление о структуре СУБД;
- ✓ откроете для себя основные функциональные компоненты SQL;
- ✓ научитесь создавать базы данных;
- ✓ научитесь защищать базы данных от потенциальных опасностей;
- ✓ научитесь работать с данными;
- ✓ узнаете, как извлечь из базы данных нужную информацию.

Цель книги заключается в том, чтобы научить вас создавать реляционные базы данных и извлекать из них нужную информацию средствами SQL. Этот язык принят в качестве международного стандарта и повсеместно применяется для создания и обслуживания реляционных баз данных. В новом издании книги рассмотрена последняя версия стандарта — SQL:2011.

В книге не будет говориться о том, как проектировать базы данных. Предполагается, что у вас уже есть подходящий проект базы данных, поэтому будет показано, как реализовать этот проект с помощью SQL. Если же вы считаете, что существующий проект недостаточно хорош, обязательно исправьте его до того, как приступить к реализации. Чем раньше вы обнаружите недостатки в проекте, тем меньше усилий придется затратить на их исправление.

Для кого предназначена книга

Если вам нужно хранить данные в СУБД или извлекать их оттуда, то практическое знание SQL поможет вам выполнять данную работу эффективнее. Чтобы применять SQL, не нужно быть программистом и знать такие языки, как Java, C или Basic. Синтаксис SQL сходен с синтаксисом английского языка.

Если вы все же программист, то сможете внедрить код SQL в свои программы. Он привнесет в обычные языки программирования мощный аппарат манипулирования данными. В книге будет показано, что именно нужно знать, чтобы реализовать в своих программах богатый набор возможностей, предоставляемых SQL.

Пиктограммы, используемые в книге



Эти советы помогут вам сэкономить время и уберегут от неприятностей.



Уделяйте особое внимание информации, помеченной данной пиктограммой, — возможно, однажды она вам пригодится.



Если будете учитывать советы, помеченные этой пиктограммой, то избежите больших неприятностей. Можете игнорировать их, но на свой страх и риск.



Данная пиктограмма указывает на описание технических деталей, которые могут быть интересны, но не особо существенны в рассматриваемой теме.

Что дальше

Базы данных — самый прекрасный инструмент, когда-либо изобретенный человечеством для отслеживания важной информации. Если вы его освоите и заставите с помощью SQL выполнять свои распоряжения, то перед вами откроются широчайшие возможности. Сотрудники будут обращаться к вам, когда им потребуется найти нужные для работы данные. Руководство станет прислушиваться к вашим советам. Молодежь будет просить у вас автограф. И, что важнее всего, вы поймете, причем во всех деталях, как функционируют информационные потоки в вашей организации.

Часть III

Хранение и извлечение данных



В этой части...

- ✓ Управление данными
- ✓ Контроль времени
- ✓ Обработка значений
- ✓ Создание запросов

Манипулирование содержимым базы данных

В этой главе...

- Работа с данными
- Извлечение из таблицы нужных данных
- Отображение информации, полученной из одной или нескольких таблиц
- Обновление информации, находящейся в таблицах и представлениях
- Добавление новой строки в таблицу
- Изменение всех или части данных, находящихся в строке таблицы
- Удаление строки таблицы

В главах 3 и 4 вы узнали о том, что для обеспечения целостности информации, хранящейся в базе данных, очень важно сделать все возможное, чтобы ее структура была безупречна. Впрочем, для пользователя интерес представляет не структура базы данных, а ее содержимое, т.е. сами данные. И с ними пользователь должен иметь возможность выполнять определенный набор действий: добавлять их в таблицы, извлекать и отображать, изменять, а также удалять из таблиц. В принципе, манипулировать данными достаточно просто. Вы можете добавить в таблицу одну или сразу несколько строк данных. Изменение, удаление и извлечение строк из таблиц баз данных также не представляют особого труда. Главная трудность при манипуляциях с базами данных — *отобрать* строки, которые следует изменить, удалить или получить. Нужные данные могут находиться в базе данных, содержащей гигантский массив совершенно *ненужной* вам информации. Если вы сможете с помощью инструкции `SELECT` точно указать, что именно вам нужно, всю остальную работу по поиску данных компьютер возьмет на себя. Лично мне кажется, что манипулирование данными с помощью SQL проще пареной репы — как извлечение, так и обновление и удаление. Возможно, с “репой” я несколько переборщил, так что начнем с простой операции извлечения данных.

Извлечение данных

Задачи, которые выполняют пользователи с базами данных, чаще всего сводятся к извлечению из них необходимой информации. К примеру, вы захотите получить содержимое одной конкретной строки, находящейся в таблице среди тысяч других. Или вы решите просмотреть строки, удовлетворяющие некоторому условию или комбинации условий. Не исключено, что вам потребуется извлечь из таблицы все ее строки. Для выполнения всех этих задач служит всего одна инструкция — `SELECT`.

Самый простой способ использования инструкции SELECT состоит в извлечении *всех* данных, хранящихся во *всех* строках конкретной таблицы. Для этого используется следующий синтаксис:

```
SELECT * FROM CUSTOMER ;
```



Звездочка (*) — это символ обобщения, подразумевающий *все*. В приведенном примере этот символ заменяет список имен всех столбцов таблицы CUSTOMER. В результате выполнения этой инструкции на экран выводятся все данные, находящиеся во всех строках и столбцах этой таблицы.

Инструкции SELECT могут быть намного сложнее, чем приведенная в данном примере. Некоторые из них могут быть настолько сложными, что в них очень трудно разобраться. Это связано с возможностью присоединения к базовой инструкции множества уточняющих предложений. Подробно об уточняющих предложениях вы узнаете в главе 10. В этой же главе мы вкратце поговорим о предложении WHERE — самом распространенном способе ограничения количества строк, возвращаемых инструкцией SELECT.

Инструкция SELECT с предложением WHERE имеет следующий общий вид.

```
SELECT список_столбцов FROM имя_таблицы  
WHERE условие;
```

Список_столбцов определяет, какие столбцы таблицы следует отобразить для вывода, поскольку только они отобразятся на экране. В предложении FROM определяется имя той таблицы, столбцы которой требуется отобразить. А предложение WHERE исключает те строки, которые не удовлетворяют заданному в нем условию. Условие может быть простым (например, WHERE CUSTOMER_STATE = 'NH', отбирающее записи, относящиеся только к штату Нью-Гэмпшир) или составным (например, WHERE CUSTOMER_STATE = 'NH' AND STATUS = 'Active', включающее в себя комбинацию двух условий).

Следующий пример показывает, как выглядит составное условие в инструкции SELECT.

```
SELECT FirstName, LastName, Phone FROM CUSTOMER  
WHERE State= 'NH'  
AND Status='Active' ;
```

Эта инструкция возвращает имена, фамилии и телефонные номера всех активных клиентов, живущих в штате Нью-Гэмпшир. Ключевое слово AND означает, что для того, чтобы строка была возвращена, она должна соответствовать обоим условиям: State = 'NH' и Status = 'Active'.

SQL в интерактивных инструментах

Инструкция SELECT — не единственное средство извлечения информации из базы данных. В СУБД, как правило, для манипуляций с данными имеются встроенные интерактивные средства. С помощью этих инструментов можно добавлять записи в базу, изменять и удалять их из нее, а также отправлять запросы к базе.

Многие СУБД предоставляют пользователю выбор между интерактивными средствами и режимом SQL. В большинстве случаев предлагаемые инструменты не покрывают всех воз-

можностей, предлагаемых SQL. Если доступные интерактивные средства вас не устраивают, вам придется вручную писать инструкции SQL. По этой причине ознакомиться с SQL просто необходимо, даже если вы предпочитаете использовать графический интерфейс. Этот язык позволит выполнить операции, слишком сложные для интерактивных инструментов, поэтому исключительно важно понять, как он работает и что можно сделать с его помощью.

Создание представлений

Структура базы данных, спроектированной в соответствии с разумными принципами, включая и соответствующую нормализацию (см. главу 5), обеспечивает максимальную целостность данных. Однако такая структура данных часто не позволяет обеспечить наилучший способ их просмотра. Одни и те же данные могут использоваться разными приложениями, и у каждого из них может быть своя специализация. Одной из самых востребованных функций SQL является вывод данных в виде представлений, структура которых отличается от структуры таблиц базы, в которой реально хранятся эти данные. Таблицы, столбцы и строки которых используются при создании представления, называются *базовыми*. В главе 3 говорилось о представлениях как о части языка определения данных (DDL). В этом разделе мы будем рассматривать представления в контексте извлечения данных и манипуляции ими.

Инструкция `SELECT` всегда возвращает результат в виде виртуальной таблицы. *Представление* же само является особой разновидностью виртуальных таблиц; оно отличается от других виртуальных таблиц тем, что в метаданных базы данных хранится его определение. Это придает представлению те признаки постоянства, которыми не обладают другие виртуальные таблицы.



Работать с представлением можно так же, как и с настоящей таблицей базы данных. Различие состоит в том, что данные представления не являются физически независимой частью базы данных. Представление извлекает данные из одной или множества таблиц, на которых оно базируется. В каждом приложении могут быть свои собственные, непохожие друг на друга представления, но созданные на основе одних и тех же данных.

Рассмотрим базу данных VetLab, описанную в главе 5. Она состоит из пяти таблиц: `CLIENT` (фирма-клиент), `TESTS` (анализы), `EMPLOYEE` (сотрудник), `ORDERS` (заказы) и `RESULTS` (результаты). Предположим, что главному менеджеру по маркетингу компании VetLab необходимо посмотреть, из каких штатов приходят заказы в эту компанию. Часть этой информации находится в таблице `CLIENT`, а часть — в `ORDERS`. В то же время сотруднику службы контроля качества нужно сравнить дату оформления заказа одного из анализов и дату получения его окончательного результата. Для этого ему требуются данные из таблиц `ORDERS` и `RESULTS`. Для каждого из этих сотрудников можно создать отдельные представления, содержащие только нужные им данные.

Создание представлений из таблиц

Для менеджера по маркетингу можно создать представление, показанное на рис. 6.1. Это представление создается с помощью следующей инструкции.

```
CREATE VIEW ORDERS_BY_STATE
  (ClientName, State, OrderNumber)
AS SELECT CLIENT.ClientName, State, OrderNumber
FROM CLIENT, ORDERS
WHERE CLIENT.ClientName = ORDERS.ClientName ;
```

Итак, в наше представление входят три столбца: `ClientName` (название фирмы-клиента), `State` (штат) и `OrderNumber` (номер заказа). Поле `ClientName` находится как в таблице `CLIENT`, так и в `ORDERS` и используется для связи между ними. Представление получает информацию из столбца `State` таблицы `CLIENT` и берет для каждого заказа значение из столбца `OrderNumber` таблицы `ORDERS`. В приведенном примере имена столбцов объявляются явным образом.

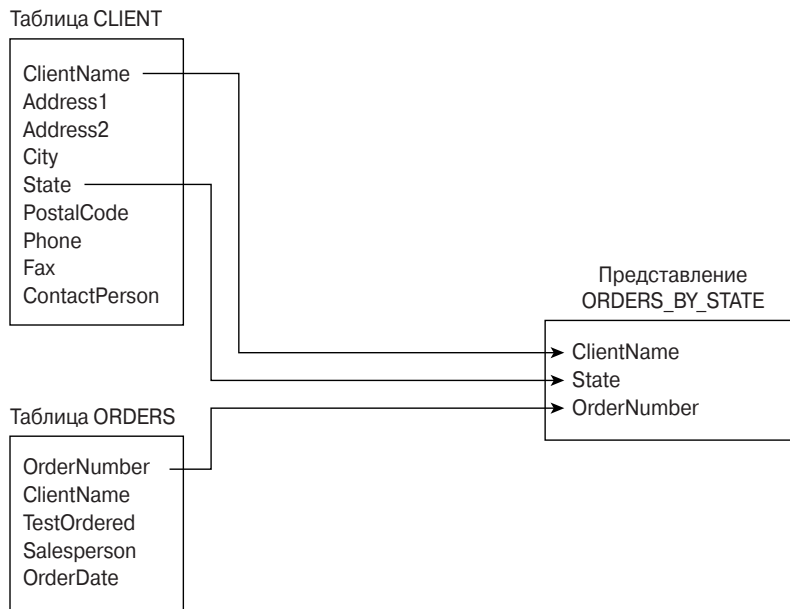


Рис. 6.1. Представление `ORDERS_BY_STATE`, предназначенное для менеджера по маркетингу

Обратите внимание на то, что перед полем `ClientName` указано имя содержащей его таблицы, а для полей `State` и `OrderNumber` — нет. Дело в том, что поле `State` имеется только в таблице `CLIENT`, а поле `OrderNumber` — только в таблице `ORDERS`, поэтому никакой двусмысленности нет. Но поле `ClientName` входит в состав двух таблиц, `CLIENT` и `ORDERS`, поэтому необходим дополнительный (уточняющий) идентификатор.



Впрочем, если имена столбцов в представлении совпадают с именами столбцов исходных таблиц, то такое детальное объявление использовать не обязательно. Пример, приведенный в следующем разделе, демонстрирует похожую инструкцию `CREATE VIEW`, в которой имена столбцов для представления не указываются явно, а только подразумеваются.

Создание представления с условием отбора

Как видно на рис. 6.2, для сотрудника службы контроля качества создано представление, отличающееся от того, которое использует менеджер по маркетингу.

Ниже приведен код, с помощью которого создано представление, показанное на рис. 6.2.

```
CREATE VIEW REPORTING_LAG
AS SELECT ORDERS.OrderNumber, OrderDate, DateReported
FROM ORDERS, RESULTS
WHERE ORDERS.OrderNumber = RESULTS.OrderNumber
AND RESULTS.PreliminaryFinal = 'F' ;
```

В приведенном представлении содержится информация из таблицы ORDERS о датах заказов и из таблицы RESULTS о датах получения результатов анализов. В этом представлении отображаются только строки, у которых в столбце PreliminaryFinal, взятом из таблицы RESULTS, находится значение 'F' (что соответствует окончательному результату). Обратите внимание на то, что список явно задаваемых столбцов, включенный в определение представления ORDERS_BY_STATE, совершенно не обязателен. Представление REPORTING_LAG прекрасно работает и без такого списка.

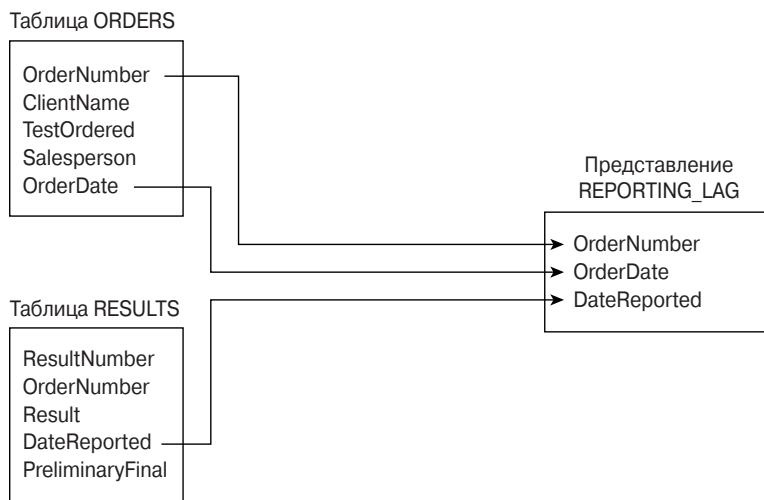


Рис. 6.2. Представление *REPORTING_LAG*, предназначенное для сотрудника службы контроля качества

Создание представления с модифицированным атрибутом

В примерах из двух предыдущих разделов предложения инструкции `SELECT` содержат только имена столбцов. Однако инструкция `SELECT` может включать и выражения. Предположим, владелец лаборатории VetLab в честь своего дня рождения хочет предоставить всем клиентам 10-процентную скидку. Это можно реализовать путем создания представления `BIRTHDAY` (день рождения) на основе двух таблиц, `ORDERS` и `TESTS`.

```

CREATE VIEW BIRTHDAY
(ClientName, Test, OrderDate, BirthdayCharge)
AS SELECT ClientName, TestOrdered, OrderDate,
StandardCharge * .9
FROM ORDERS, TESTS
WHERE TestOrdered = TestName ;
  
```

Обратите внимание на то, что в представлении `BIRTHDAY` второй столбец — `Test` (анализ) — соответствует столбцу `TestOrdered` (заказанный анализ) из таблицы `ORDERS`, который также соответствует столбцу `TestName` (название анализа) из таблицы `TESTS`. Как создается это представление, показано на рис. 6.3.

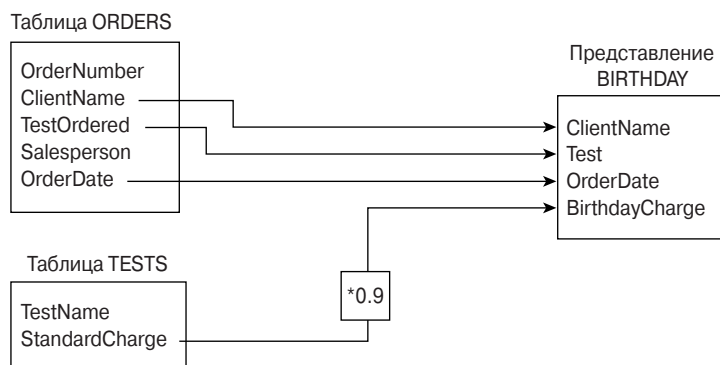


Рис. 6.3. Представление, созданное для демонстрации скидок в честь дня рождения

Представления можно создавать как на основе множества таблиц (мы видели это в предыдущих примерах), так и на основе всего лишь одной таблицы. Если вам не нужны конкретные столбцы и строки какой-либо таблицы, создайте представление, в котором их нет, а затем работайте уже не с самой таблицей, а с представлением. Этот подход защищает пользователя от путаницы и не отвлекает внимание, что бывает при обилии на экране не относящихся к делу данных.



Еще одной причиной создания представлений является обеспечение безопасности исходных таблиц базы данных. Одни столбцы ваших таблиц можно открыть для просмотра, а другие, наоборот, скрыть. Таким образом, можно создать представление только с общедоступными столбцами, открыв к нему широкий доступ, и при этом ограничить доступ к исходным таблицам этого представления. В главе 14 будут рассмотрены вопросы защиты информации в базах данных, и вы узнаете, как управлять правами доступа пользователей к ресурсам.

Обновление представлений

Созданные таблицы автоматически поддерживают возможности вставки, обновления и удаления данных. А вот к представлениям этот тезис относится не всегда. Обновляя представление, вы на самом деле обновляете исходную таблицу. Вот две потенциальные проблемы, возникающие при обновлении представлений.

- ✓ Некоторые представления получают данные из двух и более таблиц. Если вы обновляете такое представление, его базовые таблицы не всегда обновляются корректно.
- ✓ Инструкция `SELECT` представления может содержать выражения. Так как выражения не соответствуют непосредственно полям таблицы, СУБД может не знать, как обновлять их.

Предположим, вы создаете представление, используя следующую инструкцию.

```
CREATE VIEW COMP (EmpName, Pay)
AS SELECT EmpName, Salary+Comm AS Pay
FROM EMPLOYEE ;
```

Не исключено, что вы допустили возможность обновить столбец Pay (оплата), используя такую инструкцию:

```
UPDATE COMP SET Pay = Pay + 100 ;
```

К сожалению, этот подход не работает, потому что в таблице EMPLOYEE нет столбца Pay. Невозможно обновить в представлении то, чего нет в исходной таблице.



Собираясь обновлять представления, не забывайте следующее правило: столбец представления нельзя обновлять, если он не соответствует столбцу его базовой таблицы.

Добавление новых данных

В базе данных каждая таблица появляется на свет пустой, т.е. сразу после создания (с помощью SQL-инструкции или с помощью инструмента RAD) такая таблица является ничем иным, как структурной оболочкой, не содержащей данных. Чтобы таблица стала полезной, в нее необходимо поместить какие-нибудь данные. Эти данные могут уже храниться в компьютере в цифровом виде или вводиться пользователем вручную. Данные в таблицу можно ввести следующим образом.

- ✓ Если данные еще не хранятся в цифровом виде, кто-то должен ввести их построчно. Данные можно также вводить с помощью оптических сканеров и систем распознавания речи (правда, этот метод используется относительно редко).
- ✓ Если данные уже находятся в цифровом виде, но, возможно, не в том формате, который используется в таблицах базы данных, то их нужно сначала преобразовать в соответствующий формат, а затем уже вставлять в базу данных.
- ✓ Если данные уже находятся в нужном цифровом формате, то можно их скопировать в готовом виде в новую базу данных.

В следующем разделе вы узнаете, как занести в таблицу данные, находящиеся в любой из трех названных форм. В зависимости от текущей формы данных их можно занести в базу одной операцией или пошагово, по одной записи. Каждая вводимая запись соответствует одной строке таблицы базы данных.

Добавление данных в виде отдельных записей

В большинстве СУБД поддерживается ввод данных с помощью форм. Другими словами, вам предоставляется возможность создания экранной формы, в которой для каждого табличного столбца из базы данных будет существовать свое поле. По названию поля легко определить, какие данные следует вводить в него. Оператор вручную вводит в форму все данные, соответствующие одной строке. После того как СУБД примет заполненную строку, система очистит форму для ввода следующей. Таким образом, можно ввести в таблицу данные в виде отдельных строк, одну за другой.

Ввод данных с помощью форм прост и менее подвержен ошибкам, чем ввод *списков значений, разделенных запятыми*. Основная проблема ввода данных с помощью форм состоит в том, что сами формы не стандартизированы. В различных СУБД используются собственные методы создания форм. В то же время оператору, занятому вводом данных, совершенно безразлично, как именно создана форма ввода. Можно создать форму, которая будет выглядеть приблизительно одинаково в различных СУБД. (Для оператора это практически незаметно, но разработчику

приложений придется учиться заново каждый раз, когда будут изменяться средства разработки.) В этом методе ввода данных существует еще одна проблема: в некоторых реализациях невозможно полностью проверить правильность вводимых данных.

Самый лучший способ поддержки высокого уровня целостности данных в базе — это не вводить неправильные данные. Предотвратить неправильный ввод можно, применяя ограничения к полям формы ввода. Это гарантирует, что в базу данных попадут только те значения, которые имеют правильный тип данных и входят в заданный диапазон. Естественно, применение ограничений не предотвратит все возможные ошибки, но позволит все же выявить некоторые из них.



Если средства разработки форм в вашей СУБД не позволяют применить полную проверку правильности ввода данных, гарантирующую целостность данных, вам, возможно, придется создать собственную экранную форму, использовать ее для ввода данных в переменные, а затем проверять корректность их значений с помощью программного кода приложения. И только убедившись, что все значения, принятые для записи строки таблицы, являются правильными, программа может добавить эту строку в таблицу с помощью SQL-инструкции `INSERT`.

Для ввода данных, предназначенных для одной строки таблицы базы данных, используется следующий синтаксис инструкции `INSERT`.

```
INSERT INTO таблица_1 [(столбец_1, столбец_2, ..., столбец_n)]
VALUES (значение_1, значение_2, ..., значение_n) ;
```

Квадратные скобки (`[]`) означают, что заключенный в них список имен столбцов не является обязательным. По умолчанию порядок расположения столбцов в списке является таким же, как и в таблице. Если расположить значения, находящиеся после ключевого слова `VALUES`, в том же порядке, в котором столбцы находятся в таблице, то эти элементы попадут в нужное место — неважно, указаны при этом названия столбцов явно или нет. Если же вы хотите расположить вводимые значения в порядке, который не совпадает с порядком следования столбцов в таблице, то тогда последовательность имен столбцов необходимо указать явно, причем она должна совпадать с последовательностью значений, указанных в предложении `VALUES`.

Например, чтобы ввести запись в таблицу `CUSTOMER`, используйте следующий синтаксис.

```
INSERT INTO CUSTOMER (CustomerID, FirstName, LastName,
Street, City, State, Zipcode, Phone)
VALUES (:vcustid, 'Денис', 'Данилкин', 'Белорусская, 38',
'Новосибирск', 'NS', '41156', '(046) 430-5555') ;
```

После ключевого слова `VALUES` (первой в списке) стоит `vcustid` — базовая переменная-счетчик, значение которой программа увеличивает на единицу после ввода в таблицу новой строки. Это гарантирует, что в столбце `CustomerID` не будет дублирования значений. `CustomerID` является первичным ключом этой таблицы, поэтому его значения должны быть уникальными. Остальные значения в списке `VALUES` являются не переменными, а элементами данных, хотя при желании данные для этих столбцов также можно поместить в переменные. Инструкция `INSERT` работает одинаково хорошо с аргументами ключевого слова `VALUES`, выраженными как в виде переменных, так и в виде конкретных значений.

Добавление данных только в выбранные столбцы

Иногда нужно где-то зафиксировать существование объекта, даже если по нему еще нет всех данных. Если для таких объектов у вас есть таблица базы данных, то строку по новому

объекту можно вставить в нее, не заполняя значениями все столбцы этой строки. Если хотите, чтобы таблица была в первой нормальной форме, то обязательно вставлять только те данные, которые позволят отличить новую строку от всех остальных строк этой таблицы, в частности первичный ключ. (О первой нормальной форме см. в главе 5.) Кроме первичного ключа можно вставить все данные, известные об этом объекте на текущий момент. В тех столбцах, куда данные не вводятся, остаются пустые значения (NULL).

Ниже приведен пример частичного ввода строки.

```
INSERT INTO CUSTOMER (CustomerID, FirstName, LastName)
VALUES (:vcustid, 'Денис', 'Данилкин') ;
```

При выполнении этой инструкции в таблицу базы данных вставляется только уникальный идентификационный номер клиента, а также его имя и фамилия. Остальные столбцы этой строки будут содержать значения NULL.

Добавление в таблицу группы строк

Добавлять в таблицу строки одну за другой с помощью инструкции INSERT — довольно утомительное занятие, и даже при использовании эргономичной экранной формы вы очень скоро устанете. Если же у вас есть надежное средство автоматического ввода данных (вместо ввода вручную), вы будете стараться по возможности использовать именно его.

Как правило, автоматический ввод используется тогда, когда данные уже существуют в электронном виде, т.е. кто-то предварительно ввел их в компьютер. Зачем же повторять эту рутинную работу? Перенести данные из одного файла в другой можно при минимальном участии человека. Если вам известны характеристики исходных данных и желаемая структура итоговой таблицы, то компьютер может (в принципе) выполнить такой перенос данных автоматически.

Копирование из внешнего файла

Предположим, вы создаете базу данных для нового приложения. Некоторые из нужных вам данных уже имеются в каком-либо файле. Это может быть плоский файл или таблица базы данных, работающей в отличной от вашей СУБД. Данные могут находиться в коде ASCII, EBCDIC или в каком-нибудь другом закрытом внутреннем формате. Так что же делать?

Хорошо, если формат этих данных окажется одним из широко используемых. Если повезет, у вас будет хороший шанс раздобыть утилиту преобразования этого формата в приемлемый для вашей среды разработки. Если сильно повезет, то для преобразования текущего формата данных будет достаточно встроенных средств среды разработки. Пожалуй, самыми распространенными на персональных компьютерах форматами являются Access, xBASE и MySQL. Если нужные вам данные находятся в одном из этих форматов, преобразование должно пройти легко. Но если формат данных окажется не таким распространенным, то, скорее всего, преобразование придется провести в два этапа.

Перенос всех строк из одной таблицы в другую

Намного проще (чем импортировать внешние данные) извлечь содержимое одной таблицы вашей базы данных и объединить его с содержимым другой таблицы. Если структуры первой и второй таблиц идентичны (т.е. для каждого столбца первой таблицы существует соответствующий столбец во второй, а их типы данных совпадают), то задача решается очень просто. В этом случае содержимое двух таблиц можно объединить с помощью реляционного оператора UNION. В результате получится *виртуальная таблица*, содержащая данные из обеих исходных таблиц. О реляционных операторах, в том числе и о UNION, вы узнаете в главе 11.

Перенос выбранных столбцов и строк из одной таблицы в другую

Нередко бывает так, что структура данных исходной таблицы не соответствует в точности структуре той таблицы, в которую вы собираетесь их поместить. Вполне вероятно совпадение некоторых столбцов — и это могут оказаться как раз те столбцы, которые нужно перенести. Комбинируя инструкции `SELECT` с помощью оператора `UNION`, можно указать, какие именно столбцы исходных таблиц должны войти в итоговую виртуальную таблицу.

Включая в инструкции `SELECT` предложения `WHERE`, можно помещать в виртуальную таблицу только те строки, которые удовлетворяют заданным условиям. Предложения `WHERE` подробно описываются в главе 10.

Предположим, у нас есть две таблицы, `PROSPECT` (потенциальный клиент) и `CUSTOMER` (покупатель), и нам нужно составить список всех жителей штата Мэн, данные о которых находятся в обеих таблицах. В этом случае можно получить виртуальную таблицу с нужной информацией, используя следующий запрос.

```
SELECT FirstName, LastName
FROM PROSPECT
WHERE State = 'ME'
UNION
SELECT FirstName, LastName
FROM CUSTOMER
WHERE State = 'ME'
```

Рассмотрим этот код подробней.

- ✓ Инструкции `SELECT` сообщают о том, что у созданной таблицы будут столбцы `FirstName` (имя) и `LastName` (фамилия).
- ✓ Предложения `WHERE` ограничивают состав строк в создаваемой таблице, отбирая лишь те, у которых в столбце `State` (штат) содержится значение `'ME'` (штат Мэн).
- ✓ Столбца `State` в виртуальной таблице не будет, несмотря на то, что он существует в двух исходных таблицах: `PROSPECT` и `CUSTOMER`.
- ✓ Оператор `UNION` объединяет результаты выполнения первого запроса `SELECT` к таблице `PROSPECT` с результатами выполнения второго запроса `SELECT` к таблице `CUSTOMER`, удаляет все строки-дубликаты, а затем выводит окончательный результат на экран.



Еще один способ копирования информации из одной таблицы базы данных в другую состоит во вложении инструкции `SELECT` в инструкцию `INSERT`. Этот метод (о подзапросах `SELECT` будет подробнее говориться в главе 12) не создает виртуальную таблицу, а просто дублирует отобранные данные. Например, можно извлечь все строки из таблицы `CUSTOMER` и вставить их в таблицу `PROSPECT`. Конечно, эта операция будет успешной только в том случае, если обе таблицы имеют одинаковую структуру. Но для отбора только тех покупателей, которые живут в штате Мэн, достаточно простой инструкции `SELECT`, имеющей в предложении `WHERE` всего лишь одно условие. Соответствующий код показан ниже.

```
INSERT INTO PROSPECT
SELECT * FROM CUSTOMER
WHERE State = 'ME' ;
```



Даже если эта операция и создает избыточные данные — данные о покупателях теперь хранятся в обеих таблицах, PROSPECT и CUSTOMER, — зато увеличивается производительность извлечения данных. Чтобы избежать избыточности и поддерживать согласованность данных, не вставляйте, не изменяйте и не удаляйте строки в одной таблице без вставки, изменения и удаления соответствующих строк в другой. Еще одна проблема состоит в возможности генерирования инструкцией INSERT дубликатов первичных ключей. Даже если существует единственный потенциальный клиент, чей первичный ключ (ProspectID) совпадает с первичным ключом (CustomerID) зарегистрированного покупателя, данные о котором вы пытаетесь вставить в таблицу PROSPECT, эта операция вставки не будет выполнена. Если обе таблицы имеют автоинкрементные первичные ключи, вы не должны устанавливать для счетчиков автоинкрементации в обеих таблицах одни и те же начальные значения. Позаботьтесь о том, чтобы обе группы порядковых номеров не пересекались.

Обновление существующих данных

Все течет, все меняется. Если вам не нравится нынешнее положение дел, то надо просто немного подождать — через некоторое время все может измениться к лучшему. По мере изменений, происходящих в жизни, обычно приходится обновлять и базы данных, поскольку они, как правило, отражают наш изменчивый мир в тех или иных аспектах. Например, ваш клиент может поменять адрес. Количество товаров на складе меняется ежеминутно (будем надеяться, не в результате воровства, а потому, что товар хорошо расходуется). Это типичные примеры тех событий, из-за которых приходится постоянно обновлять базы данных.

В SQL для изменения данных, хранящихся в таблице, предусмотрена инструкция UPDATE. С помощью одной такой инструкции можно изменить в таблице одну либо несколько строк или даже все ее строки. В инструкции UPDATE используется следующий синтаксис.

```
UPDATE имя_таблицы
  SET столбец_1 = выражение_1, столбец_2 = выражение_2,
    ..., столбец_n = выражение_n
  [WHERE предикаты] ;
```



Предложение WHERE не является обязательным. Оно указывает, какие строки должны обновляться. Если не использовать это предложение, будут обновляться *все* строки таблицы. Предложение SET определяет новые значения изменяемых столбцов.

Рассмотрим таблицу CUSTOMER (покупатель), представленную в табл. 6.1 и содержащую столбцы Name (имя и фамилия), City (город), AreaCode (телефонный код региона) и Telephone (номер телефона).

Таблица 6.1. Таблица CUSTOMER

<i>Name</i>	<i>City</i>	<i>AreaCode</i>	<i>Telephone</i>
Денис Данилкин	Мытищи	(495)	555-1111
Геннадий Гончаров	Люберцы	(495)	555-2222
Юрий Юрченко	Зеленоград	(495)	555-3333

<i>Name</i>	<i>City</i>	<i>AreaCode</i>	<i>Telephone</i>
Карина Новиченко	Зеленоград	(495)	555-4444
Диляра Бочкарева	Зеленоград	(495)	555-5555

Время от времени списки покупателей изменяются, по мере того как эти люди переезжают, изменяются их номера телефонов и т.п. Предположим, покупатель Юрий Юрченко переехал из Зеленограда в Мытищи. Тогда запись этого покупателя, находящуюся в таблице CUSTOMER, можно обновить с помощью следующей инструкции UPDATE.

```
UPDATE CUSTOMER
  SET City = 'Мытищи', Telephone = '777-7777'
  WHERE Name = 'Юрий Юрченко' ;
```

В результате выполнения этой инструкции в записи произошли изменения, которые показаны в табл. 6.2.

Таблица 6.2. Таблица CUSTOMER после обновления одной строки инструкцией UPDATE

<i>Name</i>	<i>City</i>	<i>AreaCode</i>	<i>Telephone</i>
Денис Данилкин	Мытищи	(495)	555-1111
Геннадий Гончаров	Люберцы	(495)	555-2222
Юрий Юрченко	Мытищи	(495)	777-7777
Карина Новиченко	Зеленоград	(495)	555-4444
Диляра Бочкарева	Зеленоград	(495)	555-5555

Подобную инструкцию можно использовать для обновления сразу нескольких строк. Предположим, город Зеленоград переживает резкий рост населения и ему теперь присвоен собственный телефонный код. Все строки покупателей, проживающих в этом городе, можно изменить с помощью одной инструкции UPDATE.

```
UPDATE CUSTOMER
  SET AreaCode = '(595)'
  WHERE City = 'Зеленоград' ;
```

Теперь таблица CUSTOMER выглядит так, как показано в табл. 6.3.

Таблица 6.3. Таблица CUSTOMER после обновления нескольких строк инструкцией UPDATE

<i>Name</i>	<i>City</i>	<i>AreaCode</i>	<i>Telephone</i>
Денис Данилкин	Мытищи	(495)	555-1111
Геннадий Гончаров	Люберцы	(495)	555-2222
Юрий Юрченко	Мытищи	(495)	777-7777
Карина Новиченко	Зеленоград	(595)	555-4444
Диляра Бочкарева	Зеленоград	(595)	555-5555

Обновить в таблице все строки гораздо легче, чем только некоторые из них, ведь в таком случае не нужно использовать ограничивающее предложение `WHERE`. Предположим, что город Москва значительно увеличился в размерах и в его состав вошли все города и населенные пункты, упомянутые в базе данных. Тогда все строки можно сразу изменить с помощью одной инструкции.

```
UPDATE CUSTOMER
SET City = 'Москва' ;
```

Результат выполнения этой инструкции показан в табл. 6.4.

Таблица 6.4. Таблица `CUSTOMER` после обновления всех строк инструкцией `UPDATE`

<i>Name</i>	<i>City</i>	<i>AreaCode</i>	<i>Telephone</i>
Денис Данилкин	Москва	(495)	555-1111
Геннадий Гончаров	Москва	(495)	555-2222
Юрий Юрченко	Москва	(495)	777-7777
Карина Новиченко	Москва	(595)	555-4444
Диляра Бочкарева	Москва	(595)	555-5555

В предложении `WHERE`, используемом для ограничения строк, к которым применяется инструкция `UPDATE`, может находиться подзапрос `SELECT`, результат которого используется в качестве входных данных для другой инструкции `SELECT`.

Предположим, вы оптовый продавец и ваша база данных включает таблицу `VENDOR` с названиями всех фирм-производителей, у которых вы покупаете товары. У вас также есть таблица `PRODUCT` с названиями всех продаваемых вами товаров и ценами, которые вы на них устанавливаете. В таблице `VENDOR` имеются столбцы `VendorID` (идентификатор поставщика), `VendorName` (название поставщика), `Street` (улица), `City` (город), `State` (штат) и `Zip` (почтовый индекс). А таблица `PRODUCT` содержит столбцы `ProductID` (идентификатор товара), `ProductName` (название товара), `VendorID` (идентификатор поставщика) и `SalePrice` (цена при продаже).

Предположим, ваш поставщик, Мегасервис Корпорейшн, принял решение поднять цены на все виды товаров на 10%. И для того, чтобы поддержать планку своей прибыли, вам также придется поднять на 10% цены продажи товаров, получаемых от этого поставщика. Это можно сделать с помощью следующей инструкции `UPDATE`.

```
UPDATE PRODUCT
SET SalePrice = (SalePrice * 1.1)
WHERE VendorID IN
(SELECT VendorID FROM VENDOR
WHERE VendorName = 'Мегасервис Корпорейшн') ;
```

Подзапрос `SELECT` отбирает из столбца `VendorID` идентификатор, соответствующий Мегасервис Корпорейшн. Затем полученное значение можно использовать для поиска в таблице `PRODUCT` тех строк, которые следует обновить. В результате цены всех товаров, полученных от Мегасервис Корпорейшн, будут повышены на 10%, а цены остальных останутся прежними. О подзапросах `SELECT` мы подробно поговорим в главе 12.

Перемещение данных

Чтобы добавить данные в таблицу или представление, помимо инструкций INSERT и UPDATE, можно воспользоваться инструкцией MERGE. Инструкция MERGE позволяет объединить данные исходных таблиц или представлений с данными других таблиц или представлений. Эта же команда позволяет вставить новые строки в нужную таблицу или обновить существующие в ней. Таким образом, инструкция MERGE представляет собой весьма удобный способ копирования уже существующих данных из одного места в другое.

Возьмем в качестве примера базу данных VetLab, описанную в главе 5. Предположим, часть сотрудников, данные о которых занесены в таблицу EMPLOYEE, — это продавцы, которые уже приняли заказы. Другая часть — это сотрудники, не связанные напрямую с продажами, или продавцы, которые еще не имеют заказов. Только что закончившийся год был прибыльным, поэтому мы решили выдать премии по 100 долларов каждому, кто принял по крайней мере один заказ, и по 50 долларов всем остальным. Для начала создадим таблицу BONUS (бонус) и вставим в нее записи всех сотрудников, которые фигурируют хотя бы в одной строке таблицы ORDERS, устанавливая в каждой записи значение премии равным 100 долларам.

Затем воспользуемся инструкцией MERGE для вставки новых записей о тех сотрудниках, которые не имеют заказов, чтобы выдать им премии по 50 долларов. Ниже приведен программный код, который позволяет создать и заполнить таблицу BONUS.

```
CREATE TABLE BONUS (  
    EmployeeName CHARACTER (30) PRIMARY KEY  
    Bonus NUMERIC DEFAULT 100 ) ;  
  
INSERT INTO BONUS (EmployeeName)  
    (SELECT EmployeeName FROM EMPLOYEE, ORDERS  
    WHERE EMPLOYEE.EmployeeName = ORDERS.Salesperson  
    GROUP BY EMPLOYEE.EmployeeName) ;
```

Теперь выполним запрос к таблице BONUS и посмотрим, что она содержит.

```
SELECT * FROM BONUS ;
```

EmployeeName	Bonus
Алена Булина	100
Диана Жень	100
Николай Кукин	100
Олег Донченко	100

Затем выполним инструкцию MERGE, чтобы назначить премии в 50 долларов всем остальным сотрудникам.

```
MERGE INTO BONUS  
    USING EMPLOYEE  
    ON (BONUS.EmployeeName = EMPLOYEE.EmployeeName)  
    WHEN NOT MATCHED THEN INSERT  
        (BONUS.EmployeeName, BONUS.bonus)  
        VALUES (EMPLOYEE.EmployeeName, 50) ;
```

Записи о сотрудниках из таблицы EMPLOYEE, которые не совпадают с записями в таблице BONUS, будут вставлены в последнюю таблицу. Теперь запрос к таблице BONUS даст следующий результат.

```
SELECT * FROM BONUS ;
```

EmployeeName	Bonus
Алена Булина	100
Диана Жень	100
Николай Кукин	100
Олег Донченко	100
Наталия Гончарова	50
Михаил Бардин	50
Олег Яненко	50
Леонид Лоевский	50

Первые четыре записи, созданные с помощью инструкции `INSERT`, располагаются в алфавитном порядке по именам сотрудников. Остальные записи, добавленные с помощью инструкции `MERGE`, располагаются в том порядке, в котором они находились в таблице `EMPLOYEE`.

Инструкция `MERGE` — относительно новое добавление в SQL и может пока не поддерживаться некоторыми СУБД. В стандарт SQL:2011 внесена еще одна возможность инструкции `MERGE`, которая может показаться где-то парадоксальной, поскольку она позволяет удалять записи.

Предположим, после выполнения инструкции `INSERT` вы решили совсем не премировать тех, кто принял хотя бы один заказ, но при этом дать премии в 50 долларов всем остальным сотрудникам. Для реализации этого решения можно выполнить следующую инструкцию `MERGE`, которая позволит удалить бонусные записи о продавцах и добавить бонусные записи о тех, кто не связан с продажами.

```
MERGE INTO BONUS
  USING EMPLOYEE
  ON (BONUS.EmployeeName = EMPLOYEE.EmployeeName)
  WHEN MATCHED THEN DELETE
  WHEN NOT MATCHED THEN INSERT
    (BONUS.EmployeeName, BONUS.bonus)
  VALUES (EMPLOYEE.EmployeeName, 50);
```

Вот как выглядит результат выполнения этого кода.

```
SELECT * FROM BONUS;
```

EmployeeName	Bonus
Наталия Гончарова	50
Михаил Бардин	50
Олег Яненко	50
Леонид Лоевский	50

Удаление устаревших данных

Со временем данные могут устаревать и становиться бесполезными. Ненужные данные, находясь в таблице, только замедляют работу системы, занимают память и путают пользователей. Наилучшим решением является перенос старых данных в архивную таблицу, а затем извлечение ее из базы данных. А если вдруг вам потребуется снова взглянуть на эти данные,

то их можно будет восстановить. При этом они уже не будут замедлять ежедневную обработку данных. Впрочем, независимо от того, будете вы архивировать устаревшие данные или нет, все же наступит время, когда их придется удалить. Для удаления строк из таблицы, находящейся в базе данных, в SQL предусмотрена инструкция DELETE.

С помощью всего одной инструкции DELETE можно удалить как все строки таблицы, так и некоторые из них. Строки, предназначенные для удаления, можно отобрать, используя в инструкции DELETE необязательное предложение WHERE. Синтаксис инструкции DELETE почти такой же, как у инструкции SELECT, за исключением того, что не нужно указывать столбцы. Ведь при удалении строки удаляются данные, находящиеся во всех ее столбцах.

Предположим, например, что ваш клиент Михаил Кадилов переехал в Швейцарию и больше ничего у вас покупать не собирается. Вы можете удалить его данные из таблицы CUSTOMER, используя следующую инструкцию.

```
DELETE FROM CUSTOMER
WHERE FirstName = 'Михаил' AND LastName = 'Кадилов' ;
```

Если у вас только один покупатель, которого зовут Михаил Кадилов, эта инструкция будет выполнена безупречно. Однако существует вероятность, что Михаилом Кадиловым зовут как минимум двух ваших покупателей. Чтобы удалить данные именно того из них, к кому вы потеряли интерес, добавьте в предложение WHERE дополнительные условия (указав, например, такие столбцы, как Street, Phone или CustomerID). Если не добавить предложение WHERE, будут удалены все записи, относящиеся ко всем, кого зовут Михаилом Кадиловым.