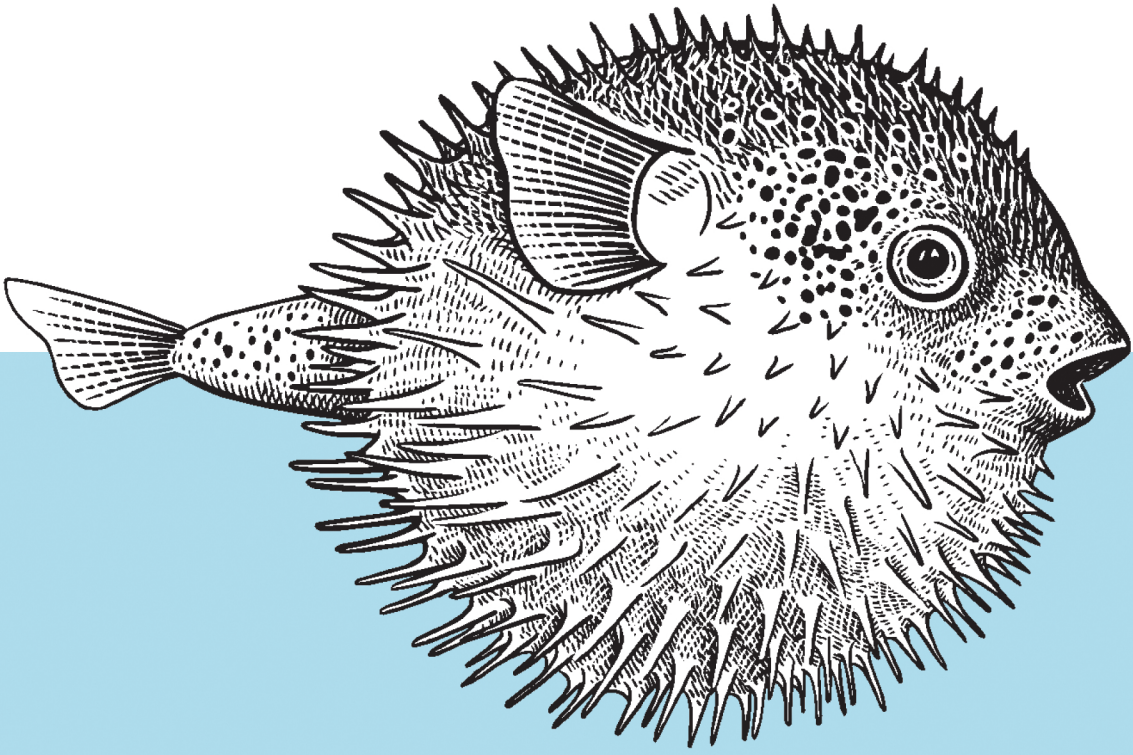
БИБЛИОТЕКА ПРОГРАММИСТА

Уолтер Шилдс

SQL



БЫСТРОЕ ПОГРУЖЕНИЕ

SQL

QuickStart Guide™

The Simplified Beginner's Guide to Managing, Analyzing, and Manipulating Data with SQL

Walter Shields

ClydeBank TECHNOLOGY

SQL

БЫСТРОЕ ПОГРУЖЕНИЕ

Уолтер Шилдс

Санкт-Петербург • Москва • Минск
2022

ББК 32.973.233-018.2
УДК 004.65
Ш57

Шилдс Уолтер

Ш57 SQL: быстрое погружение. — СПб.: Питер, 2022. — 224 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1835-9

Что общего между самыми востребованными профессиями и стремительным увеличением количества информации в мире? Ответ: язык структурированных запросов (SQL). SQL — рабочая лошадка среди языков программирования, основа основ для современного анализа и управления данными.

Книга «SQL: быстрое погружение» идеальна для всех, кто ищет новые перспективы карьерного роста; для разработчиков, которые хотят расширить свои навыки и знания в программировании; для любого человека, даже без опыта, кто хочет воспользоваться возможностями будущего, в котором будут править данные.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.233-018.2
УДК 004.65

ISBN 978-1945051753 англ.

ISBN 978-5-4461-1835-9

Хочу выразить особую благодарность моей семье:
Жюльену, Максу, Эльке и Норме.
Я не смог бы написать ни строчки
без их терпения и поддержки.

Краткое содержание

Оглавление

ЧАСТЬ II
ОПЕРАТОРЫ SQL

ЧАСТЬ III
РАСШИРЕННЫЕ ВОЗМОЖНОСТИ ЯЗЫКА SQL

Введение

С каждым днем — а точнее с каждой секундой — объем данных увеличивается. За время, которое вам понадобится, чтобы дочитать это предложение, будет отправлено более 500 000 поисковых запросов Google. За одну минуту на YouTube загружается более 300 часов видеоконтента [1]. Наши возможности хранения данных продолжают разрастаться [2]. И причина тому не только совершенствование смартфонов и рост социальных сетей. Мы сами — обычные люди — генерируем данные, постоянно создавая новые записи, отражающие наши интересы, действия, мысли и чувства. Предприятия и правительственные учреждения признают тот факт, что максимальная эффективность и прибыль не могут быть достигнуты без использования больших объемов данных.

ФЕНОМЕН BIG DATA («БОЛЬШИХ ДАННЫХ»)

GOOGLE

- Каждую секунду обрабатывается 40 000 поисковых запросов.
- В 2001 году обработано 27,5 миллиарда поисковых запросов. Для сравнения: в 2012 году их уже было 1,2 триллиона, и это число продолжает расти.

YOUTUBE

- В 2007 году каждую минуту загружалось 6 часов видеоконтента. А в 2015 году — 400 часов в минуту. Данный показатель продолжает расти.

FACEBOOK

- Каждую минуту бренды и организации получают 34 722 лайка.
- Каждый месяц передается 30 миллиардов единиц контента.

AMAZON

- Хранит данные от 152 миллионов клиентов на 1,4 миллиона серверов, расположенных в нескольких центрах обработки данных.

Рис. 1 [3], [4]

Хотя объем генерируемых данных удивляет сам по себе, еще более поразительно, что мы только начинаем их использовать. Анализу подвергается не более полупроцента от всех собираемых данных. Если бы отдельные люди, компании, правительства и другие организации эффективнее использовали собранные данные, то потенциал роста стал бы безграничным [5]. Прибыль и эффективность увеличились бы. Маркетологи смогли бы лучше понимать свою целевую аудиторию. Мошенничество и преступления могли бы быть предотвращены гораздо успешнее. И это только начало! Желание более полно анализировать данные приводит к росту спроса на квалифицированных специалистов по работе с базами данных. И это — *вы*.

Итак, вы читаете эту книгу, чтобы изучить SQL (Structured Query Language — язык структурированных запросов). Вы уже осознали важную роль данных в мире и хотите максимально эффективно их использовать — мудрое решение! Кто-то из вас изучал SQL в школе или колледже. Возможно, вы работаете в сфере бизнеса или государственного управления и хотите развить навыки, необходимые для продвижения вашей карьеры. Как бы то ни было, практический подход к SQL, изложенный в этой книге, окажется для вас бесценным ресурсом.

SQL (правильно произносится «эс-кью-эль», хотя зачастую говорят «сиквел») — это язык программирования, который используется для работы с базами данных. SQL — главный инструмент оптимизации и обслуживания баз данных, который дает возможность обычным пользователям, даже без опыта программирования, работать с базой данных и превращать большие данные в практически значимую информацию. Уже в течение восемнадцати лет я работаю с SQL и другими системами управления базами данных, но до сих пор помню, каково это — быть новичком. Мне потребовалось время, чтобы разобраться в управлении данными. Когда я впервые познакомился с Microsoft Access, управление данными я рассматривал просто как еще один пункт в области информационных технологий. Тогда еще количество полученных данных было относительно небольшим. Однако в настоящее время каждый сделанный в Google поисковый запрос, каждый опубликованный в Facebook пост и каждый обзор в любой категории товаров на Amazon в конечном итоге хранится где-то на сервере и может быть запрошен. SQL — это основной аналитический инструмент, используемый для расшифровки больших данных, поэтому и возникла огромная потребность в специалистах, знающих язык программирования SQL. Язык данных — это язык, который в той или иной степени считается основой каждой современной технологически ориентированной компании. Изучив материал этой книги, вы научитесь свободно оперировать языком данных.

Почему я написал эту книгу

Я заинтересовался языком программирования SQL в конце девяностых годов. В то время возможности изучения SQL были весьма ограниченными, поскольку феномен больших данных еще не стал таким очевидным. Я прошел суровую школу жизни. Изучал SQL методом проб и ошибок. Даже после получения степени бакалавра компьютерных наук и магистра в области информационных систем управления я не считал, что теоретический подход к обучению — лучший путь к успеху в науке о данных. Я могу подтвердить, что, проработав более восемнадцати лет на различных должностях в разных компаниях — от стартапов до компаний из списка Fortune 500 — и в разных сферах — от юриспруденции до здравоохранения и розничной торговли, — для управления системами данных я использовал язык SQL (и другие языки программирования).

В процессе учебы я понял следующее: нужно больше практики, чтобы стать отличным специалистом в области баз данных. Для вас это не будет так сложно, как для меня. Фактически я хотел, упорядочив все полученные знания, разработать простой и понятный подход к SQL. В книгах по SQL обычно много времени отведено на объяснение истории и теории информатики, лежащих в основе баз данных и языка запросов. В этой книге подробно раскрываются только вопросы, затрагивающие работу реляционных баз данных, а все остальное дано обзорно. Создание подобного ресурса, который может использовать даже новичок в SQL, — достаточно амбициозная задача. Но я преподаю SQL уже много лет: я взаимодействовал со многими студентами, изучающими SQL, и этого достаточно, чтобы понять, что работает, а что нет.

Умение работать с SQL обязательно привлечет к вам людей. Если вы освоите навык работы с SQL, ваше мнение будут ценить и станут обращаться к вам за советом. Это может быть руководитель, которому нужны целевые данные о последней маркетинговой кампании, или коллега-программист, который обратится к вам за помощью в составлении запроса. Бизнес по обучению языку программирования SQL и бизнес по визуализации данных — это главное дело моей жизни. История его развития началась в кофейне в Трибеке, штат Нью-Йорк, куда я пришел со своим ноутбуком и подготовленными учебными материалами по SQL. Я не знал, появится ли кто-нибудь. Я, возможно, недооценил потребность людей в изучении SQL, так как пришло очень много людей. И их поток не кончался. Наконец я понял, что у меня есть свое дело.

С годами я обнаружил, что мне действительно нравится преподавать. Я вырос среди людей, ценящих образование, и теперь убедился, что передача знаний развивает уверенность и способствует позитивному личностному росту. До сих пор

радуюсь, когда вижу, как у студентов светятся глаза, исчезают тревога и страх! Я никогда не устаю наслаждаться такими моментами. Я улучшаю и совершенствую программу обучения, работая со множеством студентов разного уровня подготовки и набором навыков. Думаю, было бы классно провести несколько курсов в Тринидаде и Гренаде, откуда я родом. Здорово поделиться знаниями и опытом, приобретенными в Соединенных Штатах, а также разработать франшизу на Карибах! Можно одновременно мечтать и двигаться навстречу своей цели!

Поддержка для новичка

Чтобы добиться успеха в изучении SQL, вам необходимо запастись терпением. Я верю в каждого своего ученика. Мой успех как преподавателя — это достижения моих учеников. Новичкам в SQL наука управления базами данных покажется сложной. Примите это, учитесь прилежно, и вы обязательно добьетесь успеха! Вот несколько важных фактов, о которых следует помнить, если вы еще недостаточно знакомы с SQL.

- Не бойтесь ошибаться. Существует множество способов изучать SQL без возможности «испортить» существующую базу данных, что беспокоит некоторых новичков. Вместе с этой книгой вам будет доступна учебная база данных. Мне бы хотелось, чтобы вы в полной мере использовали ее и тщательно изучили, не боясь пробовать и ошибаться.
- Относитесь к этой книге как к учебному пособию. Выделяйте, подчеркивайте, пишите на полях. SQL необходимо изучать в теории и на практике. Специально для этой книги я разработал упражнения на основе реальных практических задач. Упражнения дополняют друг друга, что позволит вам закрепить знания, изученные в предыдущей главе. Если вы считаете, что новая концепция сложна, имеет смысл вернуться назад и снова проработать упражнения.
- Приятного вам обучения! Никогда не упускайте из виду тот факт, что информационные технологии — самая динамично развивающаяся сфера. Изучая языки программирования, вы развиваете востребованный навык, который позволит радикально изменить мир. Если после всего сказанного вы немного взволнованны — это нормально!

Охват и цель книги

Для читателей, которые уже владеют базовыми или более профессиональными навыками работы с SQL, это краткое руководство станет весьма удобным

справочником, к которому всегда можно обратиться при составлении запросов. А для новичков это краткое руководство будет отличным учебным пособием.

Обратите внимание, что в первую очередь я рассказываю о базовых инструментах SQL, необходимых для понимания и получения полезной информации из баз данных. Стандартные методы запросов, описанные в этой книге, можно выполнять, не боясь при этом изменить базу данных. В одной из глав мы кратко рассмотрим добавление, изменение и удаление записей из базы данных (DML — язык управления данными). Однако для извлечения информации из базы данных изучать DML не обязательно, но полезно знать, как выполняется процесс. Эта глава может оказаться полезной тем, кто рассматривает возможность карьерного роста в сфере администрирования баз данных.

SQL и ваша карьера

SQL — один из наиболее востребованных языков программирования. Он применяется для работы в области администрирования баз данных и во множестве других технических областей, включая разработку и тестирование программного обеспечения, бизнес-анализ и прочее. Рассмотрим профессии, где требуется знание SQL.

- **Администратор баз данных (DBA).** Отвечает за разработку требований к базе данных, ее проектирование, реализацию, использование и сопровождение. Администраторы баз данных играют ключевую роль при приобретении или модификации аппаратных и программных средств, входящих в инфраструктуру баз данных компании. Администраторы баз данных также несут ответственность за контроль доступа к БД. Они устанавливают и ограничивают доступ, управляют паролями и т. д.
- **Разработчик баз данных.** Это программист, который специализируется на создании, отладке, оптимизации и обслуживании баз данных. Во многих организациях SQL-программистов просят разработать блоки кода для использования неквалифицированными специалистами. На SQL-программистов также часто возлагают ответственность за текущее тестирование базы данных, чтобы гарантировать хорошую производительность и оптимизированное функционирование.
- **Data Scientist.** Он должен уметь извлекать необходимую информацию из самых разнообразных источников, видеть логические связи в системе собранной информации и на основе количественного анализа разрабатывать эффективные бизнес-решения. В Amazon, например, такой специалист может разрабатывать систему контекстной рекламы.

По мере того как наши навыки в области хранения огромных объемов данных продолжают расти, развивается и сама отрасль. Университеты предлагают степени в области администрирования баз данных, обработки и управления данными. В этой быстроразвивающейся области SQL — основной язык, и изучение этого языка программирования — ваш ключ к успеху в области больших данных.

Пока требуются специалисты по работе с базами данных, спрос на изучение SQL будет расти все больше и больше. В любой отрасли (не только в сфере больших данных) вы можете найти множество вакансий, где требуются знания и навыки работы с SQL. Не всегда основное требование — знание SQL. Но оно всегда считается большим преимуществом.

Если вы будете постоянно развивать и демонстрировать свои навыки в SQL, вы всегда сможете рассчитывать на достойную зарплату. В США в 2018 году средняя зарплата специалиста по SQL превышала 80 000 долларов. Прежде чем компания наймет вас на полный рабочий день, вам необходимо освоить теорию и немного практики [6]. Многие компании предлагают оплачиваемые стажировки, благодаря которым вы сможете проверить свои навыки работы с SQL на реальном проекте.

Некоторые из вас могут воспользоваться знаниями SQL для карьерного роста в компании, в которой вы в настоящее время работаете. Возможно, ваши знания и навыки в SQL необходимы для вашей компании, и компания выделит вам время и ресурсы для развития. И если вы в конечном итоге станете еще более востребованным специалистом — это всегда прекрасно, не так ли?

Язык структурированных запросов SQL используется в сфере технологий практически во всех областях. Если компании извлекают выгоду из хранения и анализа данных, то им выгодно использовать SQL. Найдите время и подумайте, кому требуется анализ данных. Это несложно. На самом деле гораздо сложнее найти компании, которым невыгодно нанимать специалистов по работе с базами данных. Спрос на таких специалистов существует уже давно, однако в настоящее время потребности бизнеса в работе с данными постоянно растут, а следовательно, растет и спрос на специалистов, работающих в сфере баз данных.

Как организована книга

При желании всегда можно освоить новый навык. Я обнаружил, что для данного конкретного навыка лучший способ — одновременно изучать теорию и практиковаться. Эта книга позволит вам как можно быстрее научиться писать запросы. Книга состоит из трех основных частей, каждая включает три-четыре главы.

- Часть 1 «Создание среды обучения SQL» — краткое знакомство с терминологией и структурой баз данных, а также возможность попрактиковаться в настройке конкретного программного обеспечения для работы с базами данных, которое мы рассматриваем в этой книге. Даже если вы убеждены, что знаете и понимаете основы SQL, мы все равно настоятельно рекомендуем вам последовательно изучать материал книги, поскольку в этом разделе описаны конкретные инструменты, методы и стратегии SQL.
  - Глава 1 «Структура базы данных». В этой главе вы изучите структуру реляционной базы данных, а также типы данных. Также здесь дан краткий обзор некоторых терминов. В конце главы вы сможете выполнить практические задания.
  - Глава 2 «Инструменты и стратегии SQL». Здесь вы научитесь использовать бесплатно загружаемое программное обеспечение SQL (SQLite), а также работать с учебной базой данных SQL, чтобы закрепить и проверить усвоенные знания. В конце главы даны практические задания и вопросы для самопроверки.
  - Глава 3 «Работа с базой данных в SQLite». Вы научитесь работать с учебной базой данных в браузере SQL, использовать браузер для навигации по общей структуре базы данных, для просмотра данных в отдельных таблицах, познакомитесь с вкладкой Execute SQL (Выполнить SQL-запрос).
- Часть 2 «Операторы SQL». Здесь описаны операторы и команды, предназначенные для написания простых запросов.
  - Глава 4 «Работа с запросами». Вы познакомитесь с основным оператором `SELECT`, узнаете, как получить доступ к сохраненной в таблицах информации с помощью оператора `FROM`. Также научитесь сортировать эти данные в алфавитном порядке с помощью оператора `ORDER BY`, а затем использовать оператор `LIMIT` для ограничения числа записей.
  - Глава 5 «Преобразование данных в информацию». Вы узнаете, как использовать оператор `WHERE`, а также операторы сравнения, логические и арифметические операторы. Также вы познакомитесь с оператором `LIKE`, научитесь использовать специальные символы и работать с функцией `DATE()`, операторами `AND`, `OR` и оператором `CASE`.
  - Глава 6 «Работа с несколькими таблицами». Здесь рассказано об операторах для соединения двух или нескольких таблиц. Операторы `INNER JOIN`, `LEFT JOIN` и `RIGHT JOIN` позволяют возвращать и сравнивать данные из нескольких таблиц.
  - Глава 7 «Функции языка SQL» познакомит вас с функциями языка SQL, включая функции агрегирования, строки и функции даты и времени.

- Часть 3 «Расширенные возможности языка SQL» знакомит с более сложными, но очень полезными методами, используемыми для повышения эффективности запросов. В этой части вы познакомитесь с основами языка управления (манипулирования) данными (DML), который, в отличие от ранее изученных операторов SQL, изменяет информацию в базе данных.
  - Глава 8 «Подзапросы» посвящена концепции вложенных запросов или подзапросов. В этой главе показано, как использовать подзапросы с различными операторами SQL, которые вы уже знаете. Кроме того, вы изучите новый оператор `DISTINCT`.
  - Глава 9 «Представления». Здесь рассказывается о виртуальных таблицах, известных как представления. Это запросы, которые сохраняются и могут по мере необходимости выполняться повторно или использоваться в качестве подзапросов в других операторах SQL.
  - Глава 10 «DML — язык управления данными». В этой главе вы познакомитесь с основами языка управления данными — DML. Также вы изучите новые операторы `INSERT`, `UPDATE` и `DELETE`.

ЧАСТЬ I

СОЗДАНИЕ СРЕДЫ ОБУЧЕНИЯ SQL

ГЛАВА 1

Структура базы данных

Краткое содержание

- ✓ Понимание использования языков баз данных
- ✓ Как работает реляционная база данных
- ✓ Типы данных
- ✓ Системы управления реляционными базами данных (СУБД)
- ✓ SQLite

При обучении новой технической дисциплине начинать стоит с базового словаря. Мы постарались выдержать баланс: изложить основные термины и концепции, избегая жаргона или сложных правил. В этой главе мы познакомим вас с концепцией реляционной базы данных и продемонстрируем типы данных, с которыми вы будете иметь дело в обычной базе данных. Мы также познакомим вас с основным оператором SQL — SELECT.

Основная терминология

Данные — это часть информации [7]. Данные находятся повсюду и содержатся везде, но на практике термин «данные» обычно относится к информации уже записанной или той, которую можно записать. *Таблица* — один из самых простых инструментов, используемых для записи и визуализации данных. Таблица — это двухмерная сетка, состоящая из строк и столбцов.

ПРИМЕЧАНИЕ

При работе с базами данных таблица также может называться «базовой относительной переменной», хотя в этой книге мы будем придерживаться термина «таблица». Сводная терминология представлена на рис. 5.

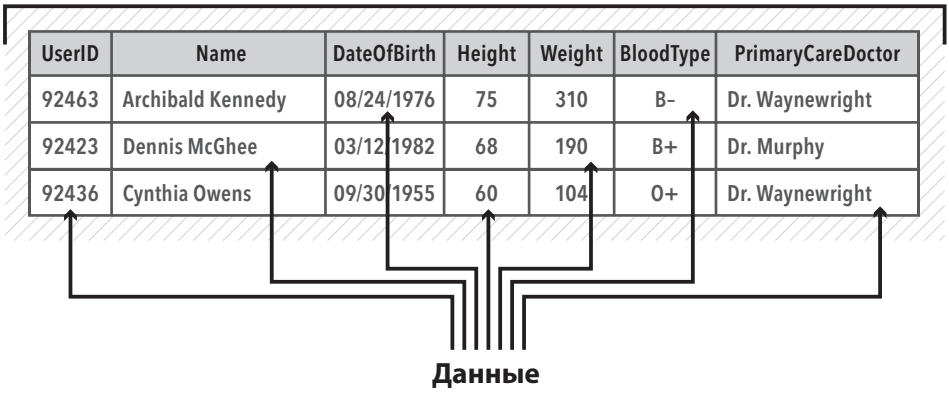


Рис. 2

На рис. 2 представлена таблица, содержащая различные типы данных. К данным могут относиться имена, числа, даты, символы (например, «+» или «-») или любые другие форматы. Данные — это просто информация. Следовательно, при обработке данных нам следует соответствующим образом ограничить их. На рис. 2 показана таблица, в которой хранится основная информация о пациентах. Данные о пациентах заданы в различных форматах — числа, имена и даты, а в поле `BloodType` (группа крови) представлена строка из двух символов (буква и символ «+» или «-»). Форматы, используемые для визуализации данных, не случайны. Все базы данных содержат *метаданные*, то есть данные, описывающие структуру и форматирование самих данных, часто называемые «данными о данных». Например, поле `DateOfBirth` (дата рождения) может содержать метаданные, которые преобразовывают информацию к формату мм/дд/гггг. Метаданные, содержащиеся в поле `Height` (рост), могут ограничивать длину данных двумя знаками, если рост измеряется в дюймах.

Термин *«база данных»* можно определить как совокупность данных, упорядоченных для упрощения и скорости поиска и извлечения с помощью компьютера. База данных, как правило, изображается графически в виде цилиндров, расположенных один над другим (рис. 3), что символизирует несколько жестких дисков, используемых при создании центра хранения данных большой емкости.

Как правило, информация внутри базы данных хранится в виде набора таблиц. Каждая таблица содержит определенные наборы данных, которые могут быть взаимосвязаны с другими данными из других таблиц и ссылаться на них.

ПРИМЕР

Таблица данных пациентов (рис. 2) – это просто таблица, а не база данных. Однако эту таблицу можно было бы включить в базу данных вместе с другими таблицами, такими как информация о лабораторных тестах, выписанных рецептах, историях посещений, персонале больницы, о врачах, их специализации и времени приема.

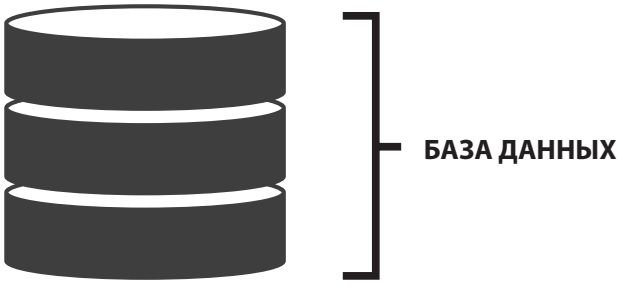
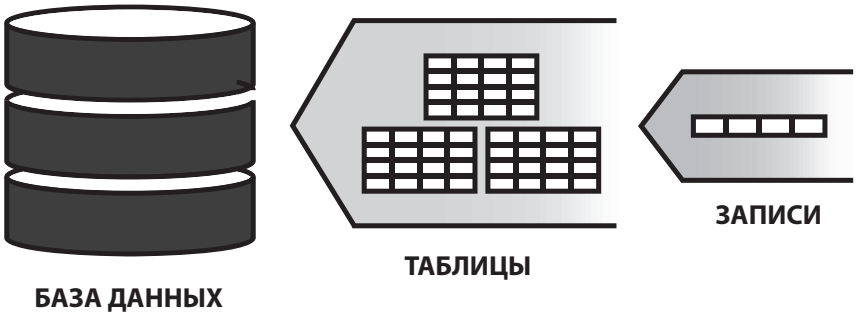


Рис. 3

Роль базы данных — упростить взаимодействие, организацию и анализ связанных данных из различных источников. Данные сохраняются во взаимосвязанных таблицах, что дает возможность манипулировать ими более гибко.



База данных состоит из таблиц. Таблицы состоят из записей.

Рис. 4

Строки в таблице называются *записями*. Также их можно называть *кортежами*. Столбцы в таблице, как правило, называются *полями*. Также их можно назвать *атрибутами*. Поля/атрибуты — это категории, используемые для определения данных в записи (строке).

ПРИМЕЧАНИЕ

В этой книге для описания строк в таблице мы будем использовать термин «записи», а для описания столбцов – «поля». См. рис. 5 – основная терминология.

ОСНОВНАЯ ТЕРМИНОЛОГИЯ

Терминология, используемая в этой книге	Другое название
Запись, строка	Кортеж
Поле, столбец	Атрибут
Таблица	Связи, базовая относительная переменная

Рис. 5

Каждая запись разбита на несколько полей, представляющих собой отдельные элементы данных, которые описывают конкретный элемент. Например, в таблице на рис. 6 хранятся сведения о пациентах определенной больницы, медицинского учреждения или страхового фонда. В данном случае база данных, скорее всего, будет состоять из нескольких таблиц. Понимание того, как таблицы связаны друг с другом, — это и есть ключ к пониманию основной архитектуры базы данных.

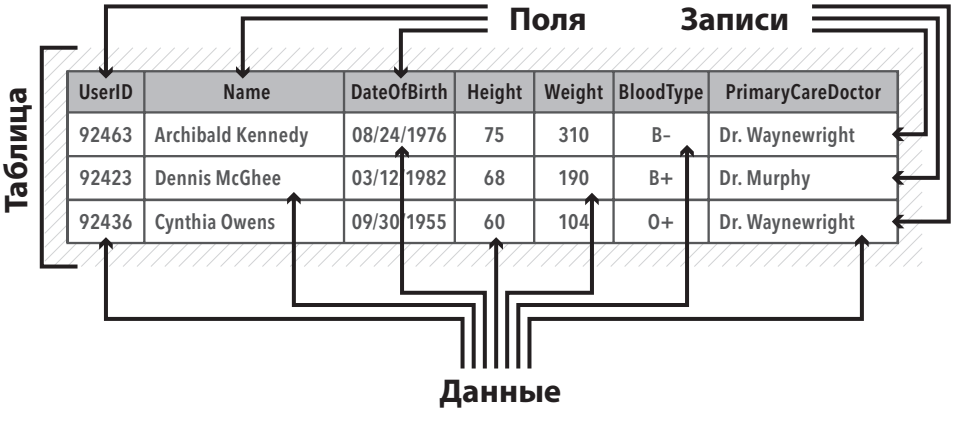


Рис. 6

Основные элементы реляционных баз данных

Реляционная база данных — это конструкция базы данных, которая в 1969 году была официально утверждена ученым из IBM Эдгаром Франком Коддом. В следующем году Кодд опубликовал статью под названием «Реляционная модель данных для больших, совместно используемых банков данных» [8]. Девять лет спустя несколько крупных игроков в сфере технологий, в том числе IBM и Relational Software Inc. (позже ставшая Oracle), начали использовать реляционные базы данных в коммерческих целях. Четыре десятилетия спустя реляционная модель становится наиболее распространенной формой проектирования баз данных.

Чтобы получить элементарное представление о работе реляционных баз данных, важно понимать роль ключевых полей.

Реляционная база данных будет содержать множество таблиц, аналогичных таблице `patient_info` (рис. 7). Таблицы связаны друг с другом с помощью ключевых полей. Как вы видите на рисунке, таблица `patient_info` содержит поля, являющиеся первичным ключом и внешним ключом. Каждая таблица в реляционной базе данных должна содержать первичный ключ. Первичный ключ — это уникальный идентификатор записи в таблице. *Первичный ключ* каждой записи должен быть уникальным и не должен быть нулевым (пустым). Обратите внимание на поле `PatientID` в таблице `patient_info`. Поскольку это поле используется как первичный ключ, у каждой записи в таблице значение данного поля должно быть уникально. Иными словами, никакие две записи не могут иметь один и тот же PatientID.

Хотя первичный ключ (в данном случае `PatientID`) должен быть уникальным, другие поля могут содержать данные, повторяющиеся более чем в одной записи. Рассмотрим поле `PrimaryCareDoctorID`. Например, если Dr. Waynewright, ID 106547 (см. первую строку рис. 7), лечит нескольких пациентов из базы данных, то его имя и идентификатор могут встречаться в нескольких записях таблицы.

ПРИМЕЧАНИЕ

В реляционной базе данных таблицы часто называют «связями», так как они содержат набор записей (строк), связанных с различными полями (столбцами). Однако в этой книге мы будем использовать термин «таблица». См. рис. 5 – «Основная терминология».

Внешний ключ — это поле в таблице, значение которого соответствует первичному ключу в другой таблице. Предположим, что в дополнение к нашей

таблице `patient_info` в базе данных существует еще одна таблица с именем `primary_care_doctors`, в которой в качестве первичного ключа используется поле `PrimaryCareDoctorID`. В таблице `primary_care_doctors` строка Dr. Waynewright с ID 106547 появится только в одной записи. Именно совпадение различных ключевых полей в разных таблицах обеспечивает исключительно важную взаимосвязь в реляционной базе данных. Как правило, эти связи отображаются в виде *схемы* базы данных, также известной как *диаграмма «сущность — связь»* (*ERD, Entity Relationship Diagram*), которая служит своего рода эскизом для базы данных.

ПЕРВИЧНЫЙ КЛЮЧ

ВНЕШНИЙ КЛЮЧ

PatientID	PatientName	PrimaryCareDoctorID	PrimaryCareDoctorName	DateOfBirth	Height	Weight	BloodType
92463	Archibald Kennedy	106547	Dr. Waynewright	8/24/1976	75	310	B–
92425	Dennis McGhee	106474	Dr. Murphy	3/12/1982	68	190	B+
92443	Cynthis Owens	106547	Dr. Waynewright	9/30/1955	60	104	O+
92478	William Hampton	106437	Dr. Salazar	6/5/1973	73	175	AB–
92392	Hilda Bass	106783	Dr. Dean	6/10/1997	68	152	B+
92436	Frankie Stone	106437	Dr. Salazar	5/28/1979	68	106	O+
92403	Verna Sullivan	106984	Dr. Conner	7/17/2010	66	125	O+
92398	Merle Doyle	106439	Dr. Frank	1/8/1962	65	143	B–
92442	Ruth Swanson	106954	Dr. Hines	2/15/1970	61	160	O–
92384	Johnathan Singleton	106474	Dr. Murphy	6/2/1970	61	232	AB+
92405	WM Patrick	106439	Dr. Frank	6/11/1955	69	196	O+
92376	Mona Norris	106984	Dr. Conner	10/15/1932	60	98	B+
92399	Rick Gordon	106366	Dr. Hart	1/25/2002	68	149	B+
92408	Don Rivera	106437	Dr. Salazar	7/26/1954	72	185	A–
92389	Sheri Griffin	106211	Dr. Harvey	12/16/1987	78	132	AB–
92466	Guillermo Lawrence	106954	Dr. Hines	2/8/1978	60	219	O+
92310	Felipe Parker	106474	Dr. Murphy	12/10/1998	61	165	O–
92413	Brandi Carlson	106399	Dr. Flowers	11/20/2000	66	112	B+
92398	Floyd Casey	106783	Dr. Dean	12/14/1986	61	203	A–
92439	Patrick Walton	106366	Dr. Hart	8/11/1973	76	189	O+
92421	Vicki Klein	106954	Dr. Hines	11/28/1980	65	98	O+
92381	Cathy Harrison	106474	Dr. Murphy	11/16/1946	78	203	AB–
92393	Ann Guerrero	106783	Dr. Dean	6/25/1974	61	142	B–
92437	Gustavo Bates	106399	Dr. Flowers	2/25/2001	78	165	A–

Рис. 7

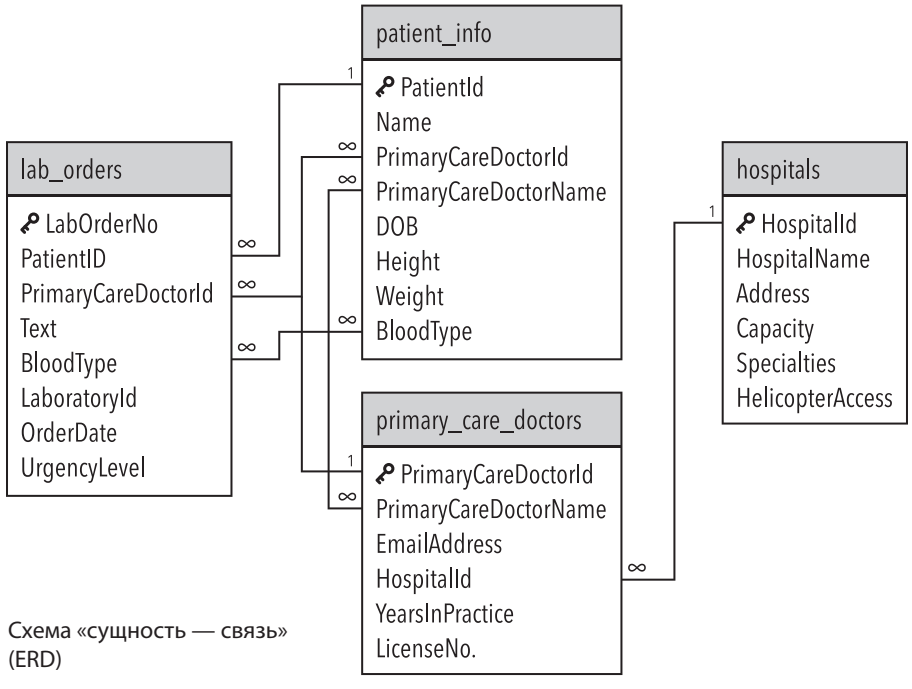


Схема «сущность — связь» (ERD)

Рис. 8

Пока не беспокойтесь, что означают символы `1` и ∞. Очень скоро мы к ним вернемся. А сейчас рассмотрим схему и ее взаимосвязи. В данной схеме всего четыре таблицы, и таблицы связаны друг с другом посредством одного или нескольких общих полей. Поле `PatientID` — это первичный ключ для таблицы `patient_info` и одновременно внешний ключ для таблицы `lab_orders`. Точно так же поле `HospitalId` — первичный ключ для таблицы `hospitals`, но это внешний ключ для таблицы `primary_care_doctors`. Очень просто, правда? Давайте рассмотрим другую схему.

Схема на рис. 9 описывает базу данных для обработки заказов и доставки товаров клиентам. И здесь пора поговорить о символах `1` и ∞, которые находятся на схеме на концах соединительных линий. Эти символы описывают связи между таблицами. Когда на одном конце соединительной линии находится символ `1`, а на другом — символ ∞, это означает связь между полями таблиц «один-ко-многим».

Рассмотрим подробнее таблицу `products` (рис. 10). В ней представлены данные о различных товарах и их атрибутах.

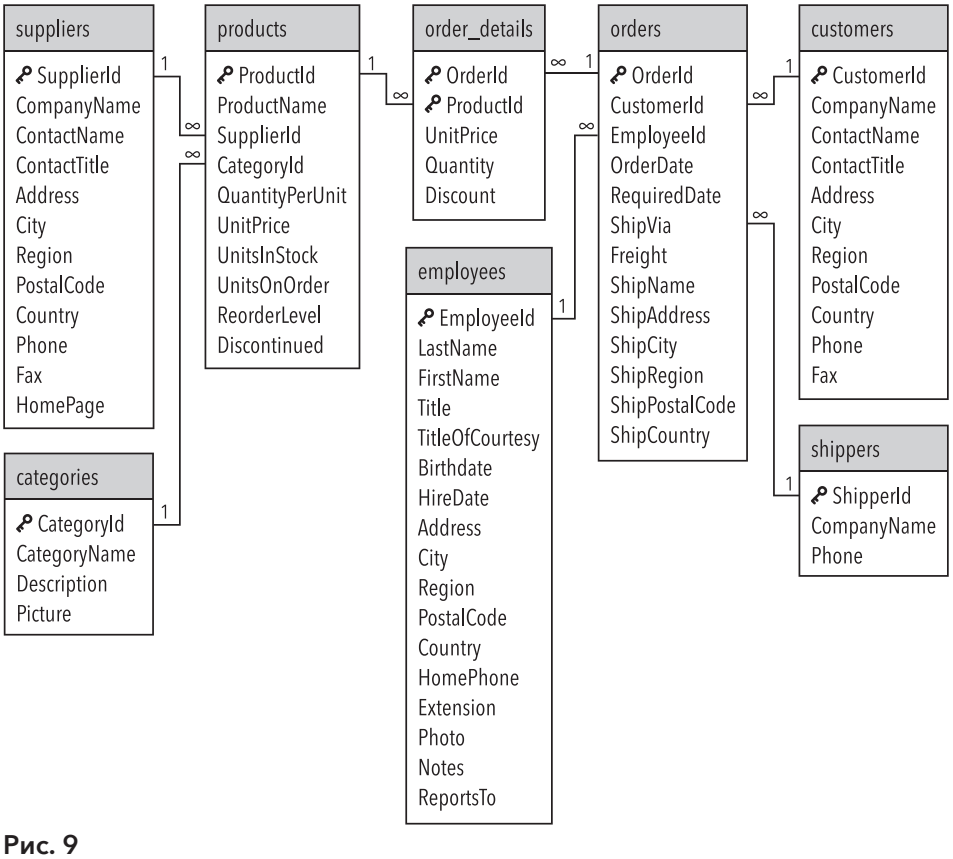

suppliers
SupplierId
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
HomePage
products
ProductId
ProductName
SupplierId
CategoryId
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued
order_details
OrderId
ProductId
UnitPrice
Quantity
Discount
orders
OrderId
CustomerId
EmployeeId
OrderDate
RequiredDate
ShipVia
Freight
ShipName
ShipAddress
ShipCity
ShipRegion
ShipPostalCode
ShipCountry
customers
CustomerId
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
categories
CategoryId
CategoryName
Description
Picture
employees
EmployeeId
LastName
FirstName
Title
TitleOfCourtesy
Birthdate
HireDate
Address
City
Region
PostalCode
Country
HomePhone
Extension
Photo
Notes
ReportsTo
shippers
ShipperId
CompanyName
Phone

Рис. 9

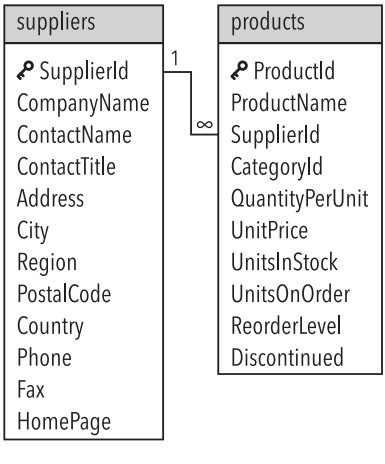

suppliers
SupplierId
CompanyName
ContactName
ContactTitle
Address
City
Region
PostalCode
Country
Phone
Fax
HomePage
products
ProductId
ProductName
SupplierId
CategoryId
QuantityPerUnit
UnitPrice
UnitsInStock
UnitsOnOrder
ReorderLevel
Discontinued

Рис. 10

Поле `ProductId` — первичный ключ для этой таблицы (обозначено значком ключа). Каждая запись в таблице будет содержать уникальный идентификационный номер товара. Фактически основная цель этой таблицы — каталогизировать атрибуты различных товаров базы данных. Теперь давайте проанализируем связи между таблицами `products` и `suppliers`.

SUPPLIERS

SupplierId	CompanyName	ContactName	ContactTitle	Address	City	etc.
S-101	Van Eck Industries	Bruce Davidson	Vp Operations	2158 Del Dew Drive	Temple Hills	...
S-102	Wright & Gate Co	Wilma Joy	Supply Chain Supervisor	291 Creekside Lane	Ventura	...
S-103	Olivias Supply	Brad Pence	Site Manager, Baton Rouge	4353 Locust View Drive	Baton Rouge	...
S-104	Cantor Corporation	Orville Bedford	President	2811 West Drive	Chicago	...
S-105	Bellagio Finland	Wallace Grim	Distributions Superviser	4939 Breezewood Court	Chanute	...
S-106	Decks Materials	John Tuck	VP Operations	4529 Counts Lane	Lextington	...
S-107	Lennor Co	Rachel Durst	Site Manager, Jackson	2216 Rhapsody Street	Gainesville	...

Рис. 11

PRODUCTS

ProductId	ProductName	SupplierId	CategoryId	QuantityPerUnit	UnitPrice	etc.
P001	Welding goggles	S102	SA-432	1	$12.99	...
P002	Welding helmet	S102	SA-432	1	$41.49	...
P003	Stick electrodes	S104	WE-214	40	$7.00	...
P004	Magnetic clamp	S101	WE-220	1	$11.86	...
P005	Heat resistant blanket	S104	WE-212	1	$3.73	...
P006	Work table	S105	GE-100	1	$1,386.67	...
P007	Replacement plates	S105	GE-100	1	$396.00	...
P008	Welding wire	S104	WE-214	1	$112.86	...
P009	Welding coveralls	S102	SA-435	1	$60.27	...
P010	Welding nozzle	S103	WE-214	1	$141.65	...
P011	Gas regulator	S106	AU-100	1	$166.25	...
P012	Welding hoods	S102	SA-432	1	$42.37	...
P013	Spot welding electrode	S104	WE-212	1	$2.35	...
P014	Plasma cutter	S107	PL-100	1	$1,645.91	...
P015	Plasma cutter cutting tip	S107	PL-100	1	$9.27	...

Рис. 12

Между таблицами `suppliers` и `products` существует связь «один-ко-многим», основанная на данных поля `SupplierId`. В таблице `suppliers` каждая запись будет иметь уникальный идентификационный номер для каждого поставщика. В таблице `products` может быть несколько записей с одним и тем же идентификационным номером поставщика.

Значок ключа, расположенный рядом с полем `SupplierId` в таблице `suppliers`, означает, что `SupplierId` — это первичный ключ для этой таблицы. В базе данных, безусловно, может содержаться большое количество разных товаров (каждый будет иметь уникальный идентификационный номер), поступающих от одного поставщика и каталогизированных в таблице `products`. Взгляните на таблицу `suppliers`, где должен быть только один уникальный идентификационный номер поставщика для каждой записи.

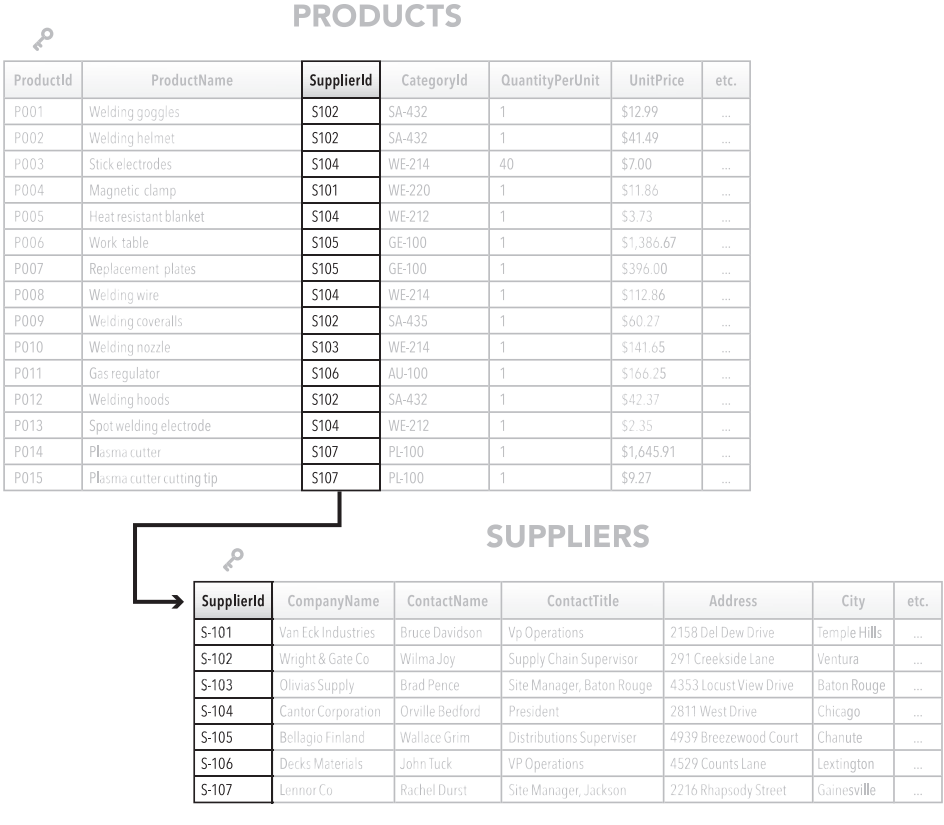


Рис. 13

ПРИМЕЧАНИЕ

Данные в поле `supplierId` могут повторяться в нескольких записях в таблице `products`, но не в таблице `suppliers`.

Давайте проанализируем взаимосвязь между таблицами `products`, `order_details` и `orders` (рис. 14).

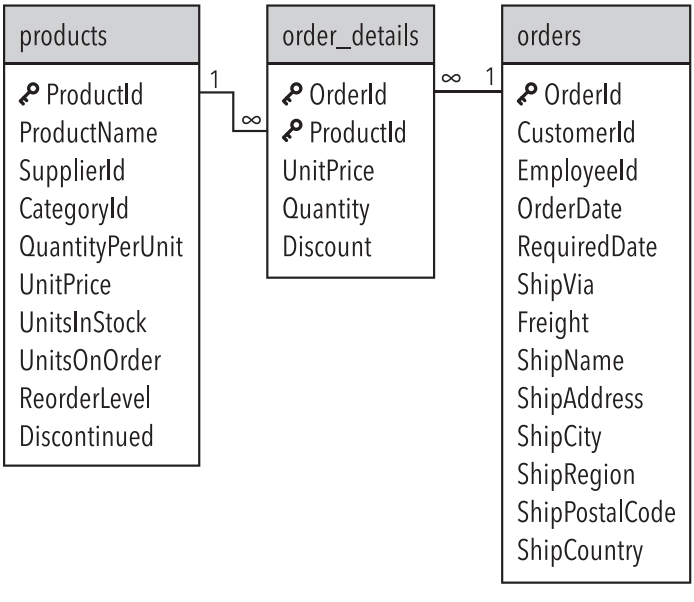


Рис. 14

Таблица `order_details` имеет два первичных ключа (обозначено значками ключей). Можно трактовать это как *составной ключ*, то есть для определения первичного ключа используются два или более поля. Хотя технически в примере задействовано два ключа, стоит рассматривать их как единый элемент — собственно первичный ключ.

Комбинация данных в полях, используемая для формирования составного ключа, работает как уникальный идентификатор для любой записи в таблице. Другими словами, если запись `OrderId` в таблице `order_details` равна 101, а `ProductId` той же записи — P006, то мы можем предположить, что никакая другая запись в таблице не будет иметь такую же комбинацию данных этих двух полей. Могут существовать одна или несколько других записей с `OrderId`, равным 101, а также одна или несколько других записей с `ProductId`, равным P006, но только одна

запись может иметь и 101 в качестве своего `OrderId`, и P006 в качестве своего `ProductId`. Эта комбинация данных работает как составной ключ, который, как и любой первичный ключ, обеспечивает уникальный идентификатор для каждой записи в таблице.

СОСТАВНОЙ КЛЮЧ

OrderId	ProductId	UnitPrice	Quantity	Discount
101	P006	$1,386.67	1	NULL
101	P003	$7.00	3	NULL
101	P005	$3.73	1	10%
102	P011	$166.23	1	NULL
102	P013	$2.35	1	NULL
103	P014	$1,645.91	1	NULL
104	P001	$12.99	3	NULL
104	P012	$42.37	3	NULL
104	P011	$166.23	2	10%
104	P003	$7.00	5	NULL
105	P010	$141.65	1	NULL
105	P004	$11.86	3	NULL
105	P003	$7.00	2	NULL
106	P014	$1,645.91	1	NULL

Рис. 15

В связи «один-ко-многим» стандартный первичный ключ находится на стороне «один». Например, в таблице `orders` мы видим, что первичный ключ, поле `OrderId`, предоставляет уникальный идентификатор для каждой записи в таблице. Сторона связи «ко-многим» находится в таблице `order_details`. Как вы думаете, почему?

Давайте рассуждать логически. Можно сделать вывод, что роль таблицы `order_details` — предоставить информацию о различных заказанных товарах. Также допустимо, что любой товар может быть заказан несколько раз несколькими заказчиками при самых разных обстоятельствах, с разными ценами и т. д. Следовательно, поле `ProductId` не может использоваться само по себе в качестве первичного ключа для таблицы `order_details`. Кроме того, любой заказ может включать в себя несколько товаров, и если мы проанализируем другие поля,

используемые в таблице `order_details` — `UnitPrice`, `Quantity` и `Discount`, тогда мы четко поймем, что эти поля обозначают свойства определенного товара, а не заказ в целом. Также поле `OrderId` не может использоваться само по себе в качестве первичного ключа для таблицы `order_details`. Решение состоит в том, чтобы объединить `ProductId` и `OrderId` в составной ключ, тем самым гарантируя, что данные, содержащиеся в столбцах `UnitPrice`, `Quantity` и `Discount`, будут соответствовать уникальному и определенному заказу и уникальному и определенному товару в этом заказе.

Типы данных

Ранее в этой главе мы описывали концепцию метаданных, которые представляют собой детальную информацию обо всех объектах системы. При разработке базы данных с использованием SQL для каждого используемого столбца должен быть назначен определенный *тип данных*. Типы данных могут незначительно отличаться в зависимости от используемой версии SQL. Однако, как правило, у вас будут числовые, символьные или текстовые типы данных, дата и время, а также логические значения. Рассмотрим каждый из них.

Числовые типы данных

В состав числовых данных входят целочисленные данные, которые служат для отображения целых чисел. Обычно, когда используется целочисленный тип данных, он имеет ограничения на длину. Напомним, что изображенная на рис. 7 таблица содержала сведения о пациентах. Для столбца `Weight` (Вес) разумно использовать целочисленный тип данных с ограничением до трех цифр. Почему?

1. Вполне достаточно округлить значение в большую и меньшую сторону до ближайшего значения фунта или килограмма и не использовать десятичные знаки.

2. Вряд ли нам понадобится более трех цифр для указания значения веса в фунтах.

Когда целочисленные данные не подходят и возникает необходимость в более точном числовом формате, мы можем использовать формат чисел с плавающей запятой. Как и целочисленные данные, они могут быть ограничены по длине.

ЧИСЛОВЫЕ ДАННЫЕ

Целые числа	Числа с плавающей запятой*
5	30,5
6176	14,65
47261	5,634
531	365,1
90	0,437
1	15347,45

Рис. 16

ПРИМЕЧАНИЕ

Для типов данных, допускающих более длинные диапазоны цифр, символов и т. д., требуется больший объем памяти в байтах. SQL также допускает денежные (финансовые) типы данных.

Символьные, или текстовые, типы данных

Символьные, или текстовые, типы данных (еще их называют «строковые») могут хранить строки символов как фиксированной, так и переменной длины. Например, если один из столбцов вашей базы данных содержит стандартные шестизначные почтовые индексы Канады (которые включают как цифры, так и буквы), то вам необходимо использовать символьный, или текстовый, тип данных со строкой фиксированной длины из шести символов. Если вы создавали столбец для хранения имени или фамилии клиента, вам необходимо использовать строку переменной длины с ограничением по максимальной и минимальной длине.

* В англоязычных странах целая часть отделяется от дробной точкой, а в России — запятой. — *Примеч. ред.*

СИМВОЛЬНЫЕ, ИЛИ ТЕКСТОВЫЕ, ДАННЫЕ

CanadianZipCode	FirstName	LastName
L4K8R3	Ronald	Dalton
V0S0N2	Clara	Abramson
H7L9N0	Joseph	Scalia
L3M0L7	Benjamin	Dreadnaught
E6K5T8	Harold	Mercedes
E7K3C5	James	Rockefeller

Рис. 17

ПРИМЕЧАНИЕ

В рассмотренных ранее примерах использовались относительно короткие текстовые данные, такие как имена и адрес. Многие базы данных содержат текстовые поля, позволяющие хранить гораздо более длинные строки текста. Некоторые структуры баз данных позволяют каталогизировать многостраничные тексты или даже целые книги.

Дата и время

Дата и время — эти данные важны во многих случаях. SQL предлагает пользователям различные форматы даты и времени: YYYY-MM-DD (ГГГГ-ММ-ДД), YYYY-MM-DD HH:MI:SS (ГГГГ-ММ-ДД ЧЧ:МИ:СС), YY-MM-DD (ГГ-ММ-ДД). Вы также можете отформатировать столбец так, чтобы он содержал только год, в четырех- или двухзначном формате. Например, 2019 или просто 19. На рис. 18 показан пример использования данных даты и времени.

ПРИМЕЧАНИЕ

В SQL форматы даты/времени содержат встроенные числовые значения, позволяющие базе данных обрабатывать запросы, где в качестве условия задан интервал времени. Например, если вы хотите узнать, сколько клиентов приобрели определенный товар в период с 1 октября 2020 г. по 31 декабря 2020 г., то с помощью SQL вы сможете сгенерировать и отсортировать эти данные.

ДАТА И ВРЕМЯ

DateOfBirth	CreditCardExpiration	TimeOfDelivery
01/25/1977	08/2023	2019-04-21 08:25:55
09/30/2003	05/2025	2020-12-05 13:30:15
08/15/1999	01/2023	2020-05-10 22:20:36
02/25/1962	11/2022	2019-01-17 10:20:01
09/12/1998	05/2026	2021-06-29 15:21:59
11/03/1959	03/2023	2022-09-03 16:42:26

Рис. 18

Логический тип данных

Логические значения — это данные, принимающие значения `True` (Правда) или `False` (Ложь). Например, если вы отвечаете за секретную операцию для правительства или частного лица, вы можете использовать базу данных для отслеживания уровня допуска сотрудников. Если вам необходимо найти список сотрудников с допусками A, B и D, но не обязательно C, то использование логического анализа данных может значительно упростить процесс. На рис. 19 показан пример использования логических данных.

ЛОГИЧЕСКИЕ ДАННЫЕ

ClearedForTakeOff	InDefault	ConvictedFelon
True	False	False
False	True	False
False	False	False
True	True	True
False	True	False

Рис. 19

ПРИМЕЧАНИЕ

В разных версиях SQL – разные списки распознаваемых типов данных. Некоторые версии SQL, такие как SQL Server и MySQL (описаны далее в этой главе), не дают пользователю возможности присвоить данным тип `Boolean`. Вместо этого они предоставляют тип данных `Bit`, который может быть легко преобразован в логический формат.

Системы управления реляционными базами данных

SQL применяется в целом ряде программных пакетов, известных как *реляционные системы управления базами данных (РСУБД)*. Эти системы упрощают применение SQL, когда пользователь дает команды и задает вопросы базе данных. Наиболее распространенные СУБД — это Oracle Database, Microsoft SQL Server, MySQL, IBM Db2 и SQLite.

Рис. 20

ПРИМЕЧАНИЕ

Программное обеспечение РСУБД часто называют базой данных. Это не совсем верно. Правильнее сказать, что РСУБД предоставляет интерфейс (обычно известный как SQL-браузер) для взаимодействия пользователя с данными, хранящимися в базе данных.

Некоторые РСУБД изначально представлены в графическом виде, другие — в текстовом. РСУБД также различают по подходу к SQL. Ранее в этой главе мы уже упоминали об одной такой аномалии при обработке логических данных. РСУБД действительно различаются по способу представления информации базы данных.

Тот факт, что мы сообщаем РСУБД, какую информацию нам предоставлять, определяет SQL как декларативный язык программирования. Это отличает его

от таких языков программирования, как C++, Java и т. д. Они являются более процедурными, так как с их помощью программа создается от начала до конца (распределяется память, в том числе для существующих справочных файлов, и т. д.). В случае SQL все распределение памяти и другие действия выполняются РСУБД.

Оператор SELECT

Как вы уже знаете, SQL — это язык структурированных запросов, и в течение нескольких десятилетий он сформировал стандарт взаимодействия с реляционными базами данных. `SELECT` — самая распространенная команда SQL. Мы будем с ней работать в главе 4 и далее — до конца книги. SQL-запрос обычно состоит из оператора `SELECT` в сочетании с другими операторами SQL и ссылок на данные, участвующие в запросе. Как и в случае с другими языками программирования, правильная последовательность и выбор операторов SQL важны для создания запроса, который будет правильно интерпретирован браузером SQL. Эта строго заданная структура называется *синтаксисом* запроса.

В следующем примере мы увидим, что синтаксис запроса несколько отличается в различных реализациях РСУБД. Это два очень простых запроса, которые по сути делают одно и то же (возвращают первые десять записей из таблицы `products`), но сформулированы они немного по-разному.

В SQL Server необходимо ввести следующее:

```
SELECT TOP 10 *
FROM
   products;
```

В MySQL это будет выглядеть следующим образом:

```
SELECT *
FROM
   products
LIMIT 10;
```

Если бы мы сформулировали запрос в MySQL так, как в примере для SQL Server, браузер SQL выдал бы *синтаксическую ошибку* и запрос не был бы выполнен. Единственное различие между этими двумя реализациями SQL заключается в том, каким образом мы указываем браузеру SQL ограничить наши результаты десятью первыми записями. В остальном запросы одинаковы. Различия между РСУБД обычно весьма незначительны. Если выражать в процентах, то эти раз-

личия составляют менее 10%. Простые декларативные принципы SQL работают в большинстве РСУБД. Следовательно, если вы будете изучать базовую логику SQL в рамках какой-либо определенной РСУБД, то вы сможете быстро адаптировать свои знания основ SQL к любой другой РСУБД.

Запросы, операторы, условия и ключевые слова

Если ранее вы уже работали с SQL, вы, возможно, встречались с терминами «запрос», «оператор», «условие» и «ключевое слово». `SELECT` — это специальное ключевое слово в SQL, также его называют оператором `SELECT`, условием `SELECT` или запросом `SELECT`. Так в чем же разница? Давайте двигаться от наиболее широкого толкования к более узкому.

Запрос — это набор команд, который возвращает информацию из базы данных в виде записей. Запрос может состоять из нескольких операторов SQL (будут рассмотрены в главе 8 в форме подзапросов). *Оператор* SQL — это выполняемый РСУБД любой допустимый фрагмент кода. Рассмотренные примеры кода являются как действительными операторами SQL (поскольку СУБД позволяет их выполнять), так и запросами (поскольку они возвращают набор записей). *Условие* — это часть запроса, содержащая по крайней мере одно ключевое слово и соответствующую информацию, которая будет использоваться вместе с этим ключевым словом (в данном случае ссылки на поля и таблицы).

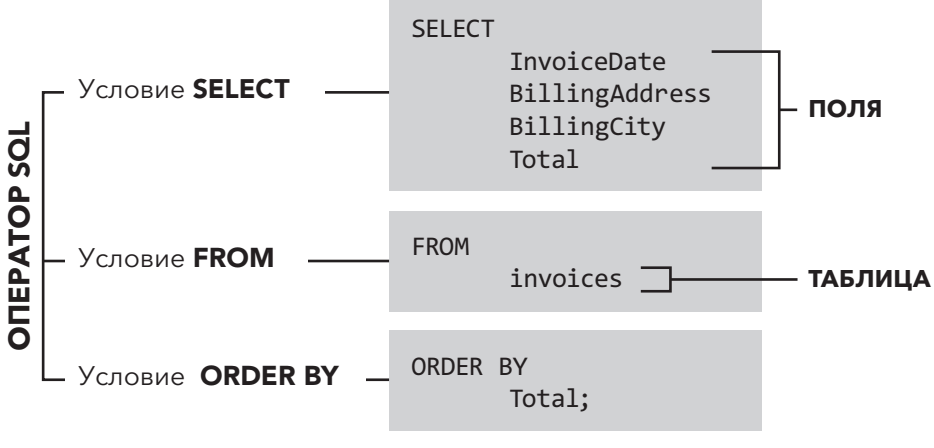


Рис. 21

Слова, написанные ЗАГЛАВНЫМИ буквами, — ключевые слова SQL.

Как показано на рис. 21, оператор SQL может состоять из множества условий, каждое из которых содержит по крайней мере одно ключевое слово, а также ссылки на поля и таблицы.

ПРИМЕЧАНИЕ

На рис. 21 показан пример использования оператора SQL и полного запроса. Запрос может содержать несколько условий, каждое из которых начинается с ключевого слова.

Введение в SQLite

Теперь, когда вы уже знакомы с основами архитектуры базы данных и способами взаимодействия с ней, давайте перейдем к практике решения реальных задач. Существуют различные версии РСУБД. Нам кажется неэффективным рассказывать о функциях применительно к каждой РСУБД в отдельности. В качестве стандартной РСУБД для этой книги мы выбрали SQLite, которую считаем имеющей практическую ценность и доступной для новичков. SQLite — это программный продукт с открытым исходным кодом, поэтому он доступен для использования. Примерно 99% изученного материала, касающегося SQLite, применимо к большинству других РСУБД. Кроме того, SQLite считается одной из наиболее широко используемых систем РСУБД в мире. Она применяется в компьютерах, мобильных устройствах и даже автомобилях [9]. Более подробную информацию можно найти на сайте: `https://www.sqlite.org/index.html`.

Некоторые известные компании, работающие с SQLite.

Рис. 22

ПРИМЕЧАНИЕ

Окончание Lite («легкий», «облегченный») не относится к возможностям данного программного обеспечения, а скорее к его настройке, административным накладным расходам и использованию ресурсов.

Резюме

- Таблица — это двумерная сетка строк и столбцов, содержащая данные.
- Данные могут быть представлены в виде множества различных типов, например строк, чисел или специальных символов.
- Метаданные описывают характер и формат данных, включая минимальную/максимальную длину строки или допустимые числа, буквы или специальные символы.
- Реляционные базы данных могут содержать множество таблиц. Каждая таблица в реляционной базе данных содержит первичный ключ, служащий уникальным идентификатором для таблицы.
- Внешний ключ — это столбец или комбинация столбцов в таблице, значения которых соответствуют первичному ключу в другой таблице.
- Взаимосвязь между таблицами и их первичными и внешними ключами называется схемой базы данных и может быть изображена с помощью схемы «сущность — связь» (ERD).
- Существует множество различных реляционных систем управления базами данных (РСУБД), например Oracle Database, Microsoft SQL Server, MvSQL, IBM Db2 и SQLite. Хотя они во многом отличаются, все они в качестве основы используют язык структурированных запросов.
- `SELECT` — это наиболее распространенная команда SQL.
- Операторы SQL могут содержать несколько условий с использованием разных операторов SQL.
- В этой книге мы взяли за основу SQLite. Однако все полученные навыки можно легко использовать на других платформах РСУБД.

ГЛАВА 2

Инструменты и стратегии SQL

Краткое содержание

- ✓ Настройка среды SQL
- ✓ База данных sTunes
- ✓ Браузер БД для SQLite
- ✓ Самопроверка
- ✓ Стратегии обучения SQL

Мы уже говорили о том, что такое реляционная база данных, каким образом данные структурируются внутри БД и как используется реляционная система управления базами данных (РСУБД) для извлечения информации (посредством написания запросов) и получения необходимых результатов. Теперь, когда мы рассмотрели основные принципы и терминологию, необходимо настроить рабочую среду SQL, чтобы потренироваться в написании запросов. В этой главе вас ждут практические задания и вопросы для самопроверки.

База данных sTunes

Представьте, что вы недавно устроились на работу в компанию, которая специализируется на розничной онлайн-продаже музыки, на должность аналитика данных SQL. Ваша цель — анализ данных о продажах. Вам предоставлен доступ к базе данных компании. Известно, что БД содержит сведения о товаре (в данном случае песни и альбомы), личную информацию о клиентах, записи о сотрудниках и данные о продажах. Руководство желает знать, содержат ли данные в БД sTunes какую-либо полезную информацию о продажах и демогра-

фических характеристиках клиентов и существуют ли способы улучшить или расширить услуги компании. Вам необходимо проанализировать базу данных компании и представить руководству любую найденную информацию. Вы будете выполнять анализ с помощью SQL. Для этого вам необходимо загрузить копию базы данных sTunes на свой компьютер.

ЗАГРУЗКА

Загрузите базу данных компании sTunes

Далее в этой книге мы будем использовать учебную базу данных sTunes. База данных sTunes, а также другие бесплатные цифровые материалы на сайте www.clydebankmedia.com/programming-tech-vault*. Не открывайте sTunes сразу после загрузки. Сохраните БД на своем компьютере.

Браузер базы данных для SQLite

В конце главы 1 мы уже говорили, что будем работать с РСУБД под названием SQLite — произносится как «SQL лайт». Хотя SQLite является конкретной системой управления, или протоколом, для нашей демонстрационной базы данных, нам в любом случае необходимо загрузить специальное приложение (браузер SQL) для «просмотра» нашей базы данных, точно так же как веб-браузер используется для чтения страниц в интернете. SQLite поставляется в комплекте с *DB Browser*. Это удобный визуальный инструмент с открытым исходным кодом; он используется для создания, проектирования и редактирования баз данных, совместимых с SQLite. DB Browser предназначен для пользователей и разработчиков, которые создают базы данных, осуществляют поиск и редактируют данные.

Установка браузера базы данных для SQLite

Чтобы установить DB Browser, перейдите по ссылке www.clydebankmedia.com/programming-tech-vault. Вы увидите несколько ссылок для загрузки, предназначенных для разных операционных систем. Обязательно выберите соответствующую операционную систему (32- или 64-битная Windows, Mac OS, Linux и т. д.). После загрузки нужного дистрибутива установите приложение.

* Для доступа к файлам, нажмите на значок «+» рядом с SQL QuickStart Guide.

Как проверить свои знания в SQL

Лучшее обучение — это практика. Кроме примеров из демонстрационной базы данных в следующих главах этой книги представлены два типа практических заданий для самопроверки. Упражнения в разделах «Практические задания» позволят вам попробовать себя в деле сразу после изучения материала. Это просто. Еще один тип заданий — анализ данных. Эти упражнения более сложные, и вам придется вспомнить предыдущие главы. Подробные решения для каждого контрольного задания вы найдете в приложении I.

Стратегии успеха

Прежде чем мы запустим наше программное обеспечение и откроем базу данных sTunes, я хотел бы дать несколько советов как новичкам, так и экспертам. Я преподаватель с большим опытом, поэтому призываю вас следовать моим советам: это значительно повысит ваши шансы на успех.

Пишите каждый пример кода вручную

Если вы имеете доступ к электронной версии книги, не копируйте и не вставляйте примеры кодов в браузер SQL! Это особенно касается тех, кто еще не изучал SQL. Я категорически против того, чтобы при изучении языка программирования копировали существующие примеры кода или упражнения в РСУБД, а затем анализировали их исполнение. При копировании и вставке сложно запомнить синтаксис и орфографию, поэтому пишите запросы самостоятельно. При вставке кода из других источников могут возникать ошибки форматирования, которые трудно обнаружить. Например, наличие кавычек, вставленных из текстового процессора типа Word, часто вызывает синтаксическую ошибку, поскольку браузер БД не интерпретирует эти символы как одинарные кавычки. Ошибки, которые мы делаем, так же значимы, как и наши достижения. Если при выполнении оператора SQL возникает синтаксическая ошибка, иногда проще удалить весь запрос и написать его заново.

Преобразование вопросов на естественном языке в запросы SQL

Составляя запрос, постарайтесь получить ответ на реальный вопрос. Вместо: «Сколько записей в таблице клиентов?» спросите: «Сколько у нас клиентов?».

Эта книга поможет вам использовать SQL, чтобы продвинуться в карьере, и значительно расширит доступный вам набор инструментов. Вопросы, которые интересуют ваших менеджеров и коллег, имеют практическое значение. Поэтому важно изучить технику преобразования вопроса на разговорном языке в запрос SQL, а результат — обратно в ответ на разговорном языке.

Эту книгу вполне можно считать справочником

Мы предположили, что некоторые из наших читателей уже знакомы с SQL и поэтому захотят сразу перейти к определенному разделу. Мы постарались включить операторы SQL (они обсуждаются в каждой главе) в заголовки глав, чтобы вам, нашим читателям, было удобно ориентироваться в тексте или быстро отыскать необходимый раздел. Также для удобства мы добавили приложение со списком ключевых слов SQL и некоторыми примерами. Мы надеемся, что эта книга станет для вас настольной и вы будете к ней периодически обращаться для получения полезной информации.

ВНИМАНИЕ

Сначала мы рекомендуем прочитать главу 3, так как в ней будет описано программное обеспечение, необходимое для выполнения всех приведенных в книге примеров запросов.

Резюме

- Эта книга — справочник, который позволяет сразу писать запросы SQL.
- Для достижения максимального результата эту книгу рекомендуется изучать, запустив программное обеспечение для работы с базами данных, а также используя учебную базу данных.
- Для анализа вам будет предоставлен пример реальной базы данных — sTunes.
- DB Browser для SQLite — это бесплатная общедоступная программа для работы с базами данных. Она проста в использовании и полезна для изучения файлов базы данных. Эту программу можно запустить в среде Windows или macOS.
- Мы предлагаем два варианта самопроверки. Ответы на контрольные вопросы вы найдете в приложении I.
- Для получения максимальной пользы мы рекомендуем все SQL-запросы переписывать вручную (а не копировать) и писать любой SQL-запрос, ориентируясь на вопрос на естественном языке.

ГЛАВА 3

Работа с базой данных в SQLite

Краткое содержание

- ✓ Запуск программного обеспечения SQL
- ✓ Открытие файла базы данных
- ✓ Вкладка Database Structure (Структура базы данных)
- ✓ Вкладка Browse Data (Просмотр данных)
- ✓ Вкладка Execute SQL (Выполнить SQL-запрос)
- ✓ Просмотр результатов запроса
- ✓ Контрольные вопросы

В этой главе вы научитесь пользоваться браузером SQL для SQLite — DB Browser.

Программное окружение

Чтобы начать анализ базы данных sTunes, необходимо выполнить подготовительную работу. Теперь, когда вы установили DB Browser для SQLite и загрузили учебную базу данных sTunes, можно приступать к работе!

1. Запустите DB Browser для приложения SQLite.
   a) **Пользователям Mac:** перейдите в Finder и в папке Applications (Приложения) дважды щелкните на DB Browser для SQLite.

б) **Пользователям Windows:** перейдите в меню Start (Пуск) и в списке установленных программ выберите DB Browser для приложения SQLite.

2. По умолчанию отобразится следующее окно (рис. 23).

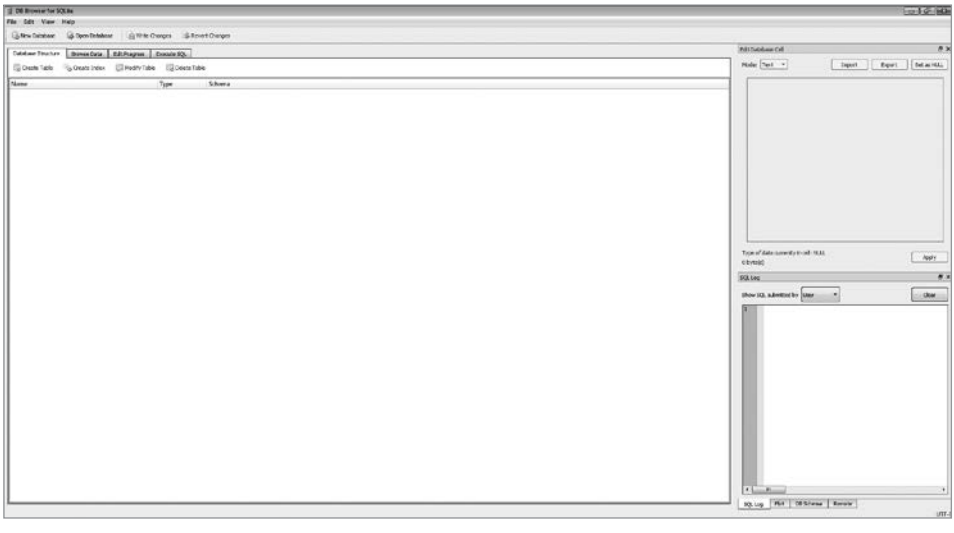


Рис. 23

Открытие базы данных sTunes

3. В открытом DB Browser выберите пункт Open Database (Открыть базу данных).

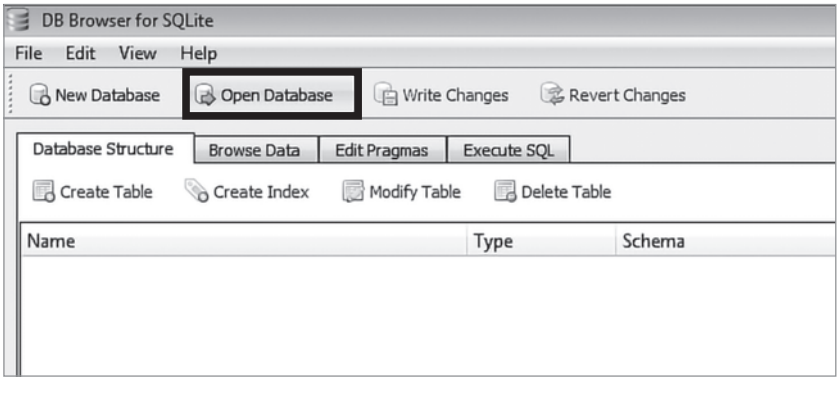


Рис. 24

4. Откроется диалоговое окно Choose a Database File (Выбрать файл базы данных). Перейдите в папку, в которую вы загрузили учебную базу данных sTunes, и нажмите кнопку Open (Открыть).

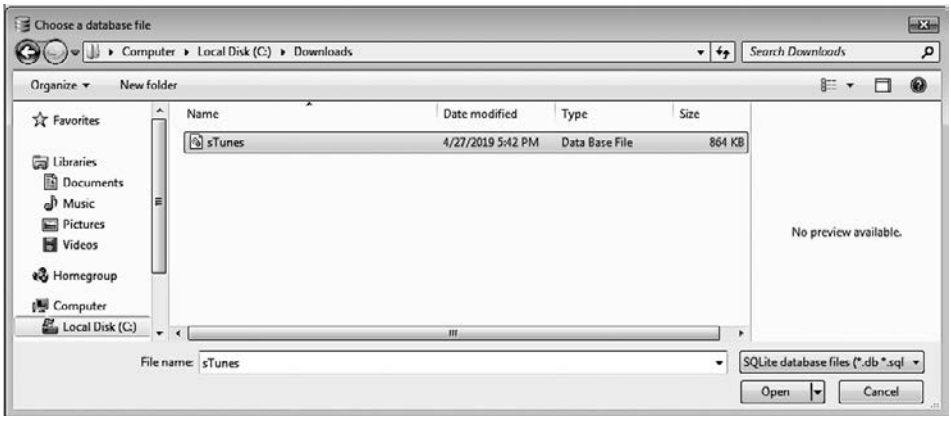


Рис. 25

Структура базы данных

При открытии файла на вкладке Database Structure (Структура базы данных) отображаются таблицы из учебной базы данных.

Теперь, когда мы можем исследовать структуру базы данных, мы получим намного больше информации о нашей вымышленной компании sTunes. Как только ИТ-отдел предоставит нам доступ, мы получим возможность ознакомиться с базой данных. Прежде чем мы начнем писать SQL-запросы, рекомендуется посмотреть, какие типы данных нам доступны.

Прежде всего мы знаем, что база данных sTunes состоит из тринадцати таблиц. Если нажать на стрелку, расположенную слева от имени таблицы, отобразятся поля, содержащиеся в каждой таблице. В таблице `albums` (рис. 27) это поля `AlbumId`, `Title` и `ArtistId`.

Поле `AlbumId` — *целочисленный* тип данных. Это означает, что поле содержит числовые данные (числа).

Поле `Title` — *символьный* тип данных (известный как `NVARCHAR`). Это означает, что поле содержит символы, или нечисловые данные.

Поле `ArtistId` — также *целочисленный* тип данных.

НАПОМИНАНИЕ

В начале главы 1 мы изучали основную терминологию этой книги. Поля, которые мы видим, – это столбцы каждой таблицы. Данные, расположенные в строках таблицы, – это записи.

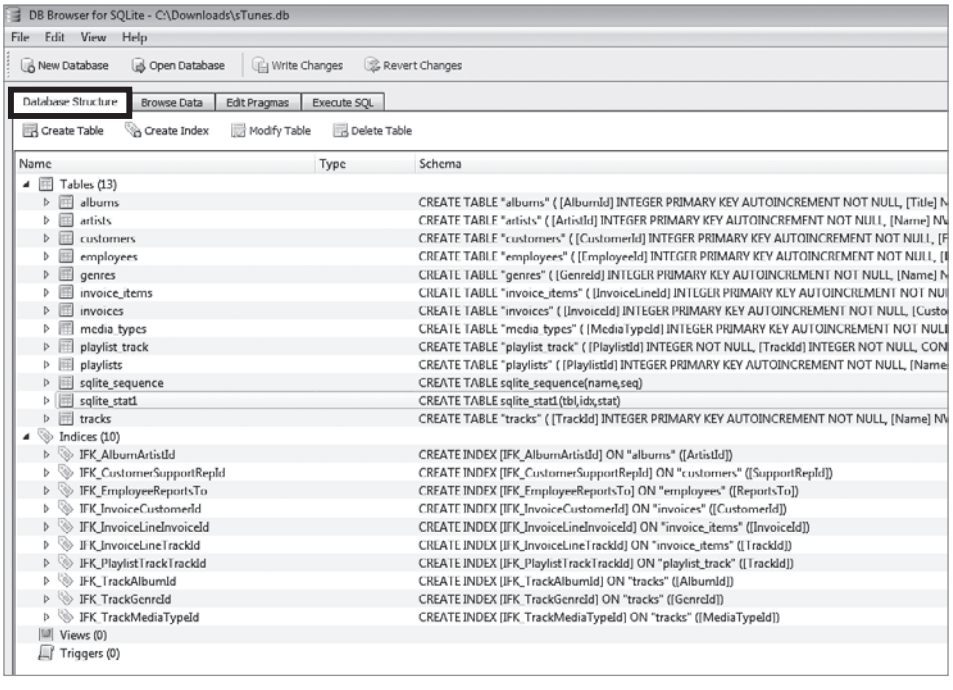


Рис. 26

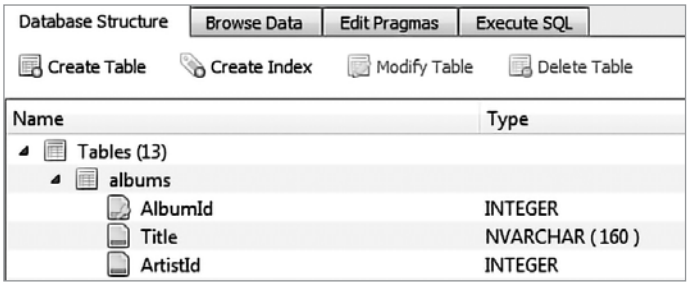


Рис. 27

НАПОМИНАНИЕ

В главе 1 мы уже рассказывали о метаданных и различных типах данных, а также описывали структуру базы данных. Поле Type на панели браузера нашей базы данных (рис. 27) – хороший пример метаданных. Тип данных для каждого поля определяется исходя из практической задачи, которая выполняется посредством данного поля.

Просмотр индивидуальных записей

Вкладку Browse Data (Просмотр данных) можно использовать для просмотра записей в каждой таблице. Чтобы переключаться между таблицами и просматривать данные, используйте раскрывающийся список.

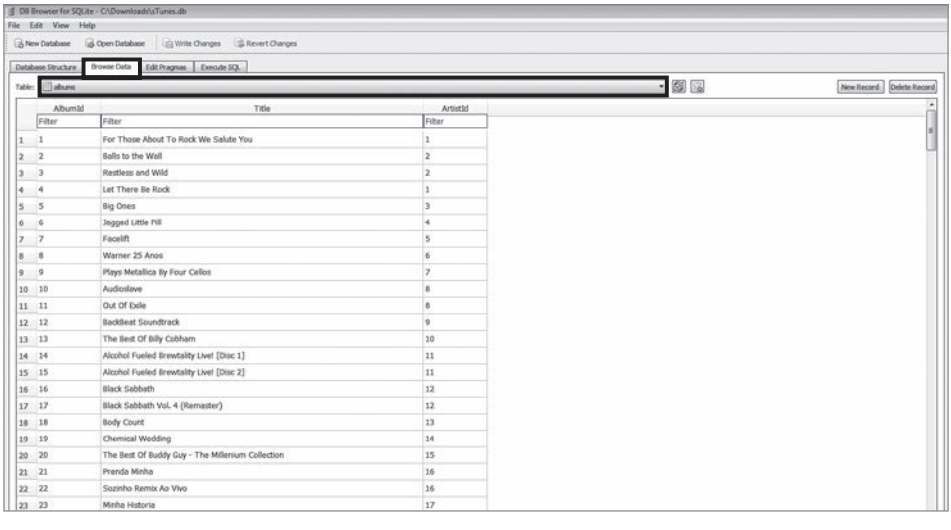


Рис. 28

На вкладке Browse Data (Просмотр данных) (рис. 28) отображаются данные, хранящиеся в таблице `albums`. На вкладке Database Structure (Структура базы данных) отображаются столбцы. Также мы видим фактические данные, содержащиеся в этих столбцах.

В строке 1 в каждом столбце таблицы `albums` содержатся следующие данные:

```
AlbumId = 1
Title = "For Those About to Rock We Salute You"
ArtistId = 1
```

НАПОМИНАНИЕ

Столбцы, которые вы видите, называются полями. Каждая строка представляет собой отдельную запись.

Изучая эту таблицу, мы также можем узнать немного о других таблицах нашей базы данных. В главе 1 мы рассмотрели концепцию внешнего ключа. `AlbumId` — первичный ключ для таблицы `albums` (для каждой записи имеется уникальный номер). Также `ArtistId` принимает целое число вместо фактического имени исполнителя. Это означает, что, вероятно, существует другая таблица, содержащая фактическое имя исполнителя (в формате символьного типа данных), и что `ArtistId`, скорее всего, является внешним ключом.

НАПОМИНАНИЕ

Внешний ключ – это поле в таблице, которое служит для указания в одной таблице на первичный ключ в другой таблице. Если мы проанализируем таблицу artists, мы можем определить, что ArtistId – первичный ключ для таблицы artists, поэтому он является внешним ключом для таблицы albums.

Вкладка Execute SQL

На вкладке Execute SQL (Выполнить SQL-запрос) мы пишем операторы SQL. Вкладка содержит три панели окон основных компонентов: *панель запросов* (Query Pane), *панель результатов* (Results Pane) и *панель сообщений* (Messages Pane). Давайте рассмотрим назаначение этих трех панелей с помощью следующего кода SQL на панели запросов (рис. 29):

```
SELECT
   *
FROM
   albums;
```

Данный пример задает выборку всех полей (символ * обозначает «все поля») из таблицы `albums`. Нажмите кнопку выполнения, расположенную над панелью запросов, и ниже, на панели результатов, отобразятся поля и данные этих полей.

Как показано на рис. 29, расположенная слева кнопка воспроизведения выполняет все введенные операторы SQL. Расположенная справа кнопка воспроизведения выполняет только оператор SQL, на котором находится курсор (одновременно выполняется только один оператор).

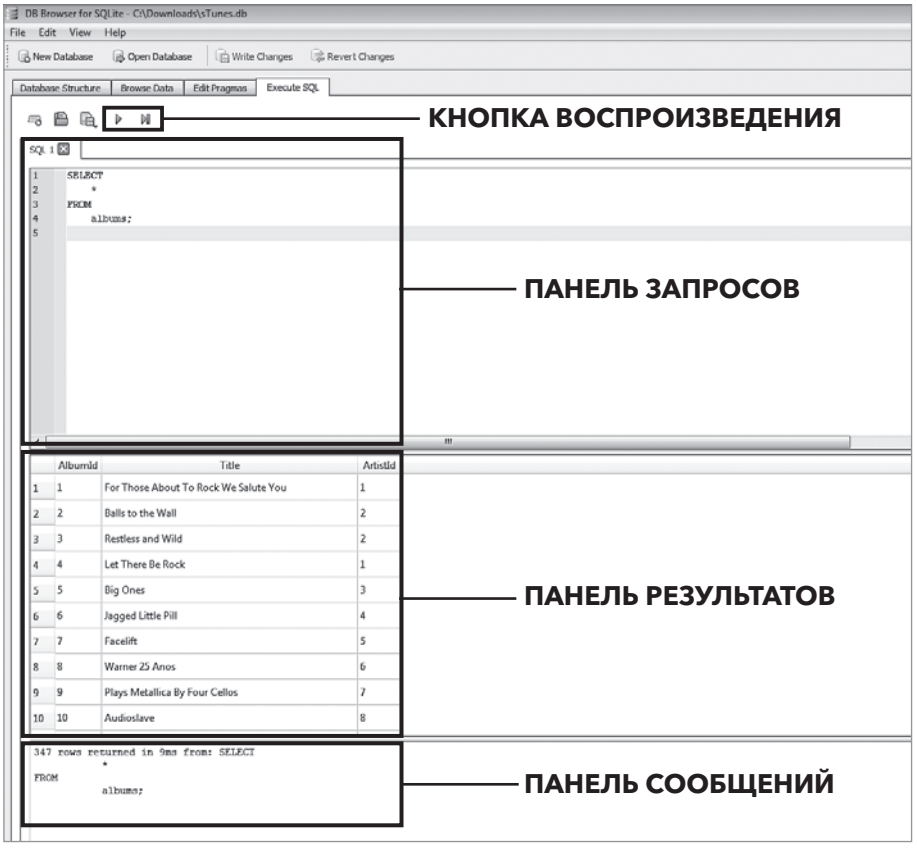


Рис. 29

ПРИМЕЧАНИЕ

Поскольку пока мы изучаем только базовые запросы, мы будем запускать только один оператор SQL. В более сложных запросах мы можем использовать несколько операторов, которые будут выполняться одновременно. Возможность выполнения операторов по отдельности очень полезна в запросах, содержащих несколько операторов. Если на панели выполнения содержится только один оператор SQL, то обе кнопки воспроизведения по сути выполняют одно и то же действие.

На панели результатов приведены выходные данные, сгенерированные нашим запросом. При получении большого количества результатов, которые не помещаются на одном экране (рис. 30), в браузере появится полоса прокрутки.

Есть еще один способ определить, сколько результатов (в данном случае на экране мы видим только пять) выдает данный запрос. Ниже панели результатов расположена панель сообщений, в которой отображается информация о нашем запросе.

- Количество строк, возвращаемых нашим оператором SQL.
- Время, необходимое на выполнение нашего запроса или оператора SQL.
- Сообщения об ошибках, если наш оператор SQL содержит ошибки.

	AlbumId	Title	ArtistId
1	1	For Those About To Rock We Salute You	1
2	2	Balls to the Wall	2
3	3	Restless and Wild	2
4	4	Let There Be Rock	1
5	5	Big Ones	3

Рис. 30

```
347 rows returned in 9ms from: SELECT
            *
FROM
            albums;
```

Рис. 31

В первую очередь из сообщения мы видим, что в результате нашего запроса вернулось 347 строк. Это сообщение полезно, в случае если бы мы не заметили появление полосы прокрутки. Далее мы прочтем, что наш запрос был выполнен за 9 миллисекунд (сокращенно 9 мс). Почему так важно знать, сколько времени требуется на обработку запроса? В данном случае, может, это и не особо важно. Однако при работе с большими базами данных и с более сложными запросами возврат данных может занять гораздо больше времени. Время выполнения запроса также зависит от оптимальности построения и структуры базы данных. Более подробно о структуре базы данных при работе с несколькими таблицами мы поговорим в главе 6.

ПРИМЕЧАНИЕ

Панель сообщений – место, где также будут отображаться сообщения об ошибках. Если ваш запрос не выполняется корректно, необходимо проверить панель сообщений.

Контрольные вопросы

ПРИМЕЧАНИЕ

Теперь пришло время первых контрольных вопросов! Они приводятся в конце каждой главы. Контрольные вопросы и практические задания – это основа закрепления теоретических знаний.

Используя вкладки Database Structure (Структура базы данных) и Browse Data (Просмотр данных), ответьте на следующие вопросы.

1. Сколько таблиц в нашей базе данных?
2. Сколько полей в таблице *tracks*?
3. Какие типы данных указаны в этой таблице?
4. Как выглядят данные в таблице?

НАПОМИНАНИЕ

В приложении I в конце книги приведены ответы на контрольные вопросы.

Резюме

- DB Browser для SQLite (выбранное нами программное обеспечение) открывается, как и любая другая компьютерная программа.
- Нажав кнопку Open Database (Открыть базу данных), вы сможете выбрать файл базы данных на вашем компьютере.
- Структуру файла базы данных вы можете изучить на вкладке Database Structure (Структура базы данных).
- Вкладка Browse Data (Просмотр данных) используется для просмотра отдельных записей в таблице, которые можно выбрать из раскрывающегося меню.
- Вкладку Execute SQL (Выполнить SQL-запрос) используют, чтобы написать, а затем выполнить запрос SQL.
- Панель результатов содержит данные, возвращаемые запросом.
- Панель сообщений содержит информацию о запросе, включая количество возвращенных строк, время выполнения запроса и сообщения о возможных ошибках.

ЧАСТЬ II

ОПЕРАТОРЫ SQL

ГЛАВА 4

Работа с запросами

Краткое содержание

- ✓ Методы записи запросов
- ✓ Основы SQL-запросов
- ✓ Использование псевдонима
- ✓ Сортировка данных по алфавиту
- ✓ Ограничение выборки данных
- ✓ Контрольные вопросы

SQL — это мощный и надежный язык, предлагающий пользователю широкий спектр команд. В действительности язык SQL содержит их гораздо больше, чем мы показываем в этой книге. Самые простые и удобные команды вы сможете сразу же начать изучать на практике, а не только в теории. В следующей главе вы познакомитесь с основами создания хорошего запроса и форматирования результатов. Изучив эту главу, вы научитесь выбирать отдельные поля из конкретной базы данных и отображать их в алфавитном порядке. Давайте начнем!

Добавление комментариев к запросам

Прежде чем составить свой первый SQL-запрос, рассмотрим, как писать комментарии. Комментарии — это простые предложения, которые помогают понять логику вашего SQL-запроса. Использование комментариев в SQL очень полезно, так как они поясняют сложную логику запроса.

В SQL комментарии бывают двух типов. Однострочные начинаются с двух дефисов (--). Пример ниже демонстрирует комментарий, созданный в строке 1 (рис. 32).

Многострочные комментарии начинаются с сочетания символов /* и заканчиваются символами */. Все, что находится между открывающим и закрывающим символами, — это комментарии (рис. 33).

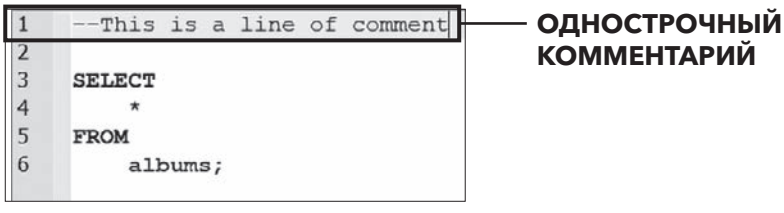


Рис. 32

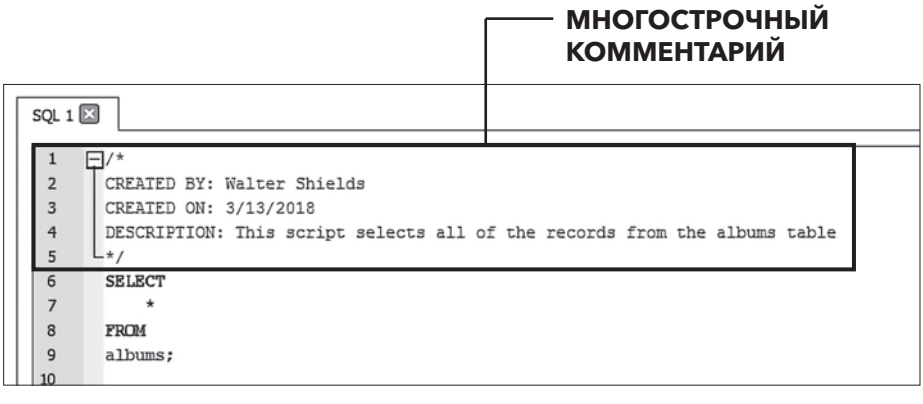


Рис. 33

В примере приведена стандартная информация, которую полезно включать в блок комментариев. Разработчик, дата разработки и описание имеют большое значение для всех, кто сталкивается с SQL-запросом или сценарием.

НА МОЙ ВЗГЛЯД,

разработчики частенько игнорируют комментарии в SQL. В следующих главах мы не используем комментарии только для краткости. При работе с реальными базами данных использование комментариев экономит время, которое пришлось бы потратить на написание дополнительных запросов, чтобы прояснить работу базы данных. Комментарии особенно важны, когда ваши запросы используются другими разработчиками.

Общая структура запроса

Написать запрос — это то же самое, что задать вопрос на любом человеческом языке. Большое значение имеют формулировка, детали и порядок слов. Чем детальнее наш вопрос, тем точнее будет ответ.

При создании SQL-запроса необходимо учитывать следующие пять моментов.

1. С какой базой данных мы работаем?
2. Из какой таблицы в этой базе данных нам необходимо извлечь данные?
3. Какие поля в этой таблице нас интересуют?
4. Хотим ли мы исключить какие-либо данные, отфильтровать или исключить какой-либо диапазон или период времени?
5. Как сформулировать наш запрос одним простым предложением на человеческом языке?

Цель этих вопросов — построить взаимосвязь между человеческим языком, на котором мы обычно говорим или пишем, и языком SQL. Если вы работаете аналитиком данных, то обычные вопросы о бизнесе, которые вам задают, необходимо преобразовать в операторы SQL. После получения результатов запроса следует преобразовать их обратно в доступный для всех человеческий язык. Таков принцип работы.

ПРИМЕЧАНИЕ

Если у вас возникли проблемы с запросом, ответьте на пять вопросов, приведенных выше. И только потом пишите свой запрос.

Пишем свой первый запрос

Чтобы написать свой первый запрос, используйте существующую вкладку Execute SQL (Выполнить SQL-запрос), обозначенную SQL 1, или откройте новую вкладку панели запросов, как если бы вы открывали новую вкладку веб-браузера. Щелкните на значке Open Tab (Открыть вкладку) (рис. 34).

Итак, новая панель запросов открыта! В первую очередь пишем блок комментариев:

```
/*
CREATED BY: <ваше имя>
CREATED ON: <дата>
DESCRIPTION: <краткое описание запроса, например вопрос № 5>
*/
```

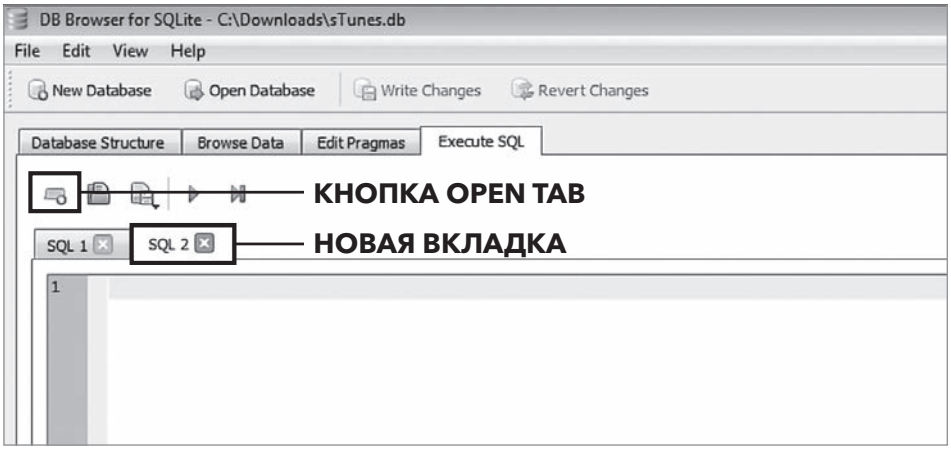


Рис. 34

Закончив блок комментариев, займемся написанием запроса SQL. Для запроса нам необходим понятный вопрос. В главе 2 мы рассмотрели пример рабочего сценария для нашей книги: вы аналитик данных компании sTunes. Как вы уже знаете из главы 3, компания sTunes специализируется на онлайн-продажах музыкальной продукции и имеет цифровую библиотеку исполнителей, треков и альбомов, а также список клиентов, которые приобрели музыкальные произведения. Предположим, служба поддержки sTunes желает разослать новую рекламу всем своим клиентам. В этом случае службе поддержки клиентов необходимо знать, обновлен ли список клиентских контактов, поэтому они наверняка поинтересуются, можем ли мы дать им полный список имен, фамилий и адресов электронной почты клиентов (если имеются) из базы данных. Как мы ответим на этот вопрос? Вернемся к пяти базовым моментам, которые следует учитывать при составлении запроса и о которых речь шла ранее в этой главе.

1. **С какой базой данных мы работаем?**

   В этом случае мы будем работать только с одной базой данных. База данных sTunes уже должна быть открыта в DB Browser. Если вы после изучения главы 3 закрыли браузер, откройте снова его, а также файл базы данных sTunes.

2. **Из какой таблицы в этой базе данных нам необходимо извлечь данные?**

   Нам необходимо получить информацию о клиентах. Просматривая вкладку Database Structure (Структура базы данных), мы видим, что у нас есть таблица с именем `customers (клиенты)`. Это то, что нам нужно!

3. **Какие поля в этой таблице нас интересуют?**

   Ответить на этот вопрос можно, заглянув во вкладку Browse Data (Просмотр данных). Щелкните на этой вкладке и в раскрывающемся меню выберите таблицу `customers` — таблица содержит поля для имени, фамилии и электронной почты.

4. **Хотим ли мы исключить какие-либо данные, отфильтровать или исключить какой-либо диапазон или период времени?**

   В данном случае служба поддержки sTunes желает получить список всех клиентов, поэтому лучше ничего не исключать.

5. **Как сформулировать наш запрос одним простым предложением на человеческом языке?**

   С помощью запроса нам необходимо осуществить выборку следующей информации из таблицы `customers`: имя, фамилия и адрес электронной почты.

Сначала добавьте блок комментариев, затем добавьте условие `FROM customers`. Оно определяет, в какой таблице искать данные. Далее перед условием `FROM` введите оператор `SELECT` и необходимые имена полей из таблицы `customers`. Имя каждого поля отделяется запятой. Полученный код будет выглядеть следующим образом:

```
/*
CREATED BY: Уолтер Шилдс
CREATED ON: 03/13/2018
DESCRIPTION: Данный запрос осуществляет выборку полей имени, фамилии,
электронной почты из таблицы customers (клиентов).
*/

SELECT
   FirstName,
   LastName,
   Email
FROM
   customers;
```

Запустите запрос на выполнение, щелкнув на кнопке воспроизведения Execute SQL (Выполнить SQL-запрос), расположенной на панели меню. Результаты запроса отобразятся ниже на панели результатов (рис. 35). Панель сообщений также показывает, что в результате выполнения запроса вернулось 59 строк (или записей) за 3 миллисекунды.

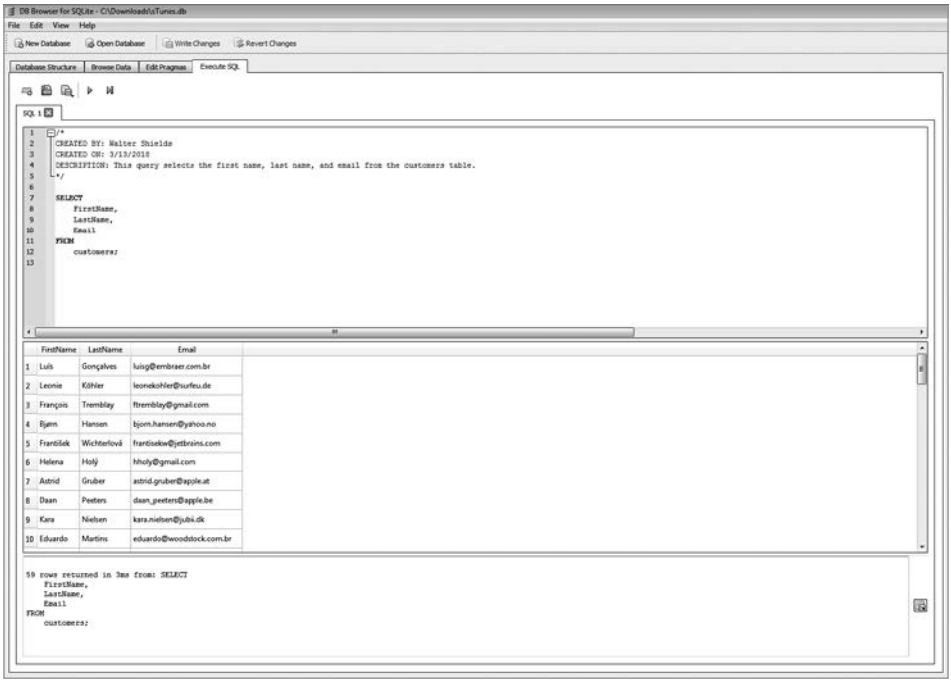


Рис. 35

Практические задания

✶ Добавьте в запрос еще одно поле из таблицы `customers`. В список рассылки добавьте поле `Company` или `Phone`. Не забудьте запятую!

Синтаксис и соглашение о кодировании

Как уже упоминалось в главе 1, все запросы должны соответствовать определенному синтаксису, чтобы их мог понять браузер SQL. Однако при написании запросов необходимо учитывать еще кое-что. Важно, чтобы другие пользователи базы данных могли легко понять ваши запросы. Набор принципов, которые задают стиль, методы написания запросов и т. д., известен как *соглашение о кодировании*. Для разных сред баз данных соглашения о кодировании различаются. В этом разделе мы приводим соглашения о кодировании, используемые в этой книге.

В предыдущих примерах после ключевого слова SELECT мы использовали символ * вместо указания отдельных полей. С помощью специального символа * осуществляется выборка всех записей из таблицы. В некоторых случаях этот символ полезен. Однако чаще рекомендуется указывать необходимые поля.

Здесь точку с запятой в конце оператора ставить необязательно, так как мы пишем только один оператор SQL. Точка с запятой обозначает конец оператора SQL. Поскольку большинство SQL-запросов, используемых в этой книге, представляют собой отдельные операторы, точку с запятой мы будем в дальнейшем опускать.

В нашем примере в условии SELECT мы определили три поля для отображения. Каждое поле необходимо отделять запятой (кроме последнего). Отсутствие запятой между полями или наличие ее после последнего поля — это распространенные синтаксические ошибки, о которых вы получите сообщение на панели результатов.

Обратите внимание, что код состоит из нескольких строк. Также запрос можно написать в одной строке, и браузер SQL по-прежнему распознает код и вернет результаты. Однако запросы рекомендуется разделять на условия, при этом каждое условие необходимо писать с новой строки. В следующих главах наши запросы станут намного длиннее и будут содержать несколько условий. Использование отступов и пробелов в запросах повышает удобочитаемость и значительно упрощает понимание сложной логики запроса.

НАПОМИНАНИЕ

Условие – это часть инструкции SQL, которая начинается со специального ключевого слова (SELECT, FROM и т. д.) и может включать дополнительные параметры и операторы.

Использование псевдонима

Как правило, технический язык базы данных отличается от общепринятого языка. Часто вам придется работать со старыми базами данных или базами данных, имена полей которых давно не обновлялись. *Псевдонимы* SQL используются для присвоения таблице или столбцу в таблице временного имени. Псевдонимы часто используются для того, чтобы сделать имена столбцов более удобочитаемыми. Псевдоним существует только на время выполнения запроса.

В следующем примере мы рассмотрим различные способы создания псевдонима для выбранных имен полей из таблицы `customers`. Псевдоним в базе данных всегда указывают сразу после имени поля. Псевдонимы обычно связаны с ключевым словом `AS`, однако в большинстве реализаций РСУБД ключевое слово `AS` между именем поля и псевдонимом использовать необязательно.

```
/*
CREATED BY: Уолтер Шилдс
CREATED ON: 13.03.2018
DESCRIPTION: Данный запрос осуществляет выборку полей имени, фамилии,
электронной почты и номера телефона из таблицы customers (клиенты)
и демонстрирует четыре различных способа использования псевдонима.
*/

SELECT
   FirstName AS 'First Name',
   LastName AS [Last Name],
   Email AS EMAIL
   Phone CELL
FROM
   customers
```

В данном запросе для первых трех полей мы использовали ключевое слово `AS`. Затем для поля `Phone`, которое мы переименовали в `CELL`, ключевое слово `AS` мы опустили. Если созданный вами псевдоним содержит несколько слов (например, имя и фамилию), его необходимо разграничить, в данном случае либо одинарными кавычками `''`, либо квадратными скобками `[]`, как показано в примере. Поскольку псевдонимы `EMAIL` и `CELL` представляют собой отдельные слова, нет необходимости их заключать в кавычки или скобки.

ПРИМЕЧАНИЕ

В SQL существует множество вариантов синтаксиса псевдонимов. Другие РСУБД могут не распознавать все перечисленные здесь варианты псевдонимов. Если при выполнении запроса обнаружена синтаксическая ошибка, проверьте, как вы указали псевдоним.

Как показано на рис. 36, в расположенных слева выходных данных имена полей не меняются. Расположенные справа выходные данные показывают, как при использовании псевдонимов изменились имена столбцов. При добавлении псевдонима данные в базе данных не изменяются. Псевдонимы изменяют только способ отображения полей на панели результатов.

БЕЗ ПСЕВДОНИМА

	FirstName	LastName	Email	Phone
1	Luís	Gonçalves	luisg@embraer.com.br	+55 (12) 3923-5555
2	Leonie	Köhler	leonekohler@surfeu.de	+49 0711 2842222
3	François	Tremblay	ftremblay@gmail.com	+1 (514) 721-4711
4	Bjørn	Hansen	bjorn.hansen@yahoo.no	+47 22 44 22 22
5	František	Wichterlová	frantisekw@jetbrains.com	+420 2 4172 5555
6	Helena	Holý	hholy@gmail.com	+420 2 4177 0449
7	Astrid	Gruber	astrid.gruber@apple.at	+43 01 5134505
8	Daan	Peeters	daan_peeters@apple.be	+32 02 219 03 03
9	Kara	Nielsen	kara.nielsen@jubii.dk	+453 3331 9991
10	Eduardo	Martins	eduardo@woodstock.com.br	+55 (11) 3033-5446

С ПСЕВДОНИМОМ

	First Name	Last Name	EMAIL	CELL
1	Luís	Gonçalves	luisg@embraer.com.br	+55 (12) 3923-5555
2	Leonie	Köhler	leonekohler@surfeu.de	+49 0711 2842222
3	François	Tremblay	ftremblay@gmail.com	+1 (514) 721-4711
4	Bjørn	Hansen	bjorn.hansen@yahoo.no	+47 22 44 22 22
5	František	Wichterlová	frantisekw@jetbrains.com	+420 2 4172 5555
6	Helena	Holý	hholy@gmail.com	+420 2 4177 0449
7	Astrid	Gruber	astrid.gruber@apple.at	+43 01 5134505
8	Daan	Peeters	daan_peeters@apple.be	+32 02 219 03 03
9	Kara	Nielsen	kara.nielsen@jubii.dk	+453 3331 9991
10	Eduardo	Martins	eduardo@woodstock.com.br	+55 (11) 3033-5446

Рис. 36

Практические задания

- Добавьте к запросу еще одно поле и укажите для него псевдоним.
- Попрактикуйтесь в изменении синтаксиса псевдонима, попробуйте исключить ключевое слово `AS`. Проанализируйте результат.

ПРИМЕЧАНИЕ

Не используйте никакие ключевые слова SQL в качестве псевдонимов. Это вызовет путаницу или синтаксические ошибки. РСУБД может интерпретировать данный псевдоним как команду.

Условие ORDER BY

Предположим, что службе поддержки необходим список клиентов sTunes. Целесообразно упорядочить полученные результаты по фамилии клиентов. Чтобы сделать это, надо использовать новое условие после условия `FROM`. Условие `ORDER BY` используется для сортировки данных в порядке возрастания или убывания на основе одного или нескольких столбцов. По умолчанию данные будут отсортированы в порядке возрастания от A до Z. Специальное ключевое слово `ASC`, определяющее порядок сортировки, необязательно. Для сортировки в порядке убывания от Z до A необходимо после сортируемого поля добавить ключевое слово `DESC`. Например, запрос `ORDER BY LastName DESC` сортирует столбец с псевдонимом `LastName` в порядке убывания.

```
/*
CREATED BY: Уолтер Шилдс
CREATED ON: 13.03.2018
DESCRIPTION: Данный запрос осуществляет выборку полей имени, фамилии
и электронной почты из таблицы customers (клиенты), отсортированных
по фамилии (Last Name).
*/

SELECT
   FirstName AS [First Name],
   LastName AS [Last Name],
   Email AS [EMAIL]
FROM
   customers
ORDER BY
   LastName ASC
```

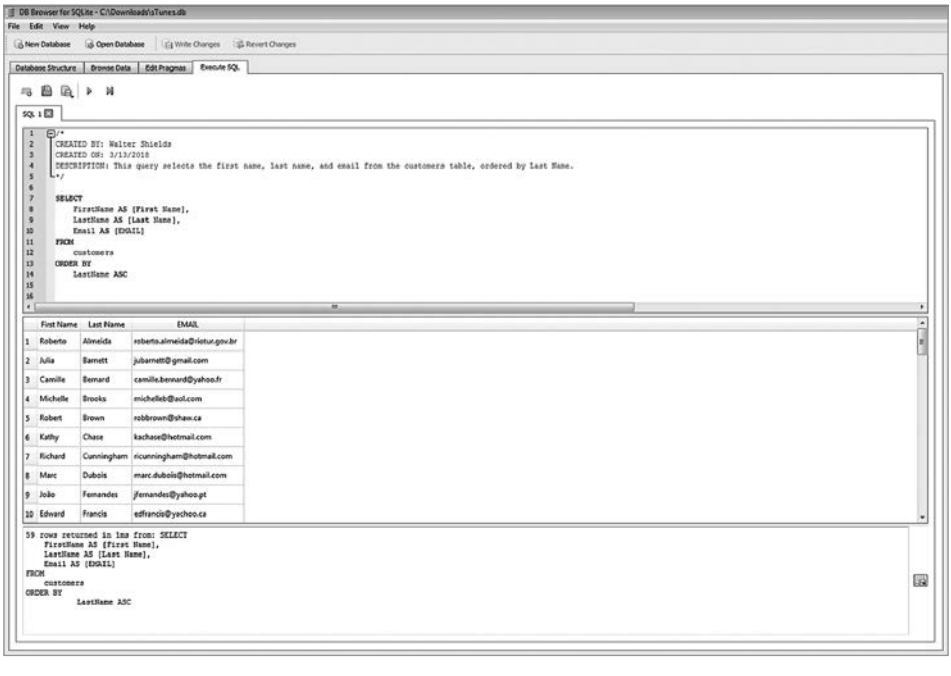


Рис. 37

ПРИМЕЧАНИЕ

Если условие ORDER BY отсутствует, каждый запрос будет возвращать данные в том порядке, в котором они были изначально сохранены в таблице.

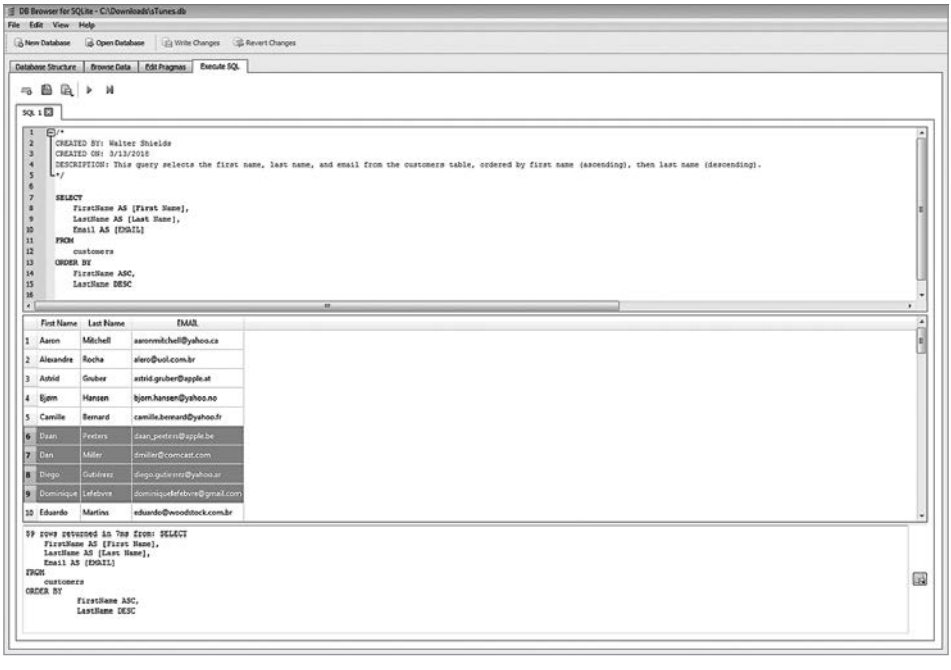


Рис. 38

Также условие `ORDER BY` мы можем использовать для сортировки по нескольким столбцам. В данном примере мы выполним сначала сортировку по имени (по алфавиту), а затем по фамилии (в обратном порядке). Для этого нам необходимо в условии `ORDER BY` указать два поля. При перечислении нескольких полей их необходимо разделять запятыми.

```
/*
CREATED BY: Уолтер Шилдс
CREATED ON: 13.03.2018
DESCRIPTION: Данный запрос осуществляет выборку полей имени, фамилии
и электронной почты из таблицы customers (клиенты), отсортированных
сначала по имени (по возрастанию), а затем по фамилии (по убыванию).
*/

SELECT
   FirstName AS [First Name],
   LastName AS [Last Name],
   Email AS [EMAIL]
FROM
   customers
```

```
ORDER BY
   FirstName ASC,
   LastName DESC
```

Выполним данный запрос и проанализируем имена клиентов, начинающиеся с буквы *D*. Мы видим, что имена расположены по алфавиту, а фамилии — в обратном порядке.

ПРИМЕЧАНИЕ

Если вы используете условие ORDER BY для столбца, в котором имеются пустые строки, вы увидите, что эти значения будут отображаться в начале списка как NULL (если сортировка осуществляется в порядке возрастания).

Практические задания

✶ Измените порядок полей в условии `SELECT` и установите `LastName` первым столбцом, а не вторым. Выполните запрос, в результате которого будут получены упорядоченные записи `LastName`. Стал ли список более читабельным?

Получение ограниченного числа записей с помощью условия LIMIT

В предыдущих примерах мы осуществляли выборку всех записей из таблицы `customers`. Хотя мы ограничили наши записи тремя полями и отсортировали эти поля, в панели сообщений мы видим, что каждый раз мы возвращаем 59 строк. Если нет необходимости в просмотре всех 59 записей, мы можем ограничить наши результаты определенным количеством строк. Это бывает полезно при сортировке по количеству (которое мы продемонстрируем позже), например по самой высокой цене или самым большим продажам. Если после условия `ORDER BY` мы добавим ключевые слова `LIMIT 10`, в результате вернутся только первые десять записей в указанном порядке сортировки. Можно указать любое число записей, которое необходимо отобразить (при условии, что в таблице имеется такое количество записей).

```
/*
CREATED BY: Уолтер Шилдс
CREATED ON: 13.03.2018
DESCRIPTION: Данный запрос осуществляет выборку первых 10 записей из
таблицы customers (клиентов), отсортированных сначала по имени (по
возрастанию), а затем по фамилии (по убыванию).
*/
```

```
SELECT
   FirstName AS [First Name],
   LastName AS [Last Name],
   Email AS [EMAIL]
FROM
   customers
ORDER BY
   FirstName ASC,
   LastName DESC
LIMIT 10
```

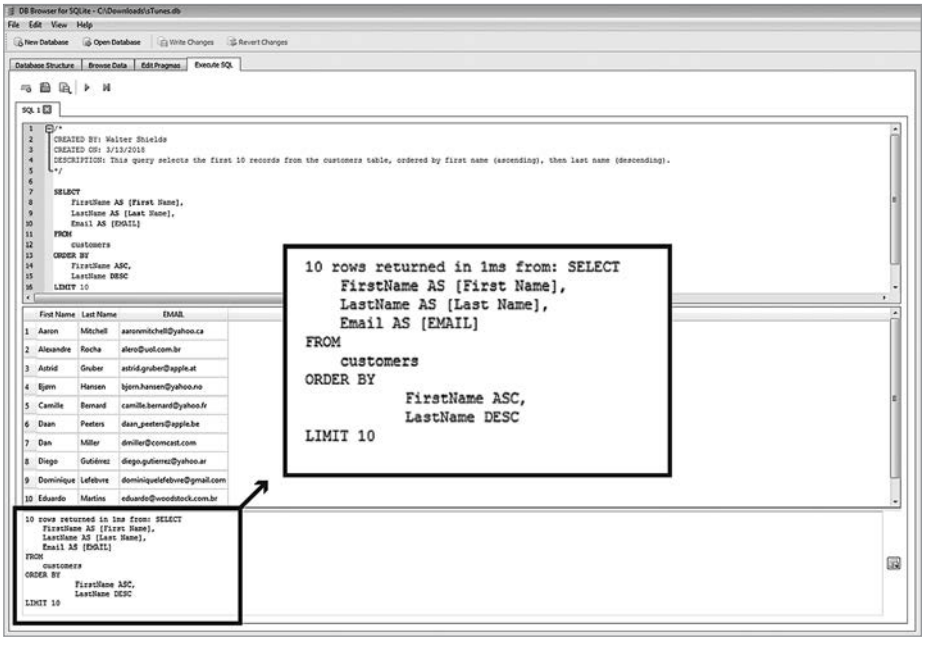


Рис. 39

На нашей панели сообщений (рис. 39) теперь появилось сообщение: 10 rows returned in 1ms (10 строк вернулось за 1 мс). Сначала запрос выполняет условие `ORDER BY`, а затем применяет ограничение.

ПРИМЕЧАНИЕ

Нет необходимости всегда использовать условие LIMIT вместе с ORDER BY. В большинстве случаев имеет смысл упорядочить результаты по определенным критериям, прежде чем ограничивать их. Если условие ORDER BY не используется, результаты оператора LIMIT будут возвращены в том порядке, в котором они были изначально добавлены в таблицу.

Контрольные вопросы

1. Напишите запрос, чтобы узнать количество клиентов, фамилии которых начинаются с буквы B.
2. Какая компания при сортировке в порядке убывания появляется в верхней строке таблицы `customers`?
3. Какое количество клиентов не указали почтовый индекс?

Резюме

- Рекомендуется добавлять комментарии к запросам. Базовый блок комментариев может содержать имя разработчика, дату создания запроса и краткое описание запроса.
- При написании запроса полезно сначала сформулировать вопрос на человеческом языке, а затем определить необходимые ключевые слова и условия.
- Запрос обычно начинается с условия `SELECT`, которое указывает, какие поля выводить из таблицы, указанной в условии `FROM`.
- Псевдонимы в SQL используются для присвоения таблице или столбцу в таблице временного имени. Псевдонимы позволяют сделать имена столбцов более удобочитаемыми. Псевдоним существует только на время выполнения запроса.
- Использование условия `ORDER BY` позволяет сортировать записи в алфавитном порядке (`ASC`) или в обратном порядке (`DESC`).
- С помощью условия `LIMIT` (после условия `ORDER BY`) можно получить ограниченное число записей. Для этого вам необходимо указать количество результатов, которые хотите показать.

ГЛАВА 5

Преобразование данных в информацию

Краткое содержание

- ✓ Операторы в SQL
- ✓ Условие `WHERE`
- ✓ Поиск текста с использованием подстановочных знаков
- ✓ Функция `DATE()`
- ✓ Использование одновременно `OR` и `AND`
- ✓ Оператор `CASE`
- ✓ Контрольные вопросы

На этом этапе обучения мы покажем, как, используя основной оператор `SELECT`, получить любой набор полей из таблицы базы данных и при необходимости упорядочить их. Отображение и сортировка полей — это важный первый шаг обучения. В дальнейшем, чтобы вы могли задавать более конкретные вопросы, понадобятся более точные инструкции. В предыдущей главе в разделе «Контрольные вопросы» мы спросили, сколько фамилий клиентов начинаются с буквы *B*. Если вы выполнили упражнение, то знаете, что в результате выполнения запроса вернулось не очень много записей. В данном случае легко отсортировать данные в алфавитном порядке по фамилии, а затем просто вручную подсчитать, сколько из них начинается на букву *B*. Но что делать, если база данных содержит миллион клиентов? Вы действительно думаете, что сможете вручную подсчитать все фамилии на букву *B*? К счастью, в SQL существуют инструменты, которые не только позволяют сузить

набор результатов (то есть результаты запроса) до очень конкретных данных, но также упорядочить и отфильтровать данные в зависимости от заданных пользователем условий.

ПРИМЕЧАНИЕ

С этого момента мы не будем показывать окна с выходными данными браузера DB Browser (только если мы специально не ссылаемся на функцию браузера). Теперь мы будем показывать результаты наших запросов в табличной форме (рис. 40).

Результат содержит только три строки

	InvoiceDate	BillingAddress
1	1/1/2009 0:00	Theodor-Heuss-Straße 34
2	2/1/2009 0:00	Barbarossastraße 19
3	2/1/2009 0:00	8, Rue Hanovre
	3 строки получено за 3 мс	

Результат содержит более трех строк, но отображает только первые три

	InvoiceDate	BillingAddress
1	1/1/2009 0:00	Theodor-Heuss-Straße 34
2	2/1/2009 0:00	Barbarossastraße 19
3	2/1/2009 0:00	8, Rue Hanovre
...	150 строк получено за 3 мс	

Рис. 40

Операторы сравнения, логические и арифметические операторы

Операторы — это специальные ключевые слова в SQL, которые мы используем вместе с условиями для сравнения значений полей, выбора подмножеств полей или выполнения арифметических операций.* В отличие от уже изученных нами ключевых слов, таких как `SELECT`, операторы не могут существовать как самостоятельные условия SQL, их следует использовать с другими условиями, такими как `SELECT` и `WHERE` (которые мы рассмотрим в этой главе). На рис. 41 показаны три типа операторов, которые мы будем использовать далее в этой главе.

---

* Существует некая путаница в терминологии. В английском языке есть два разных слова, statement и operator, которые чаще всего переводят одинаково — оператор. Встречается также перевод слова operator (в применении к арифметическим, логическим и сравнения) как «операция», а перевод слова statement как «инструкция». Мы будем использовать термин «оператор» как наиболее привычный. — *Примеч. ред.*

ТИПЫ ОПЕРАТОРОВ

ОПЕРАТОРЫ СРАВНЕНИЯ	ЛОГИЧЕСКИЕ ОПЕРАТОРЫ	АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ
= Равно	BETWEEN (Между)	+ Сложение
> Больше, чем	IN (В)	- Вычитание
< Меньше, чем	LIKE (Как)	/ Деление
>= Больше или равно	AND (И)	* Умножение
<= Меньше или равно	OR (Или)	% Остаток от деления
<> Не равно		

Рис. 41

ПРИМЕЧАНИЕ

Для создания более сложных запросов операторы разных типов можно комбинировать. Это позволяет получать данные в зависимости от определенного диапазона или создавать уникальные условия. Рассмотрим наиболее распространенные операторы.

В следующем примере мы рассмотрим, как арифметические операторы используются вместе с условием `SELECT` для увеличения значения поля `Total` из таблицы `invoices` базы данных sTunes. Арифметические операции полезны, когда необходимо считать налоги, надбавки и др.

```
SELECT
   Total AS [Original Amount],
   Total + 10 AS [Addition Operator],
   Total - 10 AS [Subtraction Operator],
   Total / 10 AS [Division Operator],
   Total * 10 AS [Multiplication Operator],
   Total % 10 AS [Modulo Operator]
FROM
   invoices
ORDER BY
   Total DESC
```

	Original Amount	Addition Operator	Subtraction Operator	Division Operator	Multiplication Operator	Modulo Operator
1	25.86	35.86	15.86	2.586	258.6	5
2	23.86	33.86	13.86	2.386	238.6	3
3	21.86	31.86	11.86	2.186	218.6	1
4	21.86	31.86	11.86	2.186	218.6	1
5	18.86	28.86	8.86	1.886	188.6	8
...	412 строк получено за 42 мс					

Рис. 42

Практические задания

✶ Используя приведенный выше пример запроса, отобразите поле `Total` из таблицы `invoices` с добавленным налогом в размере 15%.

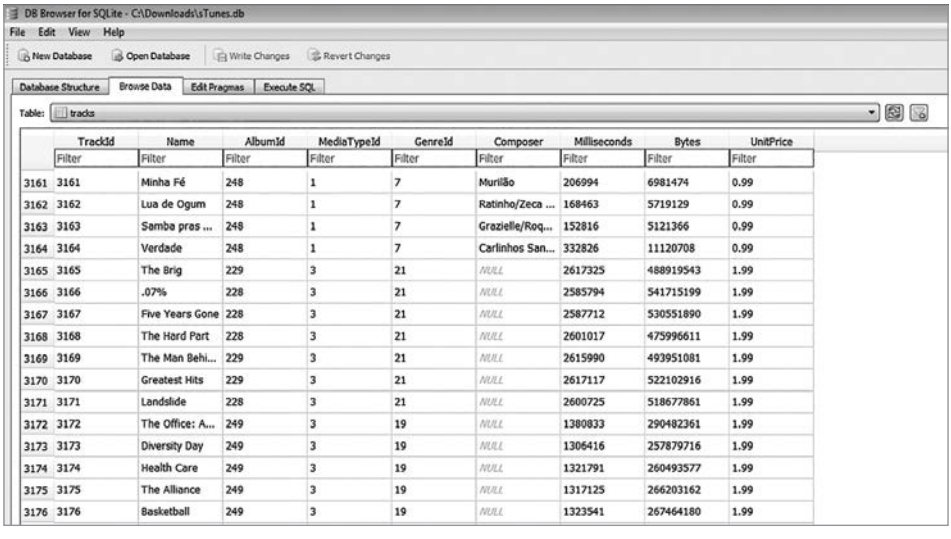
Фильтрация данных (WHERE)

Помимо условия `SELECT` операторы чаще всего используются в условии `WHERE`. С помощью `WHERE` мы можем добавлять к нашим запросам определенные условия, например ограничивать результаты наших запросов в соответствии с необходимыми задачами. Некоторые распространенные типы данных, которые мы можем фильтровать, содержат числа, текст и даты. Для фильтрации данных мы будем использовать `WHERE` вместе с операторами.

Рассмотрим следующий пример. Предположим, что отдел продаж компании sTunes хочет знать, сколько клиентов купили две песни стоимостью по $0,99 каждая. Как ответить на этот вопрос? В главе 3 в разделе «Контрольные вопросы» мы показывали таблицу `tracks`. Из нее мы знаем, что наша компания продает отдельные песни по цене $0,99 и $1,99 (рис. 43).

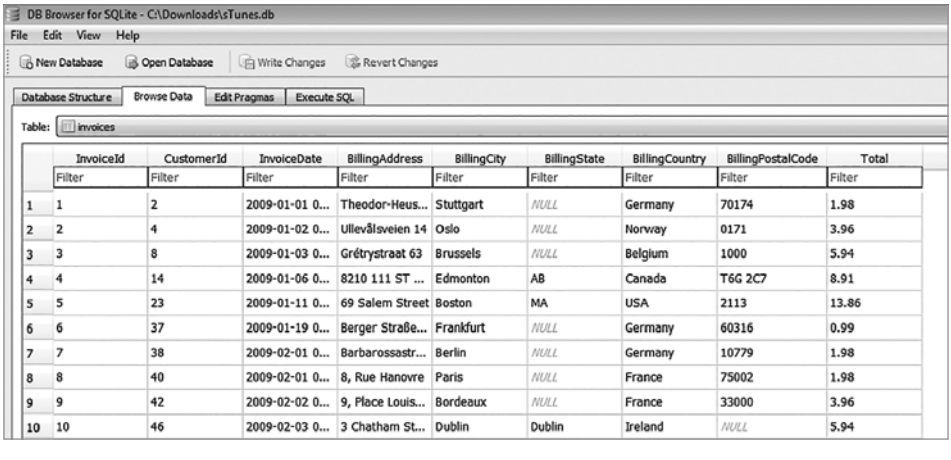
Если мы проанализируем таблицу `invoices` (рис. 44), то увидим общую стоимость заказов в поле `Total`.

Если бы нам понадобилось узнать количество клиентов, которые приобрели всего две песни по $0,99, мы бы отобразили в таблице `invoices` запись на общую сумму $1,98 — за две песни.



	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
3161	3161	Minha Fé	248	1	7	Murilão	206994	6981474	0.99
3162	3162	Lua de Ogum	248	1	7	Ratinho/Zeca ...	168463	5719129	0.99
3163	3163	Samba pras ...	248	1	7	Grazielle/Roq...	152816	5121366	0.99
3164	3164	Verdade	248	1	7	Carlinhos San...	332826	11120708	0.99
3165	3165	The Brig	229	3	21	NULL	2617325	488919543	1.99
3166	3166	.07%	228	3	21	NULL	2585794	541715199	1.99
3167	3167	Five Years Gone	228	3	21	NULL	2587712	530551890	1.99
3168	3168	The Hard Part	228	3	21	NULL	2601017	475996611	1.99
3169	3169	The Man Behi...	229	3	21	NULL	2615990	493951081	1.99
3170	3170	Greatest Hits	229	3	21	NULL	2617117	522102916	1.99
3171	3171	Landslide	228	3	21	NULL	2600725	518677861	1.99
3172	3172	The Office: A...	249	3	19	NULL	1380833	290482361	1.99
3173	3173	Diversity Day	249	3	19	NULL	1306416	257879716	1.99
3174	3174	Health Care	249	3	19	NULL	1321791	260493577	1.99
3175	3175	The Alliance	249	3	19	NULL	1317125	266203162	1.99
3176	3176	Basketball	249	3	19	NULL	1323541	267464180	1.99

Рис. 43



	InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2	2009-01-01 0...	Theodor-Heus...	Stuttgart	NULL	Germany	70174	1.98
2	2	4	2009-01-02 0...	Ullevålsveien 14	Oslo	NULL	Norway	0171	3.96
3	3	8	2009-01-03 0...	Grétrystraat 63	Brussels	NULL	Belgium	1000	5.94
4	4	14	2009-01-06 0...	8210 111 ST ...	Edmonton	AB	Canada	T6G 2C7	8.91
5	5	23	2009-01-11 0...	69 Salem Street	Boston	MA	USA	2113	13.86
6	6	37	2009-01-19 0...	Berger Straße...	Frankfurt	NULL	Germany	60316	0.99
7	7	38	2009-02-01 0...	Barbarossastr...	Berlin	NULL	Germany	10779	1.98
8	8	40	2009-02-01 0...	8, Rue Hanovre	Paris	NULL	France	75002	1.98
9	9	42	2009-02-02 0...	9, Place Louis...	Bordeaux	NULL	France	33000	3.96
10	10	46	2009-02-03 0...	3 Chatham St...	Dublin	Dublin	Ireland	NULL	5.94

Рис. 44

Используя инструменты, которые вы изучили в предыдущей главе, можно написать запрос, который осуществляет выборку всех счетов, а затем упорядочивает их по общей сумме, но для этого необходимо выполнить расчет вручную. Но можно и вставить условие `WHERE` между условиями `FROM` и `ORDER BY` для поиска только тех счетов, которые равны $1,98. Вместе с итоговой суммой добавим несколько других полей, таких как дата выставления счета и адрес, что поможет нам идентифицировать каждый счет. Получим следующий запрос:

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   Total = 1.98
ORDER BY
   InvoiceDate
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-01-01 00:00:00	Theodor-Heuss-Straße 34	Stuttgart	1.98
2	2009-02-01 00:00:00	Barbarossastraße 19	Berlin	1.98
3	2009-02-01 00:00:00	8, Rue Hanovre	Paris	1.98
4	2009-03-04 00:00:00	1 Microsoft Way	Redmond	1.98
5	2009-03-04 00:00:00	1 Infinite Loop	Cupertino	1.98
6	2009-04-04 00:00:00	421 Bourke Street	Sidney	1.98
7	2009-04-04 00:00:00	Calle Lira, 198	Santiago	1.98
8	2009-05-05 00:00:00	Rua a Assunção 53	Lisbon	1.98
9	2009-05-05 00:00:00	Tauentzienstraße 8	Berlin	1.98
10	2009-06-05 00:00:00	Qe 7 Bloco G	Brasília	1.98
…	111 строк получено за 7 мс			

Рис. 45

ПРИМЕЧАНИЕ

Условие WHERE всегда следует после условия FROM, но всегда находится перед ORDER BY. В приведенном выше примере условие WHERE добавлено для возврата всех счетов на сумму $1,98. Знак = – это оператор сравнения.

Практические задания

Используя операторы сравнения, напишите следующие запросы:

- запрос, возвращающий все счета, превышающие значение $1,98;
- запрос, возвращающий все счета, которые больше или равны $1,98;
- запрос, возвращающий все счета, кроме $1,98.

Другой полезный вид операторов — логические. Используя логические операторы, вы можете создавать более сложные и конкретные запросы, которые трудно выполнить с помощью операторов сравнения. Предположим, что вас попросили узнать, какое количество счетов имеется в определенном диапазоне, например от $1,98 до $5.

В данном случае целесообразно использовать оператор `BETWEEN`. Оператор `BETWEEN` задает диапазон для проверки условия. Для определения необходимого диапазона значений вместе с оператором `BETWEEN` используется оператор `AND`. Рассмотрим следующий запрос, который возвращает счета, находящиеся в диапазоне от $1,98 до $5,00.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   Total BETWEEN 1.98 AND 5.00
ORDER BY
   InvoiceDate
```

Проанализируем первые десять результатов этого запроса (рис. 46). Итоговые суммы счетов находятся в диапазоне от $1,98 до $5. Оператор `BETWEEN` включает

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-01-01 00:00:00	Theodor-Heuss-Straße 34	Stuttgart	1.98
2	2009-01-02 00:00:00	Ullevålsveien 14	Oslo	3.96
3	2009-02-01 00:00:00	Barbarossastraße 19	Berlin	1.98
4	2009-02-01 00:00:00	8, Rue Hanovre	Paris	1.98
5	2009-02-02 00:00:00	9, Place Louis Barthou	Bordeaux	3.96
6	2009-03-04 00:00:00	1 Microsoft Way	Redmond	1.98
7	2009-03-04 00:00:00	1 Infinite Loop	Cupertino	1.98
8	2009-03-05 00:00:00	801 W 4th Street	Reno	3.96
9	2009-04-04 00:00:00	421 Bourke Street	Sidney	1.98
10	2009-04-04 00:00:00	Calle Lira, 198	Santiago	1.98
…	178 строк получено за 3 мс			

Рис. 46

в диапазон и параметры, которые вы ему зададите. Другими словами, здесь он будет включать любые значения между 1,98 и 5,00 и равные этим значениям. Для достижения того же результата, используя условие `WHERE` и операторы сравнения, вы можете написать следующее: `Total >= 1.98 AND Total <= 5.00`. Однако в данном случае гораздо проще использовать оператор `BETWEEN`.

ПРИМЕЧАНИЕ

Хотя в предыдущем примере мы используем оператор AND вместе с BETWEEN, оператор AND имеет более широкое применение в качестве логического оператора. Его мы рассмотрим позже.

Практические задания

✶ Используя предыдущий запрос, выполните сортировку по полю `Total`. Выясните, какая сумма счета в этом наборе данных максимальная.

Еще один очень важный оператор SQL — оператор `IN`. Он позволяет определить, совпадает ли значение в условии `WHERE` с какими-то значениями в списке. В предыдущем примере оператор `BETWEEN` возвращал каждое значение в нашей таблице `invoices`, находящееся в диапазоне от 1,98 до 5,00. Оператор `IN` позволяет нам находить указанные значения в наборе данных. Значения разделены запятой и заключены в круглые скобки. Следующий запрос возвращает только суммы счетов-фактур, равные $1,98 или $3,96 (рис. 47).

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   Total IN (1.98, 3.96)
ORDER BY
   InvoiceDate
```

ПРИМЕЧАНИЕ

С помощью оператора = мы можем добавить только одно значение. С помощью оператора IN мы можем добавить сколько угодно значений, разделенных запятыми. Также мы можем использовать оператор IN с текстом (описано в следующем разделе).

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-01-01 00:00:00	Theodor-Heuss-Straße 34	Stuttgart	1.98
2	2009-01-02 00:00:00	Ullevålsveien 14	Oslo	3.96
3	2009-02-01 00:00:00	Barbarossastraße 19	Berlin	1.98
4	2009-02-01 00:00:00	8, Rue Hanovre	Paris	1.98
5	2009-02-02 00:00:00	9, Place Louis Barthou	Bordeaux	3.96
6	2009-03-04 00:00:00	1 Microsoft Way	Redmond	1.98
7	2009-03-04 00:00:00	1 Infinite Loop	Cupertino	1.98
8	2009-03-05 00:00:00	801 W 4th Street	Reno	3.96
9	2009-04-04 00:00:00	421 Bourke Street	Sidney	1.98
10	2009-04-04 00:00:00	Calle Lira, 198	Santiago	1.98
…	168 строк получено за 2 мс			

Рис. 47

Практические задания

- Сколько записей возвращает указанный выше запрос?
- Напишите запрос, в котором перечислены все счета на сумму $13,86, $18,86 и $21,86.

Фильтрация строк

Мы также можем использовать операторы для возврата определенного текста подобно тому, как мы делали с числами. Рассмотрим пример с использованием операторов сравнения. Ответим на следующий вопрос: сколько счетов было выставлено в городе Тусон (Tucson)?

Для этого необходимо структурировать наш оператор `SELECT`. Следующий запрос возвращает все счета, выставленные в городе Тусон:

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
```

```
WHERE
   BillingCity = 'Tucson'
ORDER BY
   Total
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2012-03-011 00:00:00	1033 N Park Ave	Tucson	0.99
2	2011-01-15 00:00:00	1033 N Park Ave	Tucson	1.98
3	2013-09-02 00:00:00	1033 N Park Ave	Tucson	1.98
4	2011-04-19 00:00:00	1033 N Park Ave	Tucson	3.96
5	2011-07-22 00:00:00	1033 N Park Ave	Tucson	5.94
6	2009-06-10 00:00:00	1033 N Park Ave	Tucson	8.91
7	2013-10-13 00:00:00	1033 N Park Ave	Tucson	13.86
	7 строк получено за 1 мс			

Рис. 48

В результате получено только семь счетов для города Тусон.

ПРИМЕЧАНИЕ

При использовании текста в качестве критерия в условии WHERE указанные текстовые значения должны быть заключены в одинарные кавычки (BillingCity = 'Tucson').

НАПОМИНАНИЕ

В предыдущем примере мы использовали символ =, так как требовалось найти только одно значение. Если бы стояла задача получить данные для нескольких городов, мы могли бы использовать оператор IN аналогично тому, как мы использовали его для получения числовых значений.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   BillingCity IN ('Tucson', 'Paris', 'London')
ORDER BY
   Total
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2011-11-08 00:00:00	202 Hoxton Street	London	0.99
2	2012-03-11 00:00:00	1033 N Park Ave	Tucson	0.99
3	2012-08-13 00:00:00	8, Rue Hanovre	Paris	0.99
4	2013-01-15 00:00:00	113 Lupus St	London	0.99
5	2009-02-01 00:00:00	8, Rue Hanovre	Paris	1.98
6	2009-07-06 00:00:00	113 Lupus St	London	1.98
7	2010-04-11 00:00:00	4, Rue Milton	Paris	1.98
8	2010-09-13 00:00:00	202 Hoxton Street	London	1.98
9	2011-01-15 00:00:00	1033 N Park Ave	Tucson	1.98
10	2011-11-21 00:00:00	113 Lupus St	London	1.98
...	35 строк получено за 7 мс			

Рис. 49

Использование оператора LIKE для поиска подстановочных знаков

В предыдущих примерах для поиска необходимого текста мы использовали оператор `=`. SQL также позволяет искать фрагменты текстовой строки с помощью оператора `LIKE`. Это особенно полезно, когда мы не знаем, как именно было записано в базе данных текстовое значение. Кроме того, бывают случаи, когда текстовое значение было написано с ошибками. Если понадобится найти все счета, выставленные в городах, название которых начинается с буквы *T*, в условии `WHERE` придется изменить параметры.

Оператор `LIKE` использует подстановочные знаки, представленные символом `%` (символ процента). То, что следует за знаком `=`, это единственное значение, которое вы увидите в своем наборе результатов. С помощью оператора `LIKE` и подстановочного знака вы можете определить варианты ввода.

ПРИМЕЧАНИЕ

Подстановочные знаки всегда заключаются в одинарные кавычки. Не заключенный в кавычки символ % – это просто арифметический оператор (показан ранее в этой главе в таблице операторов). Текстовый поиск не чувствителен к регистру. При использовании в запросе строчной или заглавной буквы результаты будут одинаковы.

С помощью подстановочного знака можно задать любое количество символов любого типа. Как показано в примере ниже, запрос ищет любые счета, выставленные в городах на букву *T*. В результат теперь попали Торонто и Тусон.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   BillingCity LIKE 'T%'
ORDER BY
   Total
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-07-24 00:00:00	796 Dundas Street West	Toronto	0.99
2	2012-03-11 00:00:00	1033 N Park Ave	Tucson	0.99
3	2011-01-15 00:00:00	1033 N Park Ave	Tucson	1.98
4	2011-01-15 00:00:00	796 Dundas Street West	Toronto	1.98
5	2013-06-01 00:00:00	796 Dundas Street West	Toronto	1.98
6	2013-09-02 00:00:00	1033 N Park Ave	Tucson	1.98
7	2011-04-19 00:00:00	1033 N Park Ave	Tucson	3.96
8	2013-09-03 00:00:00	796 Dundas Street West	Toronto	3.96
9	2011-07-22 00:00:00	1033 N Park Ave	Tucson	5.94
10	2013-12-06 00:00:00	796 Dundas Street West	Toronto	5.94
…	14 строк получено за 1 мс			

Рис. 50

Добавление еще одного символа `%` перед буквой *T* изменит условие поиска на поиск счета, выставленного в городе, название которого содержит букву *T*.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
```

```
WHERE
   BillingCity LIKE '%T%'
ORDER BY
   Total
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-01-19 00:00:00	Berger Straße 10	Frankfurt	0.99
2	2009-02-19 00:00:00	1600 Amphitheatre Parkway	Mountain View	0.99
3	2009-07-24 00:00:00	796 Dundas Street West	Toronto	0.99
4	2010-08-31 00:00:00	Celsiusg. 9	Stockholm	0.99
5	2010-10-01 00:00:00	230 Elgin Street	Ottawa	0.99
6	2011-01-02 00:00:00	2211 W Berry Street	Fort Worth	0.99
7	2011-09-07 00:00:00	Rua dos Campeões Europeus de Viena, 4350	Porto	0.99
...	126 строк получено за 2 мс			

Рис. 51

ПРИМЕЧАНИЕ

Разумеется, при этом в результатах появятся и города, названия которых начинаются или заканчиваются строчной буквой t. Символ % может представлять любую букву (буквы), включая строчную t.

Оператор `LIKE` также используется для исключения результатов, соответствующих указанным параметрам. Для этого необходимо поставить ключевое слово `NOT` перед оператором `LIKE`, и вы сможете исключить записи из результата запроса.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   BillingCity NOT LIKE '%T%'
ORDER BY
   Total
```

Из примеров видно, что существует множество способов использования оператора с подстановочными знаками. Рассмотрим наиболее распространенные.

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-03-22 00:00:00	110 Raeburn Pl	Edinburgh	0.99
2	2009-04-22 00:00:00	5112 48 Street	Yellowknife	0.99
3	2009-05-23 00:00:00	Praça Pio X, 119	Rio de Janeiro	0.99
4	2009-06-23 00:00:00	C/ San Bernardo 85	Madrid	0.99
5	2009-08-24 00:00:00	Grétrystraat 63	Brussels	0.99
6	2009-09-24 00:00:00	3 Chatham Street	Dublin	0.99
7	2009-10-25 00:00:00	319 N. Frances Street	Madison	0.99
8	2009-11-25 00:00:00	Ullevålsveien 14	Oslo	0.99
9	2009-12-26 00:00:00	9, Place Louis Barthou	Bordeaux	0.99
10	2010-01-26 00:00:00	801 W 4th Street	Reno	0.99
...	286 строк получено за 4 мс			

Рис. 52

Использование подстановочных знаков (где T – буква или фрагмент искомой строки)	**Результат** (не чувствительны к регистру)
'T%'	Находит все записи, начинающиеся с буквы T
'%T'	Находит все записи, заканчивающиеся буквой T
'%T%'	Находит все записи, содержащие в строке букву T
'T%T'	Находит все записи, начинающиеся и заканчивающиеся буквой T

Рис. 53

ПРИМЕЧАНИЕ

Символ % можно интерпретировать как «что угодно». Например, когда вы указываете '%T%', вы имеете в виду: «Меня не беспокоит, какие символы содержатся в строке до или после буквы T (буква T при этом не первая и не последняя)».

Фильтрация записей по дате

Используя все знания как о числах, так и о тексте, вы сможете теперь выполнить поиск счета, выставленного в определенную дату. Рассмотрим следующий пример:

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   InvoiceDate = '2009-01-03 00:00:00'
ORDER BY
   Total
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-01-03 00:00:00	Grétrystraat 63	Brussels	5.94
	1 строка получена за 1 мс			

Рис. 54

Обратите внимание на способ написания даты. При указании даты в запросе важно сначала проанализировать, каким образом дата хранится в запрашиваемой таблице. Как вы знаете из главы 1, для этого надо перейти на вкладку Browse Data (Просмотр данных), выбрать таблицу `invoices` и проанализировать формат, в котором столбец `InvoiceDate` хранит даты. В нашей базе данных даты хранятся в формате гггг-мм-дд 00:00:00. Затем перейдите на вкладку Database Structure (Структура базы данных) и проанализируйте поле `InvoiceDate` таблицы `invoices`. Вы видите, что в столбце `Type` даты имеют тип данных `DATETIME`.

В условии `WHERE` дата, как и текст, заключена в одинарные кавычки. При работе с датами используйте те же операторы, что и при работе с числами: `=`, `>`, `<`, `BETWEEN` и т. д.

Практические задания

- Получите все счета, выставленные в период с 1 января 2009 г. по 31 декабря 2009 г.
- Найдите 10 самых крупных счетов, полученных после 5 июля 2009 г.

Функция DATE()

При работе с датами в SQL можно использовать ряд функций, которые помогают получать более точные результаты. Из предыдущего примера видно, что столбец `InvoiceDate` таблицы `invoices` имеет тип данных `DATETIME`. Поэтому, когда мы указывали значение даты в услови `WHERE`, мы включали время (2009-01-03 00:00:00). Функция `DATE()`позволяет исключить время при указании параметров даты.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   Invoices
WHERE
   DATE(InvoiceDate) = '2009-01-03'
ORDER BY
   Total
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-01-03 00:00:00	Grétrystraat 63	Brussels	5.94
	1 строка получена за 2 мс			

Рис. 55

Результат этого запроса идентичен результату предыдущего запроса. Однако использование функции `DATE()` позволяет получить результат быстрее, когда параметры времени либо отсутствуют, либо не актуальны.

ПРИМЕЧАНИЕ

В SQL доступно множество функций. Функция DATE () особенно полезна при использовании условия WHERE для сортировки записей по дате. В главе 7 мы более подробно рассмотрим другие функции.

Использование операторов AND и OR с двумя отдельными полями

В этой главе мы применяли операторы только для выбора подмножества одного поля. Например, мы использовали оператор AND с оператором BETWEEN для фильтрации результатов поля Total с двумя различными числовыми значениями. Мы также можем применить операторы AND и OR для указания параметров нескольких полей. В приведенном ниже запросе оператор AND используется вместе с функцией DATE() для поиска всех счетов, оформленных после 02.01.2010, на общую сумму менее $3,00. Результат этого запроса должен удовлетворять одновременно обоим условиям: (DATE (InvoiceDate)>'2010-01-02' AND Total<3).

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   DATE(InvoiceDate) > '2010-01-02' AND Total < 3
ORDER BY
   Total
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2010-01-26 00:00:00	801 W 4th Street	Reno	0.99
2	2010-03-29 00:00:00	Barbarossastraße 19	Berlin	0.99
3	2010-04-29 00:00:00	1 Microsoft Way	Redmond	0.99
4	2010-05-30 00:00:00	421 Bourke Street	Sidney	0.99
5	2010-06-30 00:00:00	Rua da Assunção 53	Lisbon	0.99
6	2010-07-31 00:00:00	Qe 7 Bloco G	Brasília	0.99
7	2010-08-31 00:00:00	Celsiusg. 9	Stockholm	0.99
8	2010-10-01 00:00:00	230 Elgin Street	Ottawa	0.99
9	2010-11-01 00:00:00	Sønder Boulevard 51	Copenhage	0.99
10	2010-12-02 00:00:00	Via Degli Scipioni, 43	Rome	0.99
…	136 строк получено за 3 мс			

Рис. 56

Из полученных результатов можно увидеть, что возвращаются только счета, оформленные после 2 января 2010 года, общая сумма которых составляет менее $3.

ПРИМЕЧАНИЕ

Для поиска дополнительных параметров вы можете добавить дополнительные операторы AND. Как и в случае с оператором IN, можно не ограничиваться только двумя значениями.

Практические задания

✶ Найдите все счета, которые были выставлены в городе, название которого начинается с буквы *P*, а общая сумма превышает $2.

Оператор OR

Оператор OR позволяет найти записи, соответствующие *любому* из заданных вами условий. В следующем запросе выполняется поиск всех счетов, выставленных в городах, названия которых начинаются с буквы *P* или с буквы *D*.

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-09-24 00:00:00	3 Chatham Street	Dublin	0.99
2	2011-02-02 00:00:00	Klanova 9/506	Prague	0.99
3	2011-03-05 00:00:00	68, Rue Jouvence	Dijon	0.99
4	2011-09-07 00:00:00	Rua dos Campeões Europeus de Viena, 4350	Porto	0.99
5	2012-04-11 00:00:00	Rilská 3174/6	Prague	0.99
6	2012-08-13 00:00:00	8, Rue Hanovre	Paris	0.99
7	2009-02-01 00:00:00	8, Rue Hanovre	Paris	1.98
8	2009-12-08 00:00:00	Klanova 9/506	Prague	1.98
9	2010-01-08 00:00:00	68, Rue Jouvence	Dijon	1.98
10	2010-04-11 00:00:00	4, Rue Milton	Paris	1.98
…	56 строк получено за 4 мс			

Рис. 57

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   BillingCity LIKE 'p%' OR BillingCity LIKE 'd%'
ORDER BY
   Total
```

Использование круглых скобок с операторами AND и OR для указания порядка операций

При написании в SQL более длинных условий `WHERE`, включающих несколько логических операторов, можно определить порядок выполнения операций аналогично правилам базовой арифметики. Возможно, вы встречали аббревиатуру PEMDAS (Parentheses, Exponents, Multiplication, Division, Addition, Subtraction — круглые скобки, экспоненты, умножение, деление, сложение, вычитание) или похожую BEMDAS (Brackets, Exponents, Multiplication, Division, Addition, Subtraction). Первая используется в США, вторая — в странах, где сильно влияние Великобритании. Если не встречали, то не стоит беспокоиться. Способ определения порядка операций очень простой. Но пока рассмотрим работу операторов `AND` и `OR`. Допустим, нам необходимо получить все счета на сумму свыше $1,98 из любых городов, названия которых начинаются с буквы *P* или *D*. Запрос будет выглядеть следующим образом:

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   Total > 1.98 AND BillingCity LIKE 'p%' OR
   BillingCity LIKE 'd%'
ORDER BY
   Total
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2009-09-24 00:00:00	3 Chatham Street	Dublin	0.99
2	2011-03-05 00:00:00	68, Rue Jouvence	Dijon	0.99
3	2010-01-08 00:00:00	68, Rue Jouvence	Dijon	1.98
4	2010-06-12 00:00:00	12, Community Centre	Delhi	1.98
5	2011-03-18 00:00:00	3 Chatham Street	Dublin	1.98
6	2012-08-26 00:00:00	68, Rue Jouvence	Dijon	1.98
7	2012-10-27 00:00:00	12, Community Centre	Delhi	1.98
8	2013-08-02 00:00:00	3 Chatham Street	Dublin	1.98
9	2011-06-06 00:00:00	4, Rue Milton	Paris	1.99
10	2010-12-02 00:00:00	12, Community Centre	Delhi	1.99
...	43 строки получено за 2 мс			

Рис. 58

Когда данный запрос выполняется, браузер SQL сначала объединяет два условия по обе стороны от оператора `AND`, возвращая счета, общая сумма которых превышает $1,98, *и* названия городов, где они были выставлены, начинаются с буквы *P*. Затем отдельно обрабатывается условие справа от оператора `OR`, как если бы оператора `AND` не существовало. Другими словами, запрос сначала выполняет поиск результатов, где `Total>1.98 AND BillingCity LIKE 'p%'`. Затем выполняется поиск всех результатов, где `BillingCity LIKE 'd%'`, а затем возвращаются результаты для обоих условий в порядке возрастания в поле `Total`.

Если вышеуказанный запрос вы ввели в свой браузер SQL, то заметите, что получены данные, соответствующие значению менее $1,98, но только для городов, начинающихся с буквы *D*.

Это связано с тем, что порядок операций SQL определен следующим образом: сначала обрабатывается оператор `AND`, а затем — оператор `OR`. Это не совсем так, как мы изначально хотели, но есть простой способ обрабатывать наш запрос именно так, как мы планировали, без необходимости прибегать к логическим операторам.

ПРИМЕЧАНИЕ

В SQL порядок оператора AND соответствует порядку операции умножения, а оператора OR – операции сложения, если не добавлены круглые скобки. При отсутствии круглых скобок оператор AND будет обрабатываться по принципу, аналогичному арифметике: 3*2+1 равно 7, а 3*(2+1) равно 9.

При добавлении круглых скобок, как показано в примере ниже, браузер SQL сначала выполняет поиск данных, удовлетворяющих условиям внутри скобок: `(BillingCity LIKE 'p%' OR BillingCity LIKE 'd%')`. Затем только из этих записей, он ищет данные, общая сумма которых превышает $1,98 `(Total>1.98)`.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
   Total > 1.98 AND (BillingCity LIKE 'p%' OR
   BillingCity LIKE 'd%')
ORDER BY
   Total
```

	InvoiceDate	BillingAddress	BillingCity	Total
1	2011-06-06 00:00:00	4, Rue Milton	Paris	1.99
2	2013-12-22 00:00:00	12, Community Centre	Dehli	1.99
3	2011-06-19 00:00:00	8, Rue Hanovre	Paris	2.98
4	2010-03-12 00:00:00	Klanova 9/506	Prague	3.96
5	2010-04-12 00:00:00	68, Rue Jouvence	Dijon	3.96
6	2010-07-14 00:00:00	4, Rue Milton	Paris	3.96
7	2010-10-15 00:00:00	Rua dos Campeões Europeus de Viena, 4350	Porto	3.96
8	2011-05-20 00:00:00	Rilská 3174/6	Prague	3.96
9	2011-09-21 00:00:00	8, Rue Hanovre	Paris	3.96
10	2013-01-29 00:00:00	12, Community Centre	Delhi	3.96
...	35 строк получено за 1 мс			

Рис. 59

Теперь мы переписали запрос так, что все счета, сумма которых превышает $1,98, будут сформированы в городах, начинающихся либо с буквы *P*, либо с буквы *D*, что и требовалось изначально.

НА МОЙ ВЗГЛЯД,

хотя важно и полезно знать, как SQL обрабатывает логические операторы, целесообразно при использовании нескольких операторов просто добавлять круглые скобки, что упрощает понимание сложной логики кода. Если вам хочется потренироваться, выполните следующие упражнения как со скобками, так и без них, и вы увидите, как это влияет на результат.

Практические задания

- Снова запустите запрос и проанализируйте, имеются ли в столбце `Total` какие-либо данные, значения которых меньше $1,98.
- Найдите все счета с общей суммой выше $3,00, выставленные в городах, название которых начинается с буквы *P* или *D*.

Оператор CASE

Оператор `CASE` позволяет создать новое временное поле в базе данных, которое станет меткой для данных на основе заданных пользователем условий. Чтобы лучше понять смысл оператора `CASE`, рассмотрим следующий сценарий.

Сценарий

Цель отдела продаж компании sTunes — чтобы как можно больше клиентов потратили от $7 до $15 на покупку музыкальной продукции в онлайн-магазине. Для этой цели были созданы следующие категории покупок: *Baseline Purchase* (базовая покупка), *Low Purchase* (незначительная покупка), *Target Purchase* (целевая покупка) и *Top Performer* (значительная покупка).

Поскольку стоимость песни составляет от $0,99 до $1,99, любой счет из этого диапазона считается *Baseline Purchase* (базовой покупкой). Сумма счетов от $2,00 до $6,99 относится к *Low Purchase* (незначительной покупке). Поскольку основная цель продаж составляет от $7 до $15, любые покупки в этой категории — это *Target Purchase* (целевая покупка), а выше $15— *Top Performer* (значительная покупка).

Отдел продаж sTunes хочет узнать, можно ли получить из базы данных какую-либо информацию о продажах для всех перечисленных категорий.

Для создания нового поля PurchaseType в таблице invoices мы можем использовать оператор CASE. Поле PurchaseType будет отображаться вместе с другими уже существующими полями в нашем запросе, как если бы это было просто еще одно поле в базе данных.

Использование в запросе оператора CASE

Сначала создадим простой запрос SELECT, как в главе 4.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
ORDER BY
   BillingCity
```

Теперь нам необходимо отсортировать наши результаты в зависимости от названия города, где выставлен счет, чтобы мы могли увидеть целевые покупки в зависимости от региона.

Чтобы добавить оператор CASE к данному запросу, необходимо добавить его в нижнюю часть блока SELECT после всех существующих полей. Начнем с ключевого слова CASE, за которым вставим ключевое слово END. Между этими двумя ключевыми словами нам необходимо определить условия. Каждая проверка начинается с ключевого слова WHEN, за которым следует условие. Наше первое условие — это *Baseline Purchase* (базовая покупка), то есть любой счет, сумма которого менее $2,00, другими словами "TOTAL<2.00". После логического условия в случае его выполнения необходимо указать ожидаемый результат. Для этой цели используем оператор THEN. Наш ожидаемый результат — покупки менее $2,00 с меткой *Baseline Purchase* (базовая покупка), что и предусмотрено для нашего сценария.

Эту же последовательность можно повторить для любого количества условий. Поэтому этот метод мы повторим для остальных категорий покупок. Ключевое слово ELSE всегда ставят за последним явным перечисленным условием. Любые записи, которые еще не определены, будут отнесены к категории, указанной в условии ELSE.

ВНИМАНИЕ

Ключевое слово ELSE указывать не обязательно, но рекомендуется. В данных могут быть значения, выходящие за рамки требуемых условий. Условие ELSE фиксирует эти значения, и вы можете понять, что с ними делать. Если условие ELSE не добавлено, любые результаты в вашем наборе данных, выходящие за рамки требуемых условий, будут возвращены как значение NULL.

Последнее, что мы делаем, — создаем псевдоним, то есть новое поле в нашей базе данных. Этот псевдоним будет расположен после оператора `END`. Создадим новое поле с именем `PurchaseType`.

НАПОМИНАНИЕ

Мы создаем псевдонимы с помощью ключевого слова AS. Таким образом, оператор CASE завершается словом END AS, а затем псевдонимом, который мы выбрали для нового поля.

Запрос будет выглядеть следующим образом (рис. 60):

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total,
   CASE
   WHEN TOTAL < 2.00 THEN 'Baseline Purchase'
   WHEN TOTAL BETWEEN 2.00 AND 6.99 THEN 'Low
   Purchase'
   WHEN TOTAL BETWEEN 7.00 AND 15.00 THEN 'Target
   Purchase'
   ELSE 'Top Performers'
   END AS PurchaseType
FROM
   invoices
ORDER BY
   BillingCity
```

Проанализировав полученные результаты, мы увидим, что добавлена новая категория с именем `PurchaseType` и к данным добавлены все категории покупок.

Из полученных результатов мы видим, что все категории, которые мы проверяли, представлены в соответствии с их соответствующими категориями покупки (`PurchaseType`).

	InvoiceDate	BillingAddress	BillingCity	Total	PurchaseType
1	2009-05-10 00:00:00	Lijnbaansgracht 120bg	Amsterdam	8.91	Target Purchase
2	2010-12-15 00:00:00	Lijnbaansgracht 120bg	Amsterdam	1.91	Baseline Purchase
3	2011-03-19 00:00:00	Lijnbaansgracht 120bg	Amsterdam	3.96	Low Purchase
4	2011-06-21 00:00:00	Lijnbaansgracht 120bg	Amsterdam	8.94	Target Purchase
5	2012-02-09 00:00:00	Lijnbaansgracht 120bg	Amsterdam	0.99	Baseline Purchase
6	2013-08-02 00:00:00	Lijnbaansgracht 120bg	Amsterdam	1.98	Baseline Purchase
7	2013-09-12 00:00:00	Lijnbaansgracht 120bg	Amsterdam	13.86	Target Purchase
8	2009-04-05 00:00:00	3,Raj Bhavan Road	Bangalore	3.96	Low Purchase
9	2009-07-08 00:00:00	3,Raj Bhavan Road	Bangalore	5.94	Low Purchase
10	2010-02-26 00:00:00	3,Raj Bhavan Road	Bangalore	1.99	Baseline Purchase
…	412 строк получено за 17 мс				

Рис. 60

ПРИМЕЧАНИЕ

Используя условие ORDER BY, мы можем упорядочить результаты по полю, где отображается каждая категория покупки в алфавитном порядке, начиная с Baseline Purchase (базовая покупка) и заканчивая Top Performer (значительная покупка). Для ясности мы присвоили каждой категории свои имена, но вы можете присвоить им любое другое имя.

Теперь, когда новые категории созданы, нам необходимо узнать больше о демографических характеристиках наших клиентов на основе наших новых категорий продаж. Существует множество способов структурировать остальную часть оператора `SELECT`.

Ответим на следующие вопросы.

- В каких городах осуществляются самые эффективные продажи?
- Самые эффективные продажи в основном осуществляются в США или в других странах?
- В каких городах совершается больше всего базовых покупок?

Рассмотрим первый вопрос. Чтобы получить данные только об эффективных продажах и упорядочить их по городам, мы можем изменить наш существующий запрос, используя условие `WHERE`.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total,
   CASE
   WHEN TOTAL < 2.00 THEN 'Baseline Purchase'
   WHEN TOTAL BETWEEN 2.00 AND 6.99 THEN 'Low
   Purchase'
   WHEN TOTAL BETWEEN 7.00 AND 15.00 THEN 'Target
   Purchase'
   ELSE 'Top Performers'
   END AS PurchaseType
FROM
   invoices
   WHERE PurchaseType = 'Top Performers'
ORDER BY
   BillingCity
```

	InvoiceDate	BillingAddress	BillingCity	Total	PurchaseType
1	2010-02-18 00:00:00	Erzsébet krt. 58.	Budapest	21.86	Top Performers
2	2010-03-21 00:00:00	162 E Superior Street	Chicago	15.86	Top Performers
3	2012-10-06 00:00:00	68, Rue Jouvence	Dijon	16.86	Top Performers
4	2011-04-28 00:00:00	3 Chatham Street	Dublin	21.86	Top Performers
5	2012-08-05 00:00:00	2211 W Berry Street	Fort Worth	23.86	Top Performers
6	2011-05-29 00:00:00	319 N. Frances Street	Madison	18.86	Top Performers
7	2011-06-29 00:00:00	Ullevålsveien 14	Oslo	15.86	Top Performers
8	2012-09-05 00:00:00	Klanova 9/506	Prague	16.9	Top Performers
9	2013-11-13 00:00:00	Rilská 3174/6	Prague	25.9	Top Performers
10	2010-01-13 00:00:00	Calle Lira, 198	Santiago	17.9	Top Performers
...	11 строк получено за 6 мс				

Рис. 61

Проанализировав результат запроса, мы можем определить, что самые эффективные продажи в основном осуществляются в США.

Комбинации полей, по которым проводится поиск, практически безграничны. Например, мы можем получить данные в зависимости от даты выставления счета, чтобы проанализировать сезонные продажи. Использование операторов

CASE с условием WHERE и операторами, изученными в этой главе, поможет получить данные, необходимые для нашего отдела продаж.

ПРИМЕЧАНИЕ

В примерах в этой главе мы использовали оператор CASE в части запроса SELECT после необходимых для отображения полей. Далее вы можете встретить запрос, в котором оператор CASE содержится в условии WHERE (редкий случай). Все, что сейчас важно, – это помнить, что оператор CASE должен быть указан в условии SELECT, но ссылаться на него можно из другого места программного кода.

Контрольные вопросы

1. Создайте запрос для таблицы `invoices`, включающий оператор `CASE`, который будет отмечать все продажи из США — страны, откуда выставлен счет — как Domestic Sales (Продажи на внутреннем рынке), а все другие продажи — как Foreign Sales (Продажи за рубежом). После оператора `END AS` создайте новое поле `SalesType`.
2. Отсортируйте эти данные по новому полю `SalesType`.
3. Сколько счетов от продаж на внутреннем рынке превышает сумму $15?

Резюме

- Операторы — это специальные ключевые слова SQL, которые используются с условиями SQL для фильтрации данных в зависимости от определенных условий.
- Использование условия `WHERE` с комбинацией различных операторов позволяет выполнять поиск определенного текста, даты и числа.
- Функция `DATE()` позволяет исключить время при указании параметров даты.
- Порядок операций при использовании логических операторов (таких как `AND/OR`) устанавливается с помощью круглых скобок `()`.
- Оператор `CASE` позволяет отмечать записи специальным именем поля в зависимости от заданных пользователем логических условий.

ГЛАВА 6

Работа с несколькими таблицами

Краткое содержание

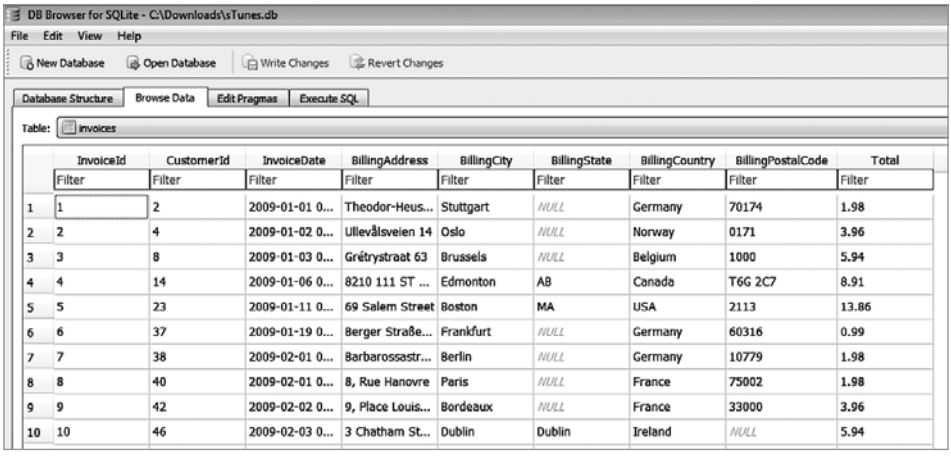
- ✓ Соединение таблиц базы данных
- ✓ Соединения и структура реляционной базы данных
- ✓ Псевдонимы соединяемых таблиц
- ✓ Внутреннее соединение, левое внешнее соединение и правое внешнее соединение
- ✓ Соединение более двух таблиц
- ✓ Контрольные вопросы

Во всех предыдущих запросах мы рассматривали получение данных только из одной таблицы. Хотя мы изучили некоторые мощные запросы, они не используют все возможности реляционной базы данных. База данных sTunes содержит тринадцать таблиц. Каждая таблица содержит некоторую, но не всю, информацию о компании. Чтобы ответить на более сложные вопросы о компании sTunes, понадобится одновременный доступ к данным из нескольких таблиц. В этой главе мы узнаем, как получить данные из двух или более таблиц с помощью одного запроса и инструментов, называемых соединениями.

Что такое соединение

Соединение (join) — это операция, которая объединяет поля двух или более таблиц реляционной базы данных. Рассмотрим очень простой пример использования таблицы `invoices` в базе данных sTunes. В предыдущих главах мы много работали с таблицей `invoices`, поэтому она хорошо нам знакома. На вкладке Browse Data (Просмотр данных) браузера SQL (рис. 62) видно, что таблица `invoices` состоит из

девяти полей. Поле `InvoiceId` содержит идентификационный номер каждого счета. Поле `CustomerId` — идентификационный номер каждого клиента (которому выставлен счет). Таблица `invoices` также содержит поля с информацией о дате и сумме счета. Остальные поля в этой таблице предназначены для адреса плательщика.



	InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	2	2009-01-01 0...	Theodor-Heus...	Stuttgart	NULL	Germany	70174	1.98
2	2	4	2009-01-02 0...	Ullevålsveien 14	Oslo	NULL	Norway	0171	3.96
3	3	8	2009-01-03 0...	Grétrystraat 63	Brussels	NULL	Belgium	1000	5.94
4	4	14	2009-01-06 0...	8210 111 ST ...	Edmonton	AB	Canada	T6G 2C7	8.91
5	5	23	2009-01-11 0...	69 Salem Street	Boston	MA	USA	2113	13.86
6	6	37	2009-01-19 0...	Berger Straße...	Frankfurt	NULL	Germany	60316	0.99
7	7	38	2009-02-01 0...	Barbarossastr...	Berlin	NULL	Germany	10779	1.98
8	8	40	2009-02-01 0...	8, Rue Hanovre	Paris	NULL	France	75002	1.98
9	9	42	2009-02-02 0...	9, Place Louis...	Bordeaux	NULL	France	33000	3.96
10	10	46	2009-02-03 0...	3 Chatham St...	Dublin	Dublin	Ireland	NULL	5.94

Рис. 62

Допустим, отделу маркетинга компании sTunes необходимо более детально проанализировать клиентскую базу. В этом случае он запросит полный список имен клиентов (имя и фамилию) со всеми счетами, выставленными каждому клиенту. Как мы напишем такой запрос, используя только таблицу `invoices`? Нам не удастся ответить на этот вопрос, используя только один запрос и ранее полученные навыки. Таблица `invoices` не содержит имена клиентов. Однако таблица `invoices` содержит поле `CustomerId`. Чтобы отобразить список счетов с именами клиентов, которым выставлены данные счета, нам необходимы таблицы `invoices` и `customers`.

Мы видим, что таблица `customers` (рис. 63) содержит требуемую нам информацию — имена и фамилии всех клиентов компании sTunes. Данная таблица также содержит поле `CustomerId`. Если мы внимательно посмотрим на значки в таблице `customers`, то увидим, что рядом с полем `CustomerId` расположен небольшой значок ключа. В главе 1 мы говорили о том, что значок ключа — это символ первичного ключа, уникального поля идентификации для конкретной таблицы.

НАПОМИНАНИЕ

Каждая таблица должна содержать хотя бы одно поле, служащее первичным ключом. Первичный ключ одной таблицы, как правило, является внешним ключом для другой таблицы.

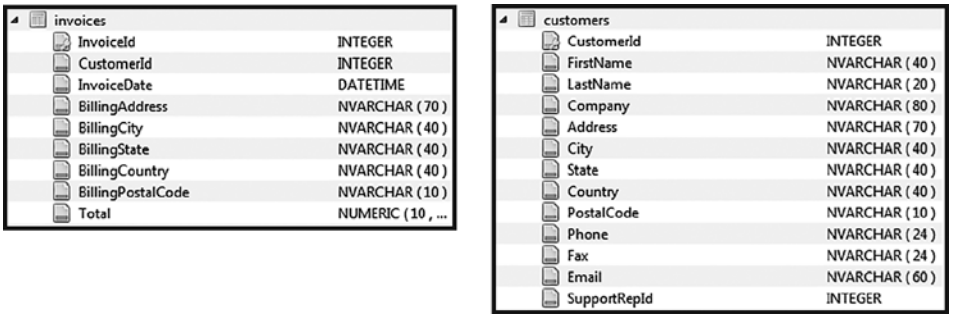


Рис. 63

Поскольку `CustomerId` — это первичный ключ таблицы `customers`, а таблица `invoices` содержит аналогичное поле с тем же именем, то эти два поля `CustomerId` — это связь, необходимая для одновременного доступа к обеим таблицам. Теперь у нас есть вся информация, необходимая для объединения этих двух таблиц и создания списка счетов с именами клиентов.

Рассмотрим пример использования операции соединения `JOIN`, объединяющей эти таблицы.

```
SELECT
   *
FROM
   invoices
INNER JOIN
   customers
ON
   invoices.CustomerId = customers.CustomerId
```

ПРИМЕЧАНИЕ

В данном примере мы используем соединение, называемое INNER JOIN (внутреннее соединение). Далее мы рассмотрим несколько различных типов соединений. Каждое работает по своему правилу. На данный момент вам лишь необходимо знать, что соединения позволяют нам получать доступ к полям из разных таблиц.

Вам уже знакомы многие операторы данного запроса. Запрос начинается с `SELECT`, как и все наши предыдущие запросы. Мы используем символ `*`, чтобы получить все поля в таблице. Выберем все поля из таблицы `invoices` и объединим их со всеми полями в таблице `customers`. Ключевое слово `ON` используется

в запросе для связи двух таблиц через поле `CustomerId`. Поскольку существует два варианта поля `CustomerId` (по одному в каждой таблице), нам нужно сообщить браузеру SQL, какой из вариантов использовать. Для этого существует специальная нотация (в форме `tablename.FieldName`). Устанавливаем равенство между полем `CustomerId` из таблицы `invoices` (записанное как `invoices.CustomerId`) и полем `CustomerId` из таблицы `customers` (`customers.CustomerId`). Результат показан на рис. 64.

ТАБЛИЦА INVOICES (3 из 9 полей) **ТАБЛИЦА CUSTOMERS** (4 из 13 полей)

	InvoiceId	CustomerId	...	Total	CustomerId	FirstName	LastName	...	SupportRepId
1	98	1	...	3.98	1	Luís	Gonçalves	...	3
2	121	1	...	3.96	1	Luís	Gonçalves	...	3
3	143	1	...	5.94	1	Luís	Gonçalves	...	3
4	195	1	...	.99	1	Luís	Gonçalves	...	3
5	316	1	...	1.98	1	Luís	Gonçalves	...	3
6	327	1	...	13.86	1	Luís	Gonçalves	...	3
7	382	1	...	8.91	1	Luís	Gonçalves	...	3
8	1	2	...	1.98	2	Leonie	Köhler	...	5
9	12	2	...	13.86	2	Leonie	Köhler	...	5
10	67	2	...	8.91	2	Leonie	Köhler	...	5
...	412 строк получено за 17 мс				...				

Рис. 64

ПРИМЕЧАНИЕ

Используя символ *, мы объединили девять полей таблицы invoices с тринадцатью полями таблицы customers. В результате получилось двадцать два поля. Для удобства печати мы убрали некоторые из этих полей, но мы можем просмотреть все двадцать два поля в браузере, используя горизонтальную полосу прокрутки на панели результатов на вкладке Execute SQL (Выполнить SQL-запрос).

Соединения и структура реляционной базы данных

Теперь проанализируем, что произойдет, если объединить таблицу `invoices` с таблицей `customers`. Рассмотрим поле `InvoiceId` из таблицы `invoices` результирующего набора данных (см. рис. 64). Мы увидим, что первые семь записей связаны с одним и тем же `CustomerId`. Это означает, что клиенту № 1 выставили все семь счетов. Если мы проанализируем часть результирующей таблицы `customers`, то увидим, что данного клиента зовут Luís Gonçalves. Один клиент связан со многими счетами. На языке реляционной базы данных (глава 1) мы можем сказать, что таблица `customers` имеет связь «один-ко-многим» с таблицей `invoices`. Клиенту с одним `CustomerId` могут выставить множество счетов (если он заказал несколько песен), но в таблице `invoices` сохранится один и тот же номер `CustomerId`. Еще один вариант описания этой связи — схема базы данных в виде ER-диаграммы (ER от Entity — Relationship, сущность — связь).

На рис. 65 графически представлены взаимосвязи между таблицей `customers` и таблицей `invoices`. В остальной части нашей базы данных мы обнаружим множество других первичных и внешних ключей, устанавливающих связь между разными таблицами. Понимание этой взаимосвязи — это и есть часть создания и использования соединений. Чтобы объединить таблицы вместе, мы должны уметь идентифицировать первичные и внешние ключи и понимать, какие поля нам необходимы.

Если бы таблица `invoices` содержала поле, включающее все имена клиентов, то в соединении таблиц не было бы необходимости. Также вместо базы данных с тринадцатью таблицами можно было бы создать одну гигантскую таблицу, содержащую все поля.

Вопрос. Зачем в базах данных нужно иметь несколько таблиц?

Ответ. *Нормализация* — это процесс организации данных в реляционной базе данных. Нормализация включает создание таблиц и установку отношений между таблицами. Нормализация позволяет уменьшить размеры баз данных, так как не надо хранить повторяющиеся поля в одной таблице. По мере увеличения размера базы данных возрастает потребность в ее нормализации. Это целесообразно, даже если экономится всего нескольких секунд при обработке запроса. Учитывая гигантский размер некоторых баз данных, каждая секунда имеет значение. Представьте, что поиск в Google занимает пять минут, а не несколько секунд.

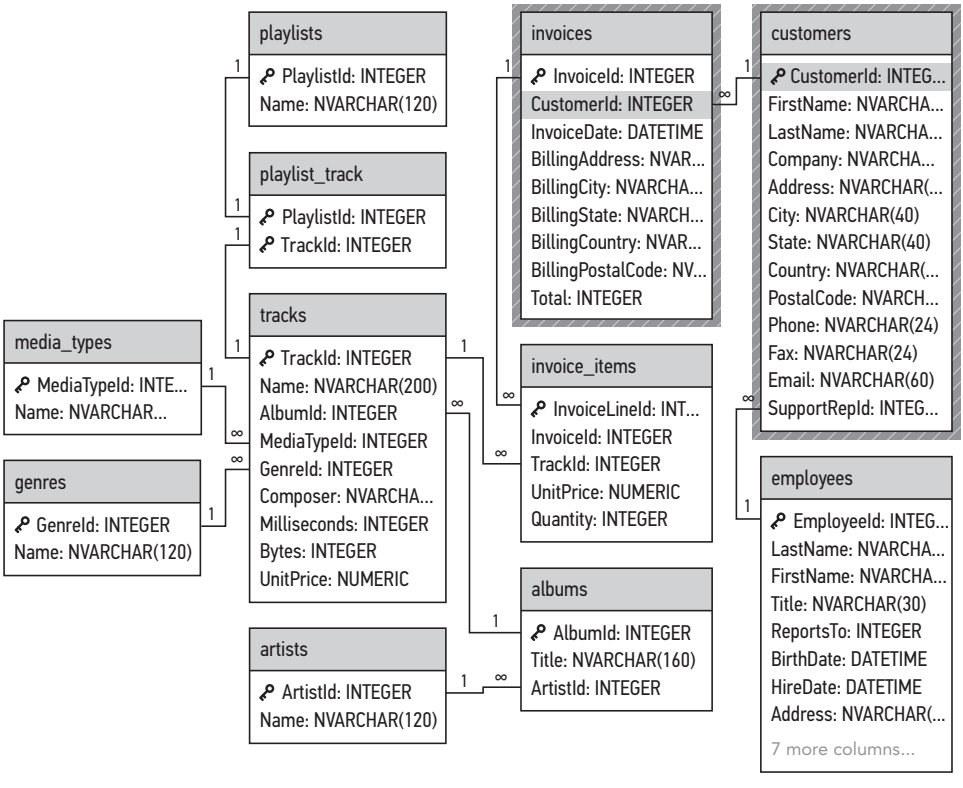


Рис. 65

Теперь, когда мы определили общую связь между двумя полями в таблице `invoices` и в таблице `customers`, стоит более внимательно разобрать, как писать запросы с соединениями.

Псевдонимы соединяемых таблиц

Из первого примера видно, что при обращении к полям соединения `JOIN` имеют особый синтаксис. Поскольку две таблицы в любой базе данных могут содержать поля с одинаковыми именами, при создании соединений необходимо для указания определенного поля задать имя таблицы, чтобы браузер SQL точно знал, какое поле имеется в виду. По правилам синтаксиса необходимо сначала указать имя таблицы, а затем через точку — имя поля. Чтобы уменьшить объем текста и повысить удобочитаемость, в соединениях часто используются псевдонимы. Следующие два соединения идентичны по функциям.

```
SELECT
   *
FROM
   invoices
INNER JOIN
   customers
ON
   invoices.CustomerId = customers.CustomerId

-------------------------------------------

SELECT
   *
FROM
   invoices AS i
INNER JOIN
   customers AS c
ON
   i.CustomerId = c.CustomerId
```

ПРИМЕЧАНИЕ

Псевдонимы для соединений должны быть краткими и удобочитаемыми. Они, как правило, состоят из одной буквы, причем используется первая буква соответствующей таблицы (tablename.FieldName будет записано как t.FieldName). Далее в этой главе для имен таблиц мы будем использовать однобуквенные псевдонимы.

Чтобы продемонстрировать необходимость использования псевдонимов при работе с соединениями, давайте рассмотрим наш первоначальный сценарий. Руководству sTunes необходим список с указанием имен клиентов и счетов, выставленных каждому клиенту. Когда в этой главе мы писали наше первое соединение, для выбора всех полей из каждой таблицы был использован символ *. Результатом данного запроса стал массивный набор результатов из двадцати двух полей. Однако нам требуются только имена клиентов и информация о счете. Также, когда мы используем символ *, мы не можем контролировать порядок отображения полей. Предположим, что руководству sTunes необходимо в списке клиентов сначала отображать фамилию. Чтобы определить порядок вывода, следует в операторе `SELECT` вместо символа * указать имена полей. Давайте создадим аналогичное приведенному выше соединение, но на этот раз в запросе укажем, что из таблицы `customers` необходимо отобразить поля `LastName` и `FirstName`, а из таблицы `invoices` — поля `InvoiceId`, `CustomerId`, `InvoiceDate` и `Total`. Поскольку мы работаем с двумя таблицами, содержащими отдельные поля с одинаковыми именами, в условии `SELECT` следует использовать обозна-

чение `tablename.FieldName`, как и в условии `ON` наших соединений, но с одним изменением: вместо полного имени таблицы мы зададим псевдоним, состоящий из первой буквы таблицы, за которой следует точка, а затем имя поля. Наконец, необходимо отсортировать результаты по фамилии клиента. В результате запрос будет выглядеть следующим образом:

```
SELECT
   c.LastName,
   c.FirstName,
   i.InvoiceId,
   i.CustomerId,
   i.InvoiceDate,
   i.Total
FROM
   invoices AS i
INNER JOIN
   customers AS c
ON
   i.CustomerId = c.CustomerId
ORDER BY
   c.LastName
```

	LastName	FirstName	InvoiceId	CustomerId	InvoiceDate	Total
1	Almeida	Roberto	34	12	2009-05-23 00:00:00	0.99
2	Almeida	Roberto	155	12	2010-11-14 00:00:00	1.98
3	Almeida	Roberto	166	12	2010-12-25 00:00:00	13.86
4	Almeida	Roberto	221	12	2011-08-25 00:00:00	8.91
5	Almeida	Roberto	350	12	2013-03-31 00:00:00	1.98
6	Almeida	Roberto	373	12	2013-07-03 00:00:00	3.96
7	Almeida	Roberto	395	12	2013-10-05 00:00:00	5.94
8	Barnett	Julia	71	28	2009-11-07 00:00:00	1.98
9	Barnett	Julia	82	28	2009-12-18 00:00:00	13.86
10	Barnett	Julia	137	28	2010-08-18 00:00:00	8.91
...	412 строк получено за 14 мс					

Рис. 66

ПРИМЕЧАНИЕ

Кажется странным, что мы ссылаемся на псевдонимы (в условии SELECT), прежде чем определим их (в условиях FROM и INNER JOIN). Но следует помнить, что браузер SQL обрабатывает запросы не в той очередности, как их прочел бы человек.

Анализируя первые десять результатов данного запроса, мы видим, что перечисление определенных полей в заданном порядке намного проще, чем вывод всех полей с использованием символа *. Мы также можем представить, насколько сложнее было бы соединение, если бы нам пришлось вводить имя таблицы каждый раз, когда мы ссылаемся на имя поля.

ПРИМЕЧАНИЕ

В большинстве случаев в условии SELECT рекомендуется указывать отдельные имена полей и избегать использования символа *. В этой главе мы будем использовать символ * только в демонстрационных целях для объяснения структуры JOIN.

Типы соединений

Существует несколько различных типов соединений. До сих пор мы использовали соединения, чтобы предоставить доступ к полям из нескольких таблиц. Мы определили первичный ключ таблицы `customers`, определили аналогичный внешний ключ в таблице `invoices` и использовали ключевое слово `ON`, чтобы связать две таблицы вместе.

Вопрос. Что произойдет, если данные из таблиц, которые мы объединяем, не полностью совпадают?

Например, что произойдет, если клиент — назовем его Customer 6 — удалил свою учетную запись в sTunes и впоследствии был удален из таблицы `customers`? Поскольку компания sTunes должна вести финансовую отчетность, в таблице `invoices` все еще содержится информация о том, что Customer 6 в какой-то момент совершил покупку. Нет ничего странного и необычного в обнаружении несовпадений в базах данных, и нам необходимо решить, хотим ли мы, чтобы наши запросы содержали несовпадающие данные или полностью их исключить. Для обработки несовпадений между таблицами используются разные типы соединений. Чтобы понять это, рассмотрим упрощенные версии наших таблиц `invoices` и `customers`.

ПРИМЕЧАНИЕ

Следующие таблицы незначительно отличаются от таблиц в нашей базе данных sTunes. Мы сохранили базовую структуру обеих таблиц, как в `invoices` и `customers`. Но мы удалили некоторые поля, сократили число записей в каждой таблице до пяти, упростили имена записей и добавили в каждую таблицу несколько записей, которые отличаются от записей в другой таблице.

УПРОЩЕННАЯ ТАБЛИЦА INVOICES

InvoiceId	CustomerId	InvoiceDate	BillingAddress	Total
1	2	1/1/2018	Billing Address 2	$1.00
2	2	2/1/2018	Billing Address 2	$2.00
3	3	3/1/2018	Billing Address 3	$3.00
4	4	4/1/2018	Billing Address 4	$4.00
5	6	5/1/2017	Billing Address 6	$5.00

УПРОЩЕННАЯ ТАБЛИЦА CUSTOMERS

CustomerId	Name	Address
1	Customer 1	Address 1
2	Customer 2	Address 2
3	Customer 3	Address 3
4	Customer 4	Address 4
5	Customer 5	Address 6

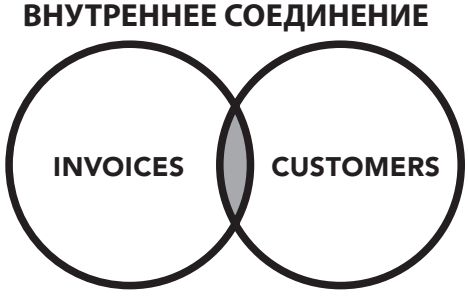
Рис. 67

Анализируя упрощенный вариант таблиц `invoices` и `customers`, мы можем выявить несколько несовпадений. Прежде всего, из таблицы `invoices` видно, что клиент с идентификатором Customer 6 совершил покупку 01.05.2017, но этот клиент в таблице `customers` не отображается. Кроме того, кажется, что Customer 1 и Customer 5 вообще никогда ничего не покупали, поскольку данные идентификаторы отсутствуют в таблице `invoices`. Customer 2 появляется дважды, поэтому мы можем сделать вывод, что этот клиент совершил две покупки. Поскольку записи для Customer 1 и Customer 5 существуют в таблице `customers`, но отсутствуют в таблице `invoices`, а Customer 6 существует только в таблице `invoices`, то каждая таблица содержит по крайней мере одну уникальную запись, которой нет в другой таблице. Теперь мы можем попытаться объединить эти две таблицы и проанализировать, как обрабатывается результат в зависимости от используемого типа соединения. Рассмотрим соединение `INNER JOIN`.

Внутреннее соединение (INNER JOIN)

При использовании внутреннего соединения возвращаются только совпадающие записи. Любые несовпадающие данные из любой таблицы игнорируются. Соединения часто описываются диаграммами Венна (рис. 68). Внутреннее соединение представляет собой только перекрывающуюся часть диаграммы Венна.

В данном примере соединение `INNER JOIN` игнорирует Invoice 5 из таблицы `invoices`, так как этот счет относится к клиенту Customer 6, которого нет в таблице `customers`. Точно так же клиентам Customer 1 и Customer 5 (из таблицы `customers`) не выставляли никаких счетов, поэтому эта запись также игнорируется. Как показано на диаграмме Венна, включены только перекрывающиеся данные. На рис. 69 наглядно продемонстрировано создание внутреннего соединения двух таблиц с разными данными.




```
SELECT *
FROM invoices AS i
INNER JOIN customers AS c
ON i.CustomerId = c.CustomerId
```

Рис. 68

УПРОЩЕННАЯ ТАБЛИЦА INVOICES

InvoiceId	CustomerId	InvoiceDate	BillingAddress	Total
1	2	1/1/2018	Billing Address 2	$1.00
2	2	2/1/2018	Billing Address 2	$2.00
3	3	3/1/2018	Billing Address 3	$3.00
4	4	4/1/2018	Billing Address 4	$4.00
5	6	5/1/2017	Billing Address 6	$5.00

ВНУТРЕННЕЕ СОЕДИНЕНИЕ

УПРОЩЕННАЯ ТАБЛИЦА CUSTOMERS

CustomerId	Name	Address
1	Customer 1	Address 1
2	Customer 2	Address 2
3	Customer 3	Address 3
4	Customer 4	Address 4
5	Customer 5	Address 6

Рис. 69

НАПОМИНАНИЕ

Как мы уже говорили ранее, поле CustomerId имеет отношение «один-ко-многим» с таблицей invoices. Хотя эти данные могут показаться несоответствующими, в нашем случае в результате вернутся четыре записи. Это происходит потому, что клиенту Customer 2 было выставлено два отдельных счета.

Ниже представлен пример использования внутреннего соединения. Сначала в условии SELECT перечислены поля, которые необходимо отобразить, при этом использованы псевдонимы.

```
SELECT
   i.InvoiceId,
   c.CustomerId,
   c.Name,
   c.Address,
   i.InvoiceDate,
   i.BillingAddress,
   i.Total
FROM
```

```
   invoices AS i
INNER JOIN
   customers AS c
ON
   i.CustomerId = c.CustomerId
```

ПРИМЕЧАНИЕ

Поскольку внутренние соединения возвращают только совпадающие данные, порядок перечисления таблиц не имеет значения. Порядок будет иметь значение для других типов соединений.

	InvoiceId	CustomerId	Name	Address	InvoiceDate	BillingAddress	Total
1	1	2	Customer 2	Address 2	1/1/2018	Billing Address 2	$1.00
2	2	2	Customer 2	Address 2	2/1/2018	Billing Address 2	$2.00
3	3	3	Customer 3	Address 3	3/1/2018	Billing Address 3	$3.00
4	4	4	Customer 4	Address 4	4/1/2018	Billing Address 4	$4.00
	4 строки получено за 1 мс						

Рис. 70

В результате выполнения данного запроса мы видим, что возвращено только четыре записи. Invoice 5, Customer 1 и Customer 5 не указаны. Для Customer 2 отображаются две записи.

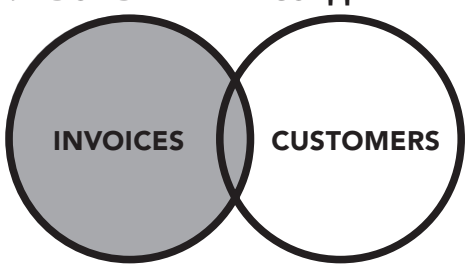
Внутреннее соединение — это наиболее распространенный тип соединения. Оно используется для соединения соответствующих данных из разных таблиц реляционной базы данных.

ПРИМЕЧАНИЕ

Слово «внутреннее» необязательно. Под всеми соединениями понимается внутренние соединение, если не указано иное. Другими словами, по умолчанию все соединения – внутренние.

Левое внешнее соединение (LEFT OUTER JOIN)

Левое внешнее соединение `LEFT OUTER JOIN` соединяет все записи из левой таблицы с любыми совпадающими записями из правой таблицы. На рис. 71 показан эквивалент диаграммы Венна для данного типа соединения.



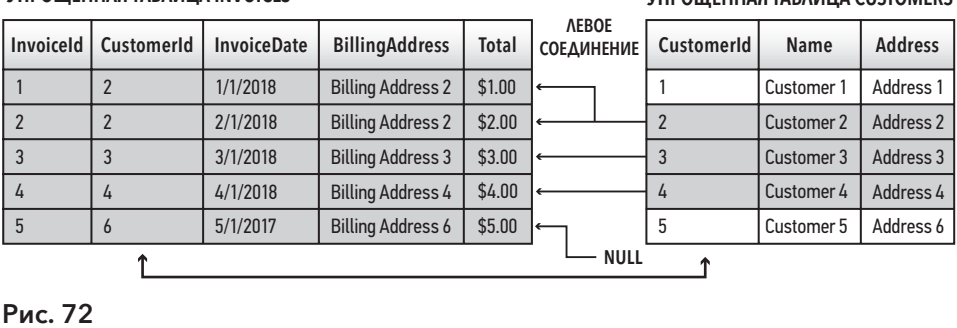

```
SELECT *
FROM invoices AS i
LEFT OUTER JOIN customers AS c
ON i.CustomerId = c.CustomerId
```

Рис. 71

ПРИМЕЧАНИЕ

Понятия «левая таблица» и «правая таблица» полностью зависят от порядка перечисления данных таблиц в соединении JOIN. Изменение порядка перечисления приведет к другому результату. Это различие станет важным, когда вы узнаете, как преобразовывать левые соединения в правые.

При данном типе соединения будет отображаться все, что содержится в таблице `invoices`. Поскольку Customer 1 не заказывал песни, данная запись не указывается. Однако, как показано на рис. 72, мы объединяем все пять записей из таблицы `invoices` только с четырьмя записями из таблицы `customers`. Помните, что в таблице `invoices` записи для Customer 1 или Customer 5 отсутствуют, а клиенту Customer 2 было выставлено два счета. В отличие от внутреннего соединения, которое отображает равное количество записей из каждой таблицы, в результате использования левого внешнего соединения вернется больше записей из «левой» таблицы. Проанализируем результат данного запроса, чтобы понять работу браузера SQL.



InvoiceId	CustomerId	InvoiceDate	BillingAddress	Total
1	2	1/1/2018	Billing Address 2	$1.00
2	2	2/1/2018	Billing Address 2	$2.00
3	3	3/1/2018	Billing Address 3	$3.00
4	4	4/1/2018	Billing Address 4	$4.00
5	6	5/1/2017	Billing Address 6	$5.00

CustomerId	Name	Address
1	Customer 1	Address 1
2	Customer 2	Address 2
3	Customer 3	Address 3
4	Customer 4	Address 4
5	Customer 5	Address 6

Рис. 72

SQL-запрос для левого внешнего соединения аналогичен запросу, который мы использовали для внутреннего соединения. Отличие только в использовании условия `LEFT OUTER JOIN`.

```
SELECT
   i.InvoiceId,
   c.CustomerId,
   c.Name,
   c.Address,
   i.InvoiceDate,
   i.BillingAddress,
   i.Total
FROM
   invoices AS i
LEFT OUTER JOIN
   customers AS c
ON
   i.CustomerId = c.CustomerId
```

	InvoiceId	CustomerId	Name	Address	InvoiceDate	BillingAddress	Total
1	1	2	Customer 2	Address 2	1/1/2018	Billing Address 2	$1.00
2	2	**2**	Customer 2	Address 2	2/1/2018	Billing Address 2	$2.00
3	3	**3**	Customer 3	Address 3	3/1/2018	Billing Address 3	$3.00
4	4	**4**	Customer 4	Address 4	4/1/2018	Billing Address 4	$4.00
5	5	NULL	NULL		5/1/2017	Billing Address 6	$5.00
	5 строк получено за 1 мс						

Рис. 73

ПРИМЕЧАНИЕ

Слово «внешнее» (OUTER) необязательно.

Когда мы анализируем результаты, полученные при использовании левого соединения, мы видим, что браузер SQL добавил данные типа `Null`. Помните, что информация о Customer 6 в таблице `customers` отсутствует. Добавление данных типа `Null` показывает, как браузер SQL обрабатывает нашу попытку сопоставить пять записей из таблицы `invoices` только с четырьмя записями из таблицы `customers`. Использование левого соединения полезно, так как это позволяет нам видеть несовпадения в наших данных. Мы можем создавать списки клиентов, которым не выставляли счета, или выполнять поиск данных, которые были удалены в правой таблице, но все еще существуют в левой.

Правое внешнее соединение (RIGHT OUTER JOIN)

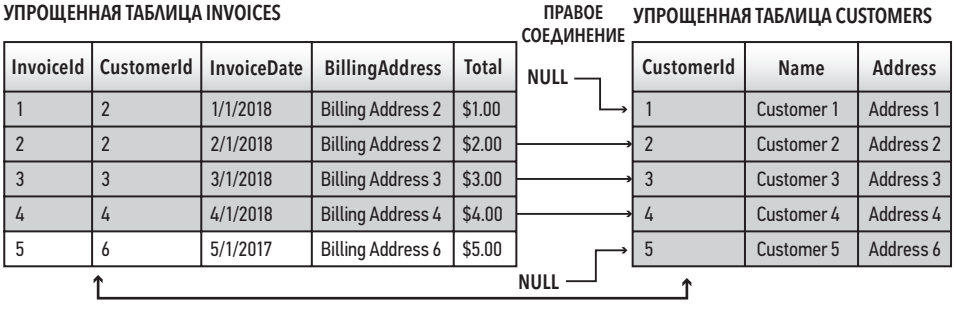
ВНИМАНИЕ

В SQLite не поддерживается использование правого внешнего соединения. Но мы эту тему рассмотрим, поскольку правое внешнее соединение по-прежнему популярно в других реализациях РСУБД. Позже мы рассмотрим обходной путь для применения правых соединений в SQLite.

В результате использования правого внешнего соединения `RIGHT OUTER JOIN` возвращаются все данные из правой таблицы, а также соответствующая информация из левой таблицы. Правое соединение — зеркальное отображение левого соединения.

Рис. 74

При использовании правого соединения берутся все поля из правой таблицы (`customers`) и ее данные сопоставляются с любыми соответствующими данными из таблицы `invoices`. Поскольку Customer 6 в таблице `customers` отсутствует, эта запись игнорируется.



InvoiceId	CustomerId	InvoiceDate	BillingAddress	Total
1	2	1/1/2018	Billing Address 2	$1.00
2	2	2/1/2018	Billing Address 2	$2.00
3	3	3/1/2018	Billing Address 3	$3.00
4	4	4/1/2018	Billing Address 4	$4.00
5	6	5/1/2017	Billing Address 6	$5.00

CustomerId	Name	Address
1	Customer 1	Address 1
2	Customer 2	Address 2
3	Customer 3	Address 3
4	Customer 4	Address 4
5	Customer 5	Address 6

Рис. 75

Оператор SQL, необходимый для создания правого соединения, такой же, как и в двух других соединениях, которые мы рассмотрели ранее.

ПРИМЕЧАНИЕ

Ключевое слово «внешнее» (OUTER) необязательно. Вариант RIGHT JOIN дает тот же результат.

```
SELECT
   i.InvoiceId,
   c.CustomerId,
   c.Name,
   c.Address,
   i.InvoiceDate,
   i.BillingAddress,
   i.Total
FROM
   invoices AS i
RIGHT OUTER JOIN
   customers AS c
ON
   i.CustomerId = c.CustomerId
```

	InvoiceId	CustomerId	Name	Address	InvoiceDate	BillingAddress	Total
1	NULL	1	Customer 1	Address 1	NULL	NULL	NULL
2	1	2	Customer 2	Address 2	1/1/2018	Billing Address 2	$1.00
3	2	2	Customer 2	Address 2	2/1/2018	Billing Address 2	$2.00
4	3	3	Customer 3	Address 3	3/1/2018	Billing Address 3	$3.00
5	4	4	Customer 4	Address 4	4/1/2017	Billing Address 4	$4.00
6	NULL	5	Customer 5	Address 5	NULL	NULL	NULL
	6 строк получено за 2 мс						

Рис. 76

В результате использования данного соединения вернулось наибольшее количество записей из трех изученных ранее соединений. Записей Customer 1 и Customer 5 нет в таблице `invoices`, поэтому им присвоены значения `Null`. Две записи из таблицы `invoices` относятся к Customer 2, поэтому в результате объединения данные Customer 2 были указаны дважды.

Правые соединения используются реже, чем левые. Поскольку SQLite не распознает правое соединение, в запросе рекомендуется изменить порядок таблиц, что приведет к тому же набору результатов. Эту тему мы рассмотрим позже в этой главе.

Внутренние соединения для случаев соединения двух и более таблиц

Можно соединять и более двух таблиц. Чтобы добавить дополнительные таблицы, нужно просто следовать тем же правилам, что мы рассмотрели ранее, когда рассказывали про внутренние соединения. Рассмотрим схему базы данных на рис. 77. Мы видим, что помимо связи между таблицами `invoices` и `customers` существует также связь между полем `SupportRepId` из таблицы `customers` и полем `EmployeeId` из таблицы `employees`.

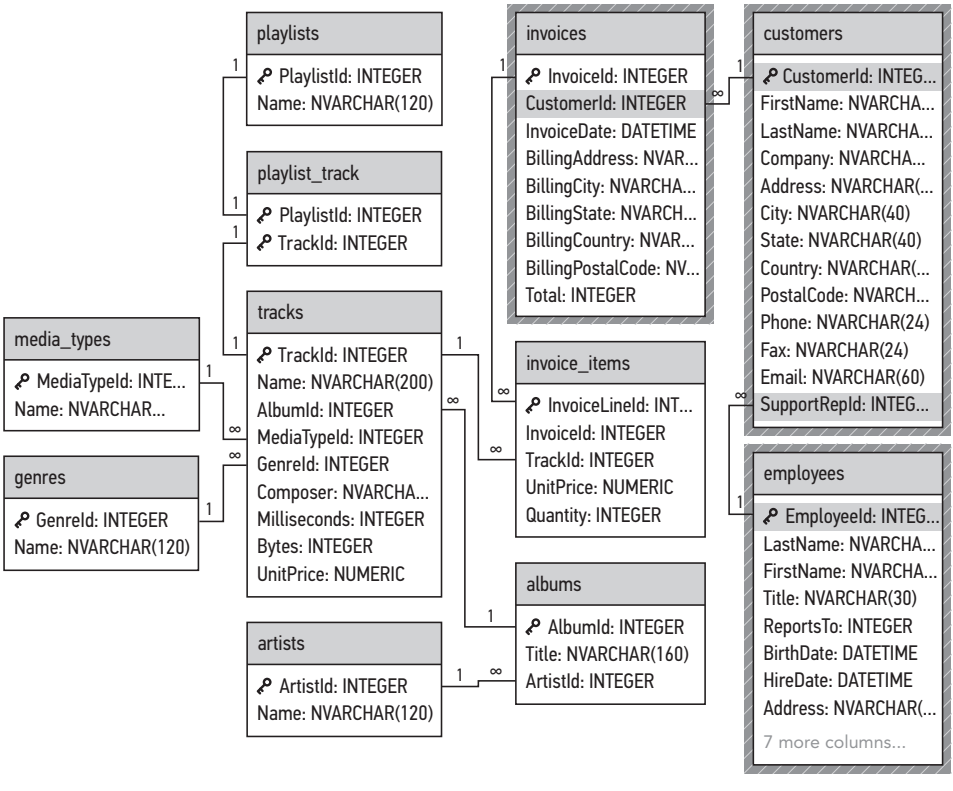


Рис. 77

Обратите внимание, что до этого момента имена двух полей, связанных друг с другом оператором `ON`, были идентичны. В данном случае мы связываем два поля с разными именами, хотя из ER-диаграммы видно, что они являются связанными полями. Почему так происходит? Данное несоответствие дает

нам важный урок о структуре реляционной базы данных. Два связанных поля необязательно должны иметь одно и то же имя. В компании sTunes каждому клиенту назначен `SupportRepId`, или персональный менеджер. Номер, присвоенный каждому менеджеру, совпадает с номером сотрудника (таблица `employees`). Разработчик этой базы данных мог бы назвать оба поля `EmployeeId` (ИН сотрудника), но это может привести к путанице. Хотя клиенту назначен персональный менеджер и логично, что данные `SupportRepId` (ИН менеджера) идентичны данным `EmployeeId` (ИН сотрудника), наличие поля `EmployeeId` в таблице `customers` может вызвать путаницу. Два поля, хранящие идентичные данные, имеют разный смысл в каждой таблице. Если поле названо `SupportRepId` в `customers`, то его назначение здесь не вызывает вопросов. Чтобы не придумывать другую систему нумерации для `EmployeeId`, мы можем связать эти две системы с помощью структуры реляционной базы данных (см. рис. 77).

Теперь, когда мы уже знаем, как связать таблицы `invoices`, `customers` и `employees`, необходимо понять, для чего это может понадобиться. Допустим, отдел обслуживания клиентов sTunes хочет поощрить сотрудников, которым удалось совершить десять самых лучших продаж. Служба поддержки клиентов хочет создать для каждого сотрудника табличку со списком их лучших клиентов. Теперь, когда у нас есть рабочий сценарий, мы можем проанализировать ER-диаграмму, чтобы определить, какие поля необходимы для написания запроса. При написании сложных запросов, в которых задействовано несколько таблиц, полезно продумать, какие поля требуются и из каких таблиц (рис. 78).

НЕОБХОДИМЫЕ ПОЛЯ	СИНТАКСИС

Имя сотрудника, фамилия сотрудника, идентификационный номер сотрудника (данные из таблицы *employees*)	`e.FirstName,`
	`e.LastName,`
	`e.EmployeeId,`

Имя клиента, фамилия клиента, идентификационный номер представителя службы поддержки (данные из таблицы *customers*)	`c.FirstName,`
	`c.LastName,`
	`c.SupportRepId,`

Идентификационный номер клиента, общая сумма покупки (данные из таблицы *invoices*)	`i.CustomerId,` `i.Total,`
Полученный результат необходимо сортировать по сумме счета (в порядке убывания) и ограничить результат до 10 счетов-фактур	`i.Total DESC,` `LIMIT 10`

Рис. 78

Сейчас мы имеем представление о том, какие поля необходимо вывести, и можем приступить к составлению запроса. Начнем с таблицы `invoices` в условии `FROM`. Затем последовательно напишем две операции `INNER JOIN`: одна соединяет счета и клиентов, а другая соединяет и счета, и клиентов с сотрудниками. Затем упорядочим данные по сумме счета (в порядке убывания).

```
SELECT
   e.FirstName,
   e.LastName,
   e.EmployeeId,
   c.FirstName,
   c.LastName,
   c.SupportRepId,
   i.CustomerId,
   i.Total
FROM
   invoices AS i
INNER JOIN
   customers AS c
ON
   i.CustomerId = c.CustomerId
INNER JOIN
   employees AS e
ON
   c.SupportRepId = e.EmployeeId
ORDER BY
   i.Total DESC
LIMIT 10
```

	FirstName	LastName	EmployeeId	FirstName	LastName	SupportRepId	CustomerId	Total
1	Steve	Johnson	5	Helena	Holý	5	6	$25.86
2	Margaret	Park	4	Richard	Cunningham	4	26	$23.86
3	Jane	Peacock	3	Ladislay	Kovács	3	45	$21.86
4	Jane	Peacock	3	Hugh	O'Reilly	3	46	$21.86
5	Steve	Johnson	5	Astrid	Gruber	5	7	$18.86
6	Steve	Johnson	5	Victor	Stevens	5	25	$18.86
7	Steve	Johnson	5	Luís	Rojas	5	57	$17.91
8	Margaret	Park	4	František	Wichterlová	4	5	$16.86
9	Jane	Peacock	3	Isabelle	Mercier	3	43	$16.86
10	Margaret	Park	4	Bjørn	Hansen	4	4	$15.86
	10 строк получено за 5 мс							

Рис. 79

Теперь имеется список сотрудников sTunes, которые обеспечили самые высокие суммы в счетах. Проанализируем некоторые моменты. Как мы и предполагали, хотя поля, которые мы используем для связи таблиц `customers` и `employees`, содержат два разных имени, их данные совпадают. Числовые значения в `SupportRepId` идентичны значениям в поле `EmployeeId`.

Практическое задание

✶ Проанализируйте ER-диаграмму и выберите другую таблицу для добавления к этому запросу с помощью другого внутреннего соединения. Определите необходимые для вывода поля и добавьте их в оператор `SELECT`.

Использование левых внешних соединений с операторами NULL, IS и NOT

Как мы уже говорили ранее в этой главе, левое внешнее соединение извлекает все данные из левой таблицы и всю соответствующую информацию из правой таблицы. Это полезно для анализа базы данных и проверки неполноты информации. Допустим, компания sTunes проводит внутренний аудит, чтобы уточнить, сколько у нее в ассортименте альбомов и отдельных треков. Руководство sTunes просит создать перечень всех исполнителей, которые *не имеют* альбомов. Анализируя предыдущую ER-диаграмму, мы можем предположить, что необходимая информация будет храниться в таблицах `artists` и `albums`. Давайте рассмотрим взаимосвязь между этими таблицами.

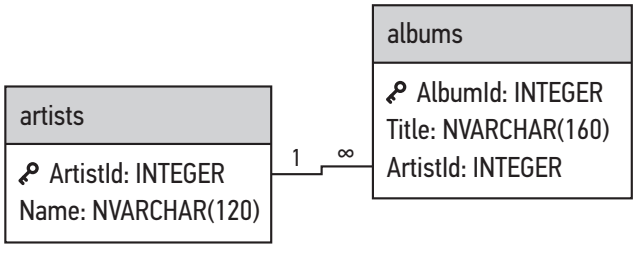


Рис. 80

Таблица `artists` содержит поле `ArtistId` (первичный ключ) и поле для имени исполнителя (рис. 80). Из ER-диаграммы видно, что таблица `artists` связана

с таблицей albums связью «один-ко-многим». Эта связь имеет смысл, так как исполнитель может записать несколько альбомов. В таблице albums имеется собственный первичный ключ AlbumId, а также поле ArtistId как внешний ключ.

Используя соединение LEFT OUTER JOIN для таблицы artists (левая таблица), мы вернем все данные из таблицы artists с соответствующими записями (если они есть) в таблице albums. С помощью левого соединения все поля, не имеющие названий альбомов, будут заполнены значениями NULL. Теперь мы можем создать запрос.

ПРИМЕЧАНИЕ

В качестве псевдонима для большинства соединений в этой главе мы использовали первую букву имени таблицы. Поскольку мы имеем две таблицы, названия которых начинаются с одной и той же буквы, в данном запросе мы для псевдонимов возьмем по две буквы.

```
SELECT
   ar.ArtistId AS [ArtistId From Artists Table],
   al.ArtistId AS [ArtistId From Albums Table],
   ar.Name AS [Artist Name],
   al.Title AS [Album Title]
FROM
   artists AS ar
LEFT OUTER JOIN
   albums AS al
ON
   ar.ArtistId = al.ArtistId
```

Запрос возвращает 418 записей, и вначале все результаты кажутся корректными. Поле ArtistId из таблицы artists соответствует полю ArtistId из таблицы albums. Большинство имен исполнителей связаны с названиями альбомов. Однако далее (рис. 81) мы обнаруживаем значения NULL.

Чтобы решить поставленную нам задачу и получить список исполнителей, у которых нет альбома, необходимо добавить условие WHERE, в котором следует указать только записи со значением NULL в таблице albums. Существуют определенные ключевые слова SQL, которые мы используем для работы со значениями NULL.

- IS NULL в условии WHERE вернет только нулевые значения.
- NOT NULL вернет только значения, которые не были нулевыми.

	ArtistId From Artists Table	ArtistId From Albums Table	Artist Name	Album	
...	...	...	...	...	...
51	25	NULL	Milton Nascimento & Bebeto	NULL	
52	26	NULL	Azymuth	NULL	
53	27	27	Gilberto Gil	As Canções de Eu Tu Eles	
54	27	27	Gilberto Gil	Quanta Gente Veio Ver (Live)	
55	27	27	Gilberto Gil	Quanta Gente Veio ver–Bônus De Carnaval	
56	28	NULL	João Gilberto	NULL	
57	29	NULL	Bebel Gilberto	NULL	
58	30	NULL	Jorge Vercilo	NULL	
59	31	NULL	Baby Consuelo	NULL	
60	32	NULL	Ney Matogrosso	NULL	
...	418 строк получено за 18 мс				

Рис. 81

Раздел `WHERE al.ArtistId IS NULL` вернет список исполнителей без названий альбомов.

ВНИМАНИЕ

При работе со значениями NULL необходимо использовать операторы IS и NOT, а не оператор равенства =. Нулевые значения указывают на недостаток данных. Оператор = сравнивает значения двух элементов. Нулевые значения не содержат значений, поэтому их нельзя сравнивать, используя оператор =. Использование оператора = приведет к ошибке.

```
SELECT
   ar.ArtistId AS [ArtistId From Artists Table],
   al.ArtistId AS [ArtistId From Albums Table],
   ar.Name AS [Artist Name],
   al.Title AS [Album]
FROM
   artists AS ar
LEFT OUTER JOIN
   albums AS al
ON
   ar.ArtistId = al.ArtistId
WHERE
   al.ArtistId IS NULL
```

Полученные результаты (рис. 82) содержат 71 запись, в которой нет альбомов и имен исполнителей.

	ArtistId From Artists Table	ArtistId From Albums Table	Artist Name	Album
1	25	NULL	Milton Nascimento & Bebeto	NULL
2	26	NULL	Azymuth	NULL
3	28	NULL	João Gilberto	NULL
4	29	NULL	Bebel Gilberto	NULL
5	30	NULL	Jorge Vercilo	NULL
6	31	NULL	Baby Consuelo	NULL
7	32	NULL	Ney Matogrosso	NULL
8	33	NULL	Luiz Melodia	NULL
9	34	NULL	Nando Reis	NULL
10	35	NULL	Pedro Luís & A Parede	NULL
…	71 строка получена за 1 мс			

Рис. 82

Преобразование правого соединения в левое

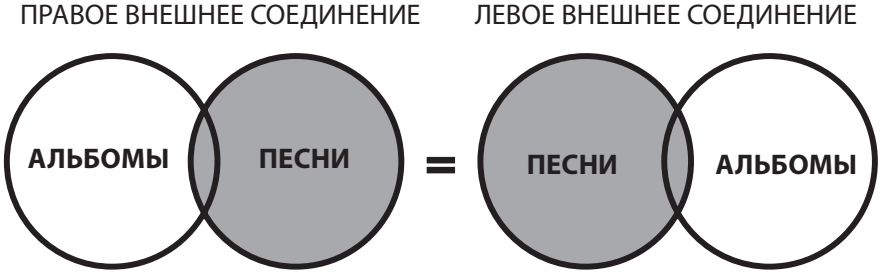
Как мы уже говорили, правые соединения в SQLite не поддерживаются. Мы также узнали, что правые соединения — это зеркальное отображение левых соединений. Рассмотрим диаграмму Венна.

При использовании правого соединения берутся все записи с правой стороны и объединяются со всеми соответствующими записями с левой стороны. Если вы просто поменяете местами левую и правую таблицы, то для получения того же результата вы можете использовать левое внешнее соединение. Следующий запрос написан с использованием правого внешнего соединения. В данном запросе любая соответствующая информация об альбоме или названии из таблицы `albums` объединяется со всеми записями из таблицы `tracks`.

```
SELECT * FROM albums AS al RIGHT OUTER JOIN tracks AS t ON t.AlbumId =
al.AlbumId
```

аналогично

```
SELECT * FROM tracks AS t LEFT OUTER JOIN albums AS al ON t.AlbumId =
al.AlbumId
```



**ПРАВОЕ ВНЕШНЕЕ СОЕДИНЕНИЕ ТАБЛИЦЫ ALBUMS =
ЛЕВОЕ ВНЕШНЕЕ СОЕДИНЕНИЕ ТАБЛИЦЫ TRACKS**

Рис. 83

```
SELECT
   t.TrackId,
   t.Composer,
   t.Name,
   al.AlbumId,
   al.Title
FROM
   albums AS al
RIGHT OUTER JOIN
   tracks AS t
ON
   t.AlbumId = al.AlbumId
```

При выполнении предыдущего запроса возникнет следующая ошибка: `RIGHT and FULL OUTER JOINs are not currently supported` (Правое и полное внешние соединения в настоящее время не поддерживаются).

Однако для решения данной проблемы мы можем просто поменять местами таблицы. Рассмотрим следующий запрос, единственное отличие которого от предыдущего — порядок перечисления таблиц.

```
SELECT
   t.TrackId,
   t.Composer,
   t.Name,
   al.AlbumId,
   al.Title
FROM
   tracks AS t
```

```
LEFT OUTER JOIN
   albums AS al
ON
   t.AlbumId = al.AlbumId
```

Выполним его и проанализируем полученные результаты (рис. 84). Мы получим следующую информацию в одном наборе результатов: композитор, название песни и название альбома. Обратите внимание, что в поле `Composer` (Композитор) обнаружилось несколько нулевых значений. Теперь можно написать другой запрос, чтобы разобраться, в чем дело.

Главный вывод от использования левых/правых соединений: они могут «устранить неполадки» в нашей базе данных и выявить несоответствия в данных.

Если вам нужно найти соответствующие данные и вам не критично потерять несколько записей из-за ошибок в базе данных, то просто пользуйтесь соединением `INNER JOINT`.

	TrackId	Composer	Name	AlbumId	Title
1	1	Angus Young, Malcolm Young, Brian Johnson	For Those About To Rock (We Salute You)	1	For Those About To Rock (We Salute You)
2	2	NULL	Balls to the Wall	2	Balls to the Wall
3	3	F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman	Fast As a Shark	3	Restless and Wild
4	4	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. Dirkscneider & W. Hoffman	Restless and Wild	3	Restless and Wild
5	5	Deaffy & R.A. Smith-Diesel	Princess of the Dawn	3	Restless and Wild
6	6	Angus Young, Malcolm Young, Brian Johnson	Put The Finger On You	1	For Those About To Rock (We Salute You)
7	7	Angus Young, Malcolm Young, Brian Johnson	Let's Get It Up	1	For Those About To Rock (We Salute You)
8	8	Angus Young, Malcolm Young, Brian Johnson	Inject The Venom	1	For Those About To Rock (We Salute You)
9	9	Angus Young, Malcolm Young, Brian Johnson	Snowballed	1	For Those About To Rock (We Salute You)
10	10	Angus Young, Malcolm Young, Brian Johnson	Evil Walks	1	For Those About To Rock (We Salute You)
...	3503 строки получено за 25 мс				

Рис. 84

Практическое задание

✶ Измените приведенный выше запрос так, чтобы отображались только записи, в которых поле `Composer` содержит значение `NULL`.

Контрольные вопросы

1. Используя DB Browser и вкладку Browse Data (Просмотр данных) или ER-диаграмму (рис. 65), проанализируйте таблицу `tracks`. Определите, какие поля в этой таблице будут внешними ключами в другой таблице. На основании определенных вами внешних ключей определите, какие таблицы связаны с таблицей `tracks`.
2. Создайте внутреннее соединение между таблицами `albums` и `tracks` и отобразите названия альбомов и названия треков в едином наборе результатов.
3. Создайте третье внутреннее соединение — с таблицей `genres`, которую вы нашли, отвечая на вопрос 1. Включите в ваш набор результатов поле `Name` из этой таблицы.

Резюме

- Соединения используются для соединения данных из разных таблиц.
- При написании соединений полезно использовать ER-диаграмму.
- При одновременном выборе полей из нескольких таблиц для указания исходной таблицы необходимо использовать псевдонимы.
- Внутренние соединения не включают строки, для которых нет соответствующих данных.
- Внешние соединения включают все строки одной из таблиц, даже если между таблицами нет соответствующих данных. Несовпадающие строки будут отображаться как `NULL`.
- Для проверки нулевых значений необходимо использовать специальные операторы `IS` и `NOT`.
- Правые соединения могут использоваться в реализациях SQL, отличных от SQLite. Чтобы в SQLite выполнить эквивалент правого соединения, просто поменяйте местами две сравниваемые таблицы в операторе запроса.

ГЛАВА 7

Функции языка SQL

Краткое содержание

- ✓ Добавление вычислений к запросам
- ✓ Типы функций в SQL
- ✓ Строковые функции
- ✓ Функции даты и времени
- ✓ Агрегатные функции
- ✓ Использование с функциями условий `WHERE` и `HAVING`
- ✓ Группировка по нескольким столбцам
- ✓ Контрольные вопросы

Если вы проверяли свои знания с помощью контрольных вопросов в конце каждой главы, вы, возможно, заметили, что для ответа на некоторые вопросы требуются дополнительные действия после написания запроса. Например, в главе 4 мы задали вопрос: сколько фамилий клиентов начинаются с буквы *B*? Или попросили найти итоговые суммы счетов в определенном ценовом диапазоне. Чтобы ответить на такие вопросы, требовалось вручную подсчитывать результаты запроса. Чтобы получить все фамилии, начинающиеся на букву *B*, при помощи информации из главы 4 вы могли выбрать поле `LastName` в таблице клиентов, отсортировать по фамилии, прокрутить вниз до тех, что начинаются с буквы *B*, а затем подсчитать их вручную. Если применить материал из главы 5, можно немного упростить задачу, ограничив набор данных, чтобы возвращались только фамилии, начинающиеся с буквы *B* (запрос `WHERE LastName LIKE 'B%'`). Однако вам все равно придется считать вручную. В этой главе мы расскажем, как упростить вычисления с помощью функций.

Добавление вычислений к запросам

Вычисления (например, подсчет количества полученных записей) можно выполнять, добавляя функции к запросам. Используя функцию `COUNT()`, можно произвести подсчет `LastName` и использовать псевдоним `NameCount` для возврата значения. Добавим функцию к базовому оператору `SELECT`:

```
SELECT
   COUNT(LastName) AS [NameCount]
FROM
   customers
WHERE
   LastName LIKE 'B%'
```

	NameCount
1	4
	1 строка получена за 2 мс

Рис. 85

Для суммирования всех полей в таблице `customers`, удовлетворяющих условию `WHERE`, мы можем использовать функцию `COUNT()`, а не перечислять все записи, начинающиеся с буквы *B* (чтобы их затем подсчитать). Конечный результат — это поле с псевдонимом и количество фамилий, начинающихся с буквы *B*.

Это всего лишь один пример того, как использование функций может сэкономить время, выполняя за нас часть работы. В данной главе мы рассмотрим три различных типа функций и проиллюстрируем наиболее полезные.

Типы функций в SQL

Функции в SQL — это специальные ключевые слова, которые принимают определенные параметры, выполняют определенные операции (например, вычисление или изменение данных в поле) и возвращают результат в виде значения. В главе 5 мы уже рассматривали функцию `DATE()`. В качестве параметров были заданы данные в формате `DATETIME`, выполнена операция (отредактирован формат написания времени), а затем в результате был возвращен только определенный фрагмент даты. Как мы уже упоминали в главе 5, в SQL для выполнения задач

предусмотрено множество типов функций. На рис. 86 показаны наиболее распространенные и полезные.

ПРИМЕЧАНИЕ

Этот список неполный. Все функции, упомянутые в этой книге, поддерживаются SQLite. Другие реализации базы данных содержат другие функции. Полный список функций, поддерживаемых SQLite, доступен на веб-странице SQLite [10].

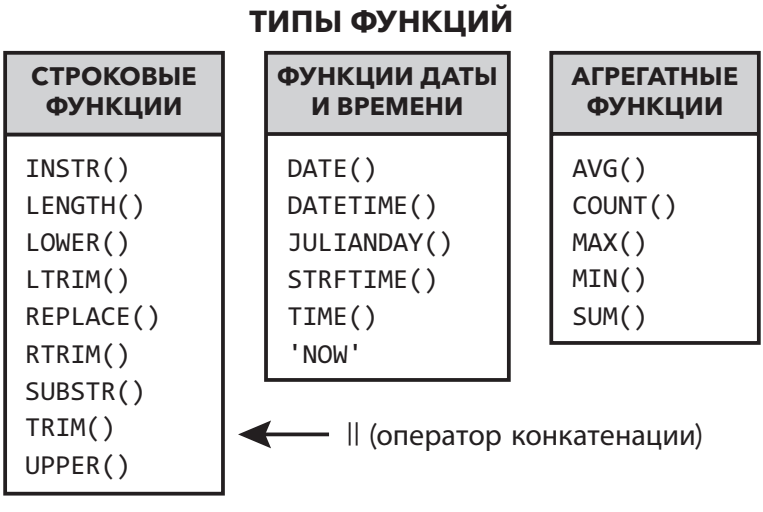


Рис. 86

Как видно на рис. 86, в SQL существуют функции трех разных типов.

- *Строковые функции* изменяют символьные и текстовые данные.
- *Функции даты и времени* изменяют данные времени и даты.
- *Агрегатные функции* выполняют математические операции.

ПРИМЕЧАНИЕ

На первый взгляд может показаться, что эти три типа функций работают только с соответствующими им типами данных (символами, датами и числами). Однако вспомните первый пример с использованием функции COUNT(). Мы смогли использовать агрегатную функцию для арифметического подсчета символьных данных. В определенных случаях мы можем использовать эти функции с разными типами данных. Дополнительные сведения о типах данных см. в главе 1.

Функции в SQL работают подобно функциям в программах для работы с электронными таблицами и в других языках программирования. Если вы когда-либо использовали функцию `SUM()` в Microsoft Excel, то вы понимаете, для чего она нужна. Еще одно сходство этих функций с функциями Excel — это подсказка в виде всплывающего окна, которое появляется, когда мы вводим функцию в браузере SQL. Рассмотрим следующий пример. На панели Execute SQL (Выполнить SQL-запрос) начните вводить имя функции с одной открытой скобкой:

```
UPPER(
```

Под названием функции появится следующий текст:

```
The UPPER(X) function returns a copy of input string X in which all
lowercase ASCII characters are converted to their uppercase equivalent.
```

(Функция `UPPER(X)` возвращает копию входной строки `X`, в которой все символы ASCII нижнего регистра преобразованы в их эквивалент в верхнем регистре.)

ПРИМЕЧАНИЕ

Часть функции в круглых скобках (X) называется аргументом функции. Некоторые функции могут содержать более одного аргумента.

Большинство функций в DB Browser содержат всплывающие подсказки, которые очень полезны для определения *аргументов*, принимаемых функцией, и описания работы функции. Я рекомендую изучить и другие функции.

Практические задания

- Функция `UPPER()` принимает только один аргумент (`X`). Сколько аргументов принимает функция `REPLACE()`?
- Прочитайте всплывающую подсказку (на экране в DB Browser) и объясните, как работает функция `TRIM()`.

ВНИМАНИЕ

Если всплывающая подсказка не появилась, убедитесь, что вы правильно набираете функцию. Подсказка не появится, если вы просто скопируете и вставите функцию и открытую скобку. Это еще одна причина, по которой всегда необходимо вводить запросы вручную. Никаких ускоренных методов!

Управление текстовыми данными с помощью строковых функций

Строка — это еще одно название для текстовых данных. Строковые функции позволяют форматировать и изменять текст. Чтобы понять, как они работают, вспомним типы данных в SQLite. В главе 3, анализируя структуру базы данных, мы видели, что текстовые данные были сохранены в формате `NVARCHAR(X)`. `NVARCHAR` — это строка переменной длины, где `X` представляет максимальную длину строки.

НАПОМИНАНИЕ

Для символьных данных фиксированной длины (например, почтовых индексов, содержащих буквы) можно использовать другой тип данных – с фиксированной длиной. Однако в базе данных sTunes символьный тип данных NVARCHAR используется для всех текстовых данных.

Возможность манипулировать текстовыми строками важна, так как поля в базе данных не всегда организованы удобно для нас. Допустим, нас попросили создать список рассылки по клиентам из США. Для этого необходимо получить имена и адреса клиентов. Используем оператор `SELECT`.

```
SELECT
   FirstName,
   LastName,
   Address
FROM
   customers
WHERE
   Country = 'USA'
```

При выполнении запроса получим следующий результат:

	FirstName	LastName	Address
1	Frank	Harris	1600 Amphitheatre Parkway
2	Jack	Smith	1 Microsoft Way
3	Michelle	Brooks	627 Broadway
4	Tim	Goyer	1 Infinite Loop
5	Dan	Miller	541 Del Medio Avenue
...	13 строк получено за 5 мс		

Рис. 87

Однако возникают некоторые проблемы. Данные адреса разбиты на части. Просто запросить поле адреса недостаточно. Нам также необходимо выбрать поля `City`, `State` и `PostalCode`. Другая проблема: все эти данные содержатся в отдельных полях. Если необходимо создать простой список рассылки, где каждая строка представляет собой полное имя и адрес клиента, то для этой цели полученный шаблон не годится.

Практическое задание

✶ Попробуйте скопировать результат последнего запроса из DB Browser в текстовый редактор. Что вы видите?

К счастью, для манипулирования текстовыми строками существует несколько отличных инструментов, так что мы можем получить результат в удобном для нас виде. Рассмотрим первый из этих инструментов — конкатенацию.

Конкатенация строк

Конкатенация — объединение двух или более строк. Для слияния двух полей вместе используется символ `||`. Например, следующий код выполняет объединение полей `FirstName` и `LastName`.

```
SELECT
   FirstName || LastName
FROM
   customers
WHERE
   CustomerId = 1
```

Получится следующий результат:

	FirstName || LastName
1	LuísGonçalves
	1 строка получена за 1 мс

Рис. 88

Оператор конкатенации `||` просто соединил оба поля вместе без пробелов. Для удобства чтения мы можем последовательно использовать две конкатенации

и заключить пробел в одинарные кавычки. Тогда запрос будет выглядеть следующим образом:

```
SELECT
   FirstName,
   LastName,
   FirstName || ' ' || LastName
FROM
   customers
WHERE
   Country = "USA"
```

При выполнении запроса получим следующий результат:

	FirstName	LastName	FirstName || LastName
1	Frank	Harris	Frank Harris
2	Jack	Smith	Jack Smith
3	Michelle	Brooks	Michelle Brooks
4	Tim	Goyer	Tim Goyer
5	Dan	Miller	Dan Miller
...	13 строк получено за 1 мс		

Рис. 89

Практическое задание

✶ Сделайте результаты вывода более читабельными, создав псевдоним `Full Name` для нашего объединенного поля.

ПРИМЕЧАНИЕ

Функция конкатенации || не похожа на остальные функции, которые мы будем рассматривать в этой главе. В других реализациях SQL существует функция CONCAT() или используется символ +. Синтаксис может отличаться в зависимости от того, какую РСУБД вы используете, но эффект будет одинаковым.

Рассмотрим следующий пример, где для создания в одной строке имени и адреса клиента используется множественная конкатенация.

```
SELECT
   FirstName || ' ' || LastName || ' ' || Address
   || ', ' || City || ', ' || State || ' ' ||
```

```
   PostalCode AS [MailingAddress]
FROM
   customers
WHERE
   Country = "USA"
```

При выполнении запроса получим следующий результат:

	MailingAddress
1	Frank Harris 1600 Amphitheatre Parkway, Mountain View, CA 94043-1351
2	Jack Smith 1 Microsoft Way, Redmond, WA 98052-8300
3	Michelle Brooks 627 Broadway, New York, NY 10012-2612
4	Tim Goyer 1 Infinite Loop, Cupertino, CA 95014
5	Dan Miller 541 Del Medio Avenue, Mountain View, CA 94040-111
	…
	13 строк получено за 5 мс

Рис. 90

Практическое задание

- После выполнения запроса скопируйте полученный в DB Browser результат в текстовый редактор. Он будет намного читабельнее, чем до выполнения конкатенации.

ПРИМЕЧАНИЕ

Пробелы между символами || не обязательны. В предыдущем примере, чтобы получить желаемый формат, мы использовали запятую, а затем пробел. Вы можете добавить любой текст, поместив его в кавычки.

Обрезка строки

С помощью функций мы также можем вырезать из текста определенное количество символов. Из приведенного выше примера видно, что почтовые индексы США в таблице `customers` не единообразны. Некоторые из них содержат дефис и дополнительный четырехзначный номер, который почтовая служба США называет кодом ZIP +4. В одном из почтовых индексов отсутствует четвертая цифра почтового кода ZIP +4.

НА МОЙ ВЗГЛЯД,

обнаружение ошибок или несоответствий в базе данных неизбежно. Способность предусматривать ошибки и исключения и обрабатывать их – часть процесса обучения и бесценный навык. Поля часто содержат орфографические ошибки, неправильное количество символов или другие несоответствия. В дальнейшем вы научитесь предусматривать и обрабатывать ошибки по мере их появления.

Чтобы почтовый индекс был единообразным, мы можем использовать функции для удаления дефиса и лишних чисел. Благодаря усовершенствованной системе маршрутизации почтовой службы США дополнительные номера в почтовых индексах США (сверх начальных пяти) не требуются.

Чтобы использовать функции редактирования строк, нам необходимо больше узнать о хранении строк в базе данных. Символы в каждой строке нумеруются, начиная с 1. Это важно при работе со строками, так как позволяет указать фрагмент строки, используя порядковый номер символа.

Мы используем функцию `LENGTH()` в таком поле, как `PostalCode` (таблица `customers`), и видим, что длина каждого кода может быть вычислена.

```
SELECT
   PostalCode,
   LENGTH(PostalCode) AS [Postal Code Length]
FROM
   customers
WHERE
   Country = "USA"
```

PostalCode	Postal Code Length
94043-1351	10
98052-8300	10
10012-2612	10
95014	5
94040-111	9
89503	5
32801	5
2113	4
60611	5
53703	5
...	13 строк получено за 1 мс

Рис. 91

Анализируя результаты (рис. 91), мы видим, что почтовые индексы различаются размером строк. Минимальная необходимая длина почтового индекса США — пять цифр. С помощью функции `SUBSTR()` удалим всю информацию после пятой позиции в строке.

Существует два вида функции `SUBSTR()`: `SUBSTR(X, Y)` и `SUBSTR(X, Y, Z)`. Чтобы посмотреть краткое описание функции, введите в браузере `SUBSTR(`.

ПРИМЕЧАНИЕ

Чтобы DB Browser отображал описание функции `SUBSTR(X, Y, Z)`, необходимо ввести `SUBSTR (X,Y,` , что означает, что вас интересует версия функции с тремя аргументами.

ФУНКЦИЯ	ОПИСАНИЕ
`SUBSTR(X,Y)`	Извлекает все символы до позиции X, начиная с позиции Y.
`SUBSTR(X,Y,Z)`	Извлекает из строки X подстроку заданной длины Z, начиная от заданной начальной позиции Y.

Рис. 92

Если для удаления дополнительных данных из почтовых индексов США мы используем функцию `SUBSTR (X, Y, Z)`, аргумент `X` будет соответствовать полю `PostalCode`, а аргумент `Y` — начальной позиции строки. В данном случае мы хотим, чтобы первые пять чисел остались, поэтому выберем `1`. Аргумент `Z` указывает количество символов, которые функция вернет из начальной позиции, в нашем случае оно равно `5`.

Если мы выберем только адреса США, а затем добавим функцию `SUBSTR (X, Y, Z)` с псевдонимом, мы получим следующее:

```
SELECT
   PostalCode,
   SUBSTR(PostalCode,1,5) AS [Five Digit Postal Code]
FROM
   customers
WHERE
   Country = "USA"
```

Анализируя результат (рис. 93), мы видим, что все наши почтовые индексы содержат только первые пять цифр.

ПРИМЕЧАНИЕ

Почтовые индексы, которые не содержат дополнительных номеров, этот запрос не затрагивает.

Мы также можем разделить данные, используя функцию `SUBSTR()`. Обратите внимание, что версия функции `SUBSTR()`, принимающей два аргумента, возвращает все символы до конца строки, начиная с позиции `Y`. Если позиция `Y` не начинается с `1`, мы можем указать функции, чтобы она возвращала только символы, начиная с позиции `Y` и далее.

PostalCode	Five Digit Postal Code
94043-1351	94043
98052-8300	98052
10012-2612	10012
95014	95014
94040-111	94040
89503	89503
32801	32801
2113	2113
60611	60611
53703	53703
…	13 строк получено за 1 мс

Рис. 93

Практическое задание

✶ Повторите предыдущий запрос, но в блок `SELECT` добавьте дополнительное поле. Используйте функцию `SUBSTR(X, Y)`, возвращающую только последние четыре цифры почтового индекса (там, где они есть) с псевдонимом ZIP + 4 Code.

Дополнительные строковые функции

В нашей книге перечислена лишь часть функций. Как упоминалось в начале этой главы, полный список функций, поддерживаемых SQLite, можно найти на сайте SQLite. Альтернативный способ изучить новые функции — ввести каждую, прочитать всплывающую подсказку и попытаться понять, как функция работает. Прежде чем приступить к функциям даты и времени и агрегатным функциям, мы изучим еще две полезные строковые функции.

ФУНКЦИЯ	ОПИСАНИЕ
`UPPER()`	Переводит все символы аргумента в верхний регистр.
`LOWER()`	Переводит все символы аргумента в нижний регистр.

Рис. 94

Функция `UPPER(X)` возвращает копию входной строки `X`, в которой все символы ASCII в нижнем регистре переведены в верхний регистр. Функция `LOWER()` переводит все символы аргумента в нижний регистр.

Рассмотрим пример использования этих функций:

```
SELECT
   FirstName as [First Name Unmodified],
   UPPER(FirstName) as [First Name in UPPERCASE],
   LOWER(FirstName) as [First Name in lowercase],
   UPPER(FirstName) || ' ' || UPPER(LastName) AS [Full Name in
UPPERCASE]
FROM
   customers
```

В данном запросе в качестве аргумента функций `UPPER()` и `LOWER()` выступает поле `FirstName`. Также для обозначения результата мы использовали псевдоним `Full Name`, чтобы показать, что вы можете объединить два поля после выполнения функций.

ПРИМЕЧАНИЕ

Функции UPPER() и LOWER() работают только с символами ASCII. Любые символы, изначально отформатированные в Юникоде, остаются без изменений.

	First Name Unmodified	First Name in UPPERCASE	First Name in lowercase	Full Name in UPPERCASE
1	Luís	LUíS	luís	LUíS GONçALVES
2	Leonie	LEONIE	leonie	LEONIE KöHLER
3	François	FRANçOIS	françois	FRANçOIS TREMBLAY
4	Bjørn	BJøRN	bjørn	BJøRN HANSEN
5	František	FRANTIšEK	františek	FRANTIšEK WICHTERLOVá
6	Helena	HELENA	helena	HELENA HOLý
7	Astrid	ASTRID	astrid	ASTRID GRUBER
8	Daan	DAAN	daan	DAAN PEETERS
9	Kara	KARA	kara	KARA NIELSEN
10	Eduardo	EDUARDO	eduardo	EDUARDO MARTINS
...	59 строк получено за 3 мс			

Рис. 95

Практическое задание

✶ Используйте функцию `SUBSTR()` вместе с функциями `UPPER()` и `LOWER()` и создайте список клиентов: фамилия должна быть выведена первой и всеми заглавными буквами, а от имени следует оставить только первую букву.

Функции даты и времени

Функции даты и времени позволяют управлять данными, хранящимися в различных форматах даты и времени. В базе данных sTunes информация о дате хранится в формате `DATETIME`: `YYYY-MM-DD HH:MM:SS`. Хотя формат позволяет вводить временной код, он не используется в нашей базе данных — все временные коды пусты и отображают 00:00:00. Поэтому, чтобы убрать временной код и оставить только информацию о дате, мы использовали функцию `DATE()` (см. главу 5). Поскольку в разных базах данных информация о дате может храниться по-разному, важно знать, как преобразовать один формат в другой. С датами мы можем сделать гораздо больше, чем просто изменить их формат. Например, чтобы вычислить возраст сотрудников, мы можем посчитать разницу между любой заданной и текущей датой, поскольку в таблице `employees` имеется поле `BirthDate` (Дата рождения).

Чтобы рассчитать возраст сотрудников, рассмотрим функцию `STRFTIME()`, также известную как функция времени в строковом формате. Она позволяет

отформатировать информацию о времени и дате в виде текстовой строки. Для корректной работы функции `STRFTIME()` требуется как минимум информация двух видов. Необходимо указать желаемый формат (его называют спецификацией преобразования) и строку времени для форматирования. Строку времени можно ввести вручную или использовать поле `DATETIME`. В качестве аргумента строка времени также может использовать функцию `NOW`. Третий аргумент, модификатор, необязательный и может применяться для дискретного сдвига даты вперед или назад и выполнения некоторых других функций.

ФУНКЦИЯ	ОПИСАНИЕ
`STRFTIME()`	`STRFTIME(format, timestring, modifier, modifier, ...)` Возвращает дату, отформатированную в соответствии со строкой формата, указанной в качестве первого аргумента. Второй параметр используется для упоминания строки времени, за ним следует один или несколько модификаторов (необязательный параметр).
`'NOW'`	Возвращает текущую дату и время. Функция не содержит аргументов.

Рис. 96

ПРИМЕЧАНИЕ

Функцию NOW иногда называют недетерминированной функцией. Это означает, что результирующие данные, возвращаемые этой функцией, будут разными при каждом ее вызове, поскольку дата и/или время будут разными при каждом вызове. Функция STRFTIME() и большинство других функций, описанных в этой книге, – детерминированные. То есть они дают один и тот же результат каждый раз, когда используются с одними и теми же аргументами. Функцию NOW необходимо постоянно обновлять, чтобы она оставалась точной и результат соответствовал времени вашего компьютера.

ПРИМЕЧАНИЕ

Функция STRFTIME() способна выполнять различные преобразования времени и дат. Если вы пока не совсем понимаете значения всех аргументов и выполняемых преобразований, не беспокойтесь. Главное то,

что функция STRFTIME() принимает данные в формате времени и даты и использует ключевые слова для возврата определенных пользователем фрагментов даты.

АРГУМЕНТЫ ФУНКЦИИ STRFTIME
в определенном порядке: формат, строка времени, модификатор (опционально)

(формат ((format)) *Чувствителен к регистру*	ОПИСАНИЕ
'%d'	день месяца: 00
'%f'	доли секунды: SS.SSS
'%H'	час: 00-24
'%j'	день года: 001-366
'%J'	юлианская дата
'%m'	месяц: 01-12
'%M'	минута: 00-59
'%s'	количество секунд с 1 января 1970 года
'%S'	секунда: 00-59
'%w'	день недели: 0–6 (воскресенье = 0)
'%W'	неделя года: 00-53
'%Y'	годы: 0000-9999

Рис. 97

(строка времени (timestring))	ОПИСАНИЕ
'YYYY-MM-DD'	Формат даты: Год-Месяц-День
'now'	Текущие дата и время
'DATETIME' поле	Поле базы данных в формате даты и/или времени

Рис. 98

(модификатор (modifier))	ОПИСАНИЕ
'+ X days'	Где X – количество дней, добавляемых к результату
'+ X months'	Где X – количество месяцев, добавляемых к результату
'+ X years'	Где X – количество лет, добавляемых к результату
'- X days'/months/years	Где X – количество дней/месяцев/лет, вычитаемых из результата
'start of day'	Сдвиг времени на начало дня
'start of month'	Сдвиг даты на начало месяца
'start of year'	Сдвиг даты на начало года

Рис. 99

Мы можем использовать любые символы внутри одинарных кавычек при условии, что всю строку заключаем в кавычки.

```
SELECT
   STRFTIME('The Year is: %Y The Day is: %d The Month is %m', '2011-05-
22') AS [Text with Conversion Specifications]
```

	Text with Conversion Specifications
1	The Year is: 2011 The Day is: 22 The Month is 05
	1 строка получена за 1 мс

Рис. 100

НАПОМИНАНИЕ

Спецификация в функции преобразования всегда начинается с символа %, за которым следует чувствительный к регистру буквенный символ. Использование, например, %M (верхний регистр) вместо %m даст нам минуты вместо месяцев.

Давайте рассмотрим работу функции `STRFTIME()` на примере вычисления возраста сотрудников. Первое, что необходимо сделать, это указать необходимый формат. Поскольку `BirthDate` (Дата рождения) имеет тип данных `DATETIME`, а время в нашей базе данных не указано, для простоты опустим временные коды. Чтобы узнать возраст сотрудников, вычислим разницу во времени между датой рождения каждого сотрудника и текущей датой. Текущую дату можно получить, используя функцию `NOW`.

```
SELECT
   LastName,
   FirstName,
   STRFTIME('%Y-%m-%d',BirthDate) AS [Birthday No Timecode],
   STRFTIME('%Y-%m-%d','now') - STRFTIME('%Y-%m-%d', BirthDate) AS [Age]
FROM
   employees
ORDER BY
   Age
```

	LastName	FirstName	Birthday No Timecode	Age
1	Peacock	Jane	8/29/1973	46
2	Mitchell	Michael	7/1/1973	46
3	King	Robert	5/29/1970	49
4	Callahan	Laura	1/9/1968	51
5	Johnson	Steve	3/3/1965	54
6	Adams	Andrew	2/18/1962	57
7	Edwards	Nancy	12/8/1958	61
8	Park	Margaret	9/19/1947	72
	8 строк получено за 1 мс			

Рис. 101

Мы можем использовать функцию `STRFTIME()` так же, как мы использовали функцию `DATE()` для удаления временных кодов. Затем, чтобы узнать возраст сотрудников, необходимо получить разницу между двумя функциями `strftime`.

Практические задания

- Компания sTunes отмечает дни рождения сотрудников первого числа каждого месяца. Создайте для отдела кадров таблицу, отображающую имена сотрудников, дни рождения и день празднования.

- ✶ Отдел кадров компании sTunes сообщил нам, что возраст сотрудников — это тема весьма деликатная. Перепишите этот запрос, чтобы в результате получить количество лет работы каждого сотрудника в компании.
- ✶ Какой сотрудник проработал в компании дольше всех?

Агрегатные функции

Агрегатные функции воздействуют на значения столбца, чтобы получить единое результирующее значение с помощью различных математических операций. В начале этой главы, чтобы вычислить количество клиентов с фамилией, начинающейся с буквы *B*, мы использовали функцию `COUNT()`. Существует множество практических способов использования агрегатных функций. Так, используем функцию `SUM()` в таблице `invoices` для вычисления итоговой суммы всех счетов:

```
SELECT
   SUM(Total) AS [Total Sales]
FROM
   invoices
```

Существует множество агрегатных функций [11], но здесь мы рассмотрим пять основных функций в SQL: `SUM()`, `AVG()`, `MIN()`, `MAX()` и `COUNT()`.

ФУНКЦИЯ	ОПИСАНИЕ
`SUM()`	Возвращает суммарное значение столбца таблицы базы данных
`AVG()`	Возвращает среднее значение для столбца
`MIN()`	Возвращает минимальное значение для столбца
`MAX()`	Возвращает максимальное значение для столбца
`COUNT()`	Возвращает количество записей таблицы базы данных

Рис. 102

ПРИМЕР

Напишем следующий запрос:

```
SELECT
   SUM(Total) AS TotalSales,
   AVG(Total) AS AverageSales,
   MAX(Total) AS MaximumSale,
   MIN(Total) AS MinSale,
   COUNT(*) AS SalesCount
FROM
   invoices
```

И получим следующий результат:

	TotalSales	AverageSales	MaximumSale	MinSale	SalesCount
1	"2328.6"	"5.651941..."	"25.86"	"0.99"	"412"
	1 строка получена за 2 мс				

Рис. 103

ПРИМЕЧАНИЕ

По умолчанию функция COUNT() возвращает только ненулевые значения. Однако если необходимо подсчитать все записи, даже записи с ошибками или нулевыми значениями, рекомендуется использовать символ звездочки * или поле первичного ключа. Символ звездочки * обозначает «вернуть все записи». Поэтому, используя ее с агрегатной функцией COUNT(), мы получим количество всех записей в таблице invoices.

Практические задания

- Сколько счетов содержится в таблице `invoices`?
- Какова средняя сумма счета?
- Какова сумма самого большого счета в таблице `invoices`?

Вложенные функции на примере ROUND()

Вложенная функция — это та, которая содержится в другой функции. Одна из целей использования вложенных функций — модифицировать формат внутренней функции. Если мы проанализируем предыдущий пример, в котором

мы использовали функцию AVG(), то увидим, что Average Sales (средний объем продаж) содержит слишком много десятичных знаков. Такой формат обычно не используется для денежных единиц. Функция ROUND(), хотя и не агрегатная, очень полезна при выполнении каких-либо математических операций или если требуется привести в порядок результаты. Для этой цели функцию AVG() можно поместить в функцию ROUND() (это и есть вложение) и указать количество десятичных знаков, до которого мы хотим округлить результат.

ФУНКЦИЯ	ОПИСАНИЕ
ROUND(X,Y)	Функция ROUND() округляет число X до указанного числа десятичных знаков Y. Если аргумент Y отсутствует, это означает, что он равен 0.

Рис. 104

```
SELECT
   AVG(Total) AS [Average Sales],
   ROUND(AVG(Total), 2) AS [Rounded Average Sales]
FROM
   invoices
```

	Average Sales	Rounded Average Sales
1	5.65194174757283	5.65
	1 строка получена за 1 мс	

Рис. 105

ВНИМАНИЕ

При использовании функции ROUND() с денежными единицами будьте внимательны при округлении и изменении значений в промежуточных вычислениях. Обычно округление выполняется только на последнем шаге. Также вы можете добавить комментарии, чтобы указать, что результаты округляются до двух знаков после запятой.

Использование агрегатных функций и условия GROUP BY

Полезной особенностью агрегатных функций считается их способность вычислять промежуточные значения, или агрегаты, для различных групп дан-

ных. Для таблицы `invoices` в базе данных sTunes мы можем легко получить среднюю сумму счета с помощью функции `AVG()`. Предположим, компании sTunes необходимо рассчитать среднюю сумму счета для каждого города, где его выставили.

ВНИМАНИЕ

Следующий запрос написан неправильно, чтобы показать, что происходит при сочетании в операторе `SELECT` агрегатных функций с неагрегатными полями. Этот запрос не вызывает ошибок, но он неправильно отображает запрашиваемую информацию.

```
SELECT
   BillingCity,
   AVG(Total)
FROM
   invoices
ORDER BY
   BillingCity
```

Запустите этот запрос и проанализируйте результаты.

	BillingCity	AVG(Total)
1	Delhi	5.65194174757283
	1 строка получена за 1 мс	

Рис. 106

Нам требовалось получить среднюю сумму счета из таблицы `invoices` для каждого города. Несмотря на то что мы включили город в оператор `SELECT`, запрос по-прежнему дает нам только глобальное среднее значение всех счетов. Почему наш запрос не возвращает среднюю сумму для каждого города из таблицы `invoices`?

Чтобы решить эту задачу, давайте проанализируем запрос. Нам задали вопрос: какова средняя сумма счетов по городам?

НАПОМИНАНИЕ

Мы уже упоминали, что полезно разбивать запрос на компоненты, а также поразмыслить, какая таблица содержит нужную информацию и как ее отобразить. Ответ на эти два вопроса поможет вам устранить недочеты, связанные с запросом, который возвращает некорректную информацию.

В предыдущем (некорректном) запросе мы запросили у браузера SQL два вида информации из таблицы `invoices`. Сначала — перечислить все города в поле `BillingCity`. Затем — вычислить среднее значение поля `Total`. Результат выполнения первого запроса — многострочный ответ, а результат второго — однострочный ответ. Другими словами, мы указываем браузеру отображать одновременно как агрегатные, так и неагрегатные поля. Мы не получили необходимую информацию, так как неправильно сформулировали вопрос.

Исправить эту проблему можно, добавив в запрос условие `GROUP BY` следующим образом:

```
SELECT
   BillingCity,
   AVG(Total)
FROM
   invoices
GROUP BY
   BillingCity
ORDER BY
   BillingCity
```

	BillingCity	AVG(Total)
1	Amsterdam	5.802857143
2	Bangalore	6.106666667
3	Berlin	5.374285714
4	Bordeaux	5.66
5	Boston	5.374285714
...	53 строки получено за 2 мс	

Рис. 107

Анализируя выполнение запроса (рис. 107), мы видим, что все города, где были выставлены счета, в нашем наборе результатов теперь появляются один раз и для каждого города отображается среднее значение счета.

Практическое задание

✶ В данный запрос добавьте функцию `ROUND()`, чтобы округлить средние значения до двух десятичных знаков.

Использование условий WHERE и HAVING со сгруппированными запросами

Добавление критериев в сгруппированный запрос работает так же, как и с другими, уже знакомыми нам запросами. Использование условия `WHERE` позволяет нам добавлять новые критерии. В примере ниже критерии добавляются для неагрегатного поля `BillingCity`.

```
SELECT
   BillingCity,
   AVG(Total)
FROM
   invoices
WHERE
   BillingCity LIKE 'L%'
GROUP BY
   BillingCity
ORDER BY
   BillingCity
```

	BillingCity	AVG(Total)
1	Lisbon	5.66
2	London	5.374285714
3	Lyon	5.374285714
	3 строки получено за 1 мс	

Рис. 108

НАПОМИНАНИЕ

Неагрегатное поле – это просто поле в условии `SELECT`, которое вызывается без агрегатной функции.

Практическое задание

✶ Сколько городов, где были выставлены счета, начинаются с буквы *L*?

В последнем примере мы добавили критерии в неагрегатное поле. Может возникнуть необходимость использовать критерии для агрегированных полей, например `AVG(Total)`. Скажем, когда нам надо найти все средние значения,

меньшие 20. Мы могли бы попытаться ответить на этот вопрос с помощью условия WHERE, но существует одна проблема.

ВНИМАНИЕ

Следующий оператор SQL содержит ошибку. Но важно видеть, что критерии, созданные в условии WHERE, не работают с агрегатными данными.

```
SELECT
   BillingCity,
   AVG(Total)
FROM
   Invoices
WHERE
   AVG(Total) > 5
GROUP BY
   BillingCity
ORDER BY
   BillingCity
```

При выполнении запроса возникнет следующее сообщение об ошибке:

```
Misuse of aggregate: AVG():
```

(Неправильное использование агрегата: AVG())

Это сообщение об ошибке информирует нас, что для создания условия на основе агрегатной функции (по крайней мере, в данном случае) мы не можем использовать условие WHERE. В данном случае условие WHERE может указывать только, какую информацию извлекать из полей, указанных в условии SELECT. Если необходима дополнительная фильтрация на основе агрегатных функций, необходимо включить вторичную фильтрацию, известную как условие HAVING.

Условие HAVING всегда следует после условия GROUP BY. Измененный запрос теперь выглядит следующим образом:

```
SELECT
   BillingCity,
   AVG(Total)
FROM
   invoices
GROUP BY
   BillingCity
HAVING
   AVG(Total) > 5
ORDER BY
   BillingCity
```

	BillingCity	AVG(Total)
1	Amsterdam	5.802857143
2	Bangalore	6.106666667
3	Berlin	5.374285714
4	Bordeaux	5.66
5	Boston	5.374285714
...	53 строки получено за 1 мс	

Рис. 109

ПРИМЕЧАНИЕ

Условие HAVING позволяет фильтровать результат группировки, сделанной с помощью команды GROUP BY. Условие HAVING фильтрует агрегированные данные. Если вы попытаетесь использовать HAVING без условия GROUP BY, то получите сообщение об ошибке.

Условия WHERE и HAVING

Если кратко, то разница между условиями `WHERE` и `HAVING` заключается в том, что `WHERE` предназначено для фильтрации неагрегатных данных, а `HAVING` — для фильтрации результатов, содержащих агрегаты. Если более подробно, то два типа фильтрации возникают, когда в запрос включены как условие `WHERE`, так и условие `HAVING`. Условие `WHERE` указывает запросу, какую информацию следует исключить из таблицы, а затем, после фильтрации данных и применения к полям агрегатных функций, условие `HAVING` действует как дополнительный фильтр для агрегатных данных. Давайте повторим предыдущий запрос, но на этот раз выберем только города, начинающиеся с буквы *B*, а затем из этого списка отфильтруем счета, среднее значение которых больше пяти (рис. 110).

```
SELECT
   BillingCity,
   AVG(Total)
FROM
   invoices
WHERE
   BillingCity LIKE 'B%'
GROUP BY
   BillingCity
HAVING
   AVG(Total) > 5
ORDER BY
   BillingCity
```

В запросе мы выполнили ту же задачу, но на этот раз добавили условие WHERE для фильтрации результатов только по городам, начинающимся с буквы *B*. Данный этап фильтрации выполняется до обработки условий HAVING и ORDER BY. Так как нам необходимо выполнить фильтрацию, прежде чем мы сможем группировать, то порядок условий фильтрации важен, а условие WHERE всегда предшествует HAVING.

	BillingCity	AVG(Total)
1	Bangalore	6.106666667
2	Berlin	5.374285714
3	Bordeaux	5.66
4	Boston	5.374285714
5	Brasília	5.374285714
6	Brussels	5.374285714
7	Budapest	6.517142857
8	Buenos Aires	5.374285714
	8 строк получено за 1 мс	

Рис. 110

Группировка по нескольким столбцам

В условиях GROUP BY можно одновременно указывать столько столбцов, сколько вам требуется. Предположим, необходимо получить более подробную разбивку средних значений счетов. Мы можем написать запрос так, чтобы агрегированные данные были сначала сгруппированы по странам, а затем по городам. В приведенном ниже примере мы добавим в условие GROUP BY еще одно поле, BillingCountry. Давайте посмотрим, как работает запрос.

```
SELECT
   BillingCountry,
   BillingCity,
   AVG(Total)
FROM
   invoices
GROUP BY
   BillingCountry, BillingCity
ORDER BY
   BillingCountry
```

На рис. 111 мы видим, что у нас имеется несколько записей для одной страны выставления счета, а отдельные города указаны в соседнем столбце. Группировка по нескольким столбцам может быть очень полезна, когда необходимо получить более детальную информацию.

НАПОМИНАНИЕ

Файлы базы данных могут содержать орфографические и/или ошибки, связанные с регистром.

	BillingCountry	BillingCity	AVG(Total)
1	Argentina	Buenos Aires	5.374285714
2	Australia	Sidney	5.374285714
3	Austria	Sidney	6.088571429
4	Belgium	Brussels	5.374285714
5	Brazil	Brasília	5.374285714
6	Brazil	Rio de Janeiro	5.374285714
7	Brazil	São José dos Campos	5.66
8	Brazil	São Paulo	5.374285714
9	Canada	Edmonton	5.374285714
10	Canada	Halifax	5.374285714
...	53 строки получено за 6 мс		

Рис. 111

Несколько заключительных слов о функциях

Цель этой главы — познакомить вас с функциями. Мы описали возможности некоторых функций, их способность превращать данные в информацию и решать практические задачи. Если бы пришлось рассматривать все функции в SQLite и иллюстрировать работу каждой из них, то эту книгу мне никогда не удалось бы закончить. К счастью, в интернете можно найти массу информации о функциях SQL и их использовании. Я всегда рекомендую студентам поискать информацию о функциях в интернете, чтобы увидеть разные примеры их использования. Надеюсь, что вы продолжите обучение самостоятельно и узнаете еще больше о том, как использовать эти мощные инструменты.

Контрольные вопросы

1. Создайте однострочный список рассылки для всех клиентов из США, включая полные имена, написанные заглавными буквами, и полные адреса с пятизначными почтовыми индексами, в следующем формате:

   FRANK HARRIS 1600 Amphitheatre Parkway, Mountain View, CA 94043

2. Каковы средние годовые продажи клиентам из США согласно имеющимся данным за все годы?
3. Каков общий объем продаж компании за все время?
4. Кто входит в десятку лучших клиентов с точки зрения совершенных ими покупок? *Подсказка*: чтобы ответить на этот вопрос, необходимо использовать соединение (глава 6).

Резюме

- Функции позволяют изменять, форматировать и выполнять вычисления с данными в таблицах.
- Запросы, содержащие числовые данные, можно обрабатывать с помощью различных арифметических операций и агрегатных функций.
- Запросы, содержащие текст, можно разделить, объединить, использовать для них заглавные буквы и т. д.
- После агрегирования данные можно дополнительно отсортировать с помощью условий `GROUP BY` и `HAVING`.
- Условие `HAVING` работает с агрегатными полями так же, как условие `WHERE` с неагрегатными.
- Условие `HAVING` можно использовать в запросе, только если имеется условие `GROUP BY`.
- Условие `GROUP BY` можно использовать с несколькими полями, чтобы еще больше детализировать агрегатные данные.

ЧАСТЬ III

РАСШИРЕННЫЕ ВОЗМОЖНОСТИ ЯЗЫКА SQL

ГЛАВА 8

Подзапросы

Краткое содержание

- ✓ Подзапросы и агрегатные функции
- ✓ Подзапросы оператора `SELECT`
- ✓ Подзапросы условия `WHERE`
- ✓ Подзапросы без агрегатных функций
- ✓ Возврат нескольких значений
- ✓ Подзапросы и ключевое слово `DISTINCT`
- ✓ Контрольные вопросы

Подзапрос — это просто один запрос, вложенный в другой. Подзапрос обычно добавляется в условия `SELECT`, `FROM` или `WHERE`. Использование подзапросов полезно, когда запрос, который мы хотим создать, требует нескольких дополнительных шагов или вычислений для создания ожидаемого набора данных. Например, подзапросы очень полезны в сценариях, когда мы хотим просмотреть или сравнить запрос по условию, для вычисления которого требуется собственный запрос. Чтобы не писать один запрос, а затем копировать результаты в следующий, мы можем использовать подзапрос, выполняющий обе операции одновременно. Подзапросы также предоставляют нам другой метод одновременного доступа к данным из нескольких таблиц. Хотя использование подзапроса не так эффективно, как соединение (join), он позволяет нам выполнить вычисление в одной таблице, а затем использовать это вычисление в другой таблице. Начнем с использования подзапросов с агрегатными функциями.

Использование агрегатных функций в подзапросах

Рассмотрим простой оператор `SELECT`, который мы использовали в предыдущей главе для возврата среднего значения суммы счета из таблицы `invoices`:

```
SELECT
   ROUND(AVG(Total), 2) AS [Average Total]
FROM
   invoices
```

	Average Total (Итоговое среднее значение)
1	5.65
	1 строка получена за 1 мс

Рис. 112

НАПОМИНАНИЕ

Чтобы округлить значение функции AVG() до двух значащих цифр, добавим функцию ROUND().

Мы видим, что в примере запроса средняя сумма счета из таблицы `invoices` составляет $5,65. Предположим, компания sTunes попросила нас собрать данные обо всех счетах, сумма которых меньше этого среднего значения. Прежде всего, необходимо использовать оператор `SELECT`, который выводит отдельные поля счета (например, `InvoiceDate`, `BillingAddress`, `BillingCity` и, конечно же, `Total`). Затем отфильтруем полученные результаты, сравнив их с агрегатной функцией. Используем условие `WHERE`, чтобы сравнить поле `Total` с полем `AVG(Total)`. В предыдущей главе вы узнали, что попытка использования условия `WHERE Total<AVG(Total)` в условии `WHERE` для прямого сравнения приводит к ошибке `misuse of aggregate function` (неправильное использование агрегатной функции). Итак, нам надо взять указанный выше запрос и вставить его в другой запрос, который сортирует счета по итоговой сумме. И, к счастью, есть простой способ сделать это.

Начнем с базового оператора `SELECT`, а затем, используя круглые скобки `()`, вставим весь приведенный выше запрос в условие `WHERE`, заставляя его функционировать как подзапрос.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE Total <
(SELECT
   AVG(Total)
from
   invoices)
ORDER BY
   Total DESC
```

Запрос, заключенный в круглые скобки, называется *внутренним запросом*, он и станет частью условия `WHERE` нашего *внешнего запроса*.

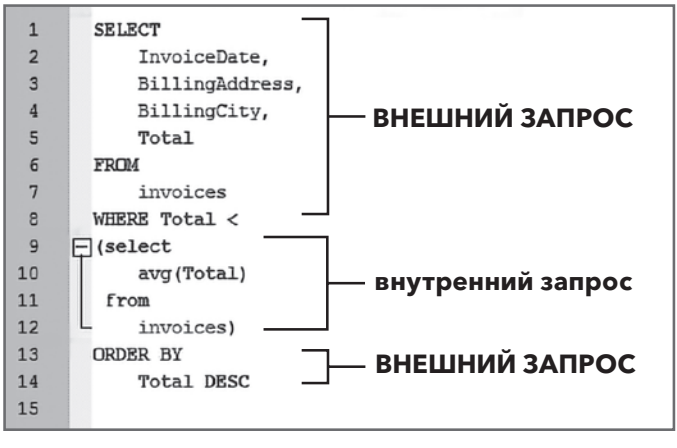


Рис. 113

На рис. 113 показано, что оператор `SELECT` и другие операторы во внешнем запросе написаны прописными, а все операторы и функции во внутреннем запросе — строчными буквами. Не существует универсального правила, каким образом следует писать операторы или функции. Я лично считаю, что указанный способ написания операторов и функций внешних запросов и подзапросов упрощает восприятие, поскольку помогает визуально различать внешние и внутренние операторы запроса.

Использование подзапроса в операторе SELECT

Если в операторе `SELECT` требуется добавить дополнительное действие (например, агрегатное вычисление), то для его выполнения нам понадобится подзапрос. В предыдущей главе, посвященной функциям, показано, что для отображения средних значений в счетах для разных городов мы использовали условие `GROUP BY`. Что будет, если для компании sTunes нам надо узнать показатели продаж в каждом отдельном городе и сравнить их со средними мировыми продажами? Один из способов ответить на это — написать запрос, который будет отображать средний объем продаж в каждом городе рядом со среднемировым показателем.

Запрос для отображения среднего объема продаж `BillingCity` идентичен запросу, который мы использовали в предыдущей главе, за одним исключением. Для расчета глобального среднего показателя мы включаем подзапрос в условие `SELECT`. Таким образом, мы можем сравнить два значения.

```
SELECT
   BillingCity,
AVG(Total) AS [City Average],
(SELECT
   avg(total)
from
   invoices) AS [Global Average]
FROM
   invoices
GROUP BY
   BillingCity
ORDER BY
   BillingCity
```

Результат этого запроса показывает, как продажи в каждом городе соотносятся со среднемировым уровнем.

	BillingCity	City Average	Global Average
1	Amsterdam	5.802857143	5.651941748
2	Bangalore	6.106666667	5.651941748
3	Berlin	5.374285714	5.651941748
4	Bordeaux	5.66	5.651941748
5	Boston	5.374285714	5.651941748
...	53 строки получено за 5 мс		

Рис. 114

Из рис. 114 видно, что значение для Global Average в каждой возвращаемой записи остается неизменным, что позволяет нам легко сравнивать средние итоговые суммы счетов по городам с мировым средним значением.

Практическое задание

- Измените запрос, используя функцию `ROUND()`, чтобы отображалось только два десятичных знака.

Использование подзапроса с условием WHERE

Иногда надо получить более подробный запрос в качестве подзапроса. Внешний запрос может содержать условие `WHERE`, которое, в свою очередь, содержит подзапрос с собственным условием `WHERE`. Хороший пример того, когда в подзапросе необходимо использовать условие `WHERE`, если требуется сравнить все поля с отдельным значением. Предположим, нас попросили найти самые большие продажи за весь период сбора данных (2009–2012 гг.) и проверить, имеются ли какие-либо итоговые суммы счетов за последний отчетный год (2013 г.), превышающие это значение. Чтобы ответить, сначала необходимо узнать самые большие продажи до 2013 года. Для этого воспользуемся функцией `MAX()`.

```
SELECT
   MAX(Total)
FROM
   invoices
WHERE
   InvoiceDate < '2013-01-01'
```

	MAX(Total)
1	23.86
	1 строка получена за 1 мс

Рис. 115

Теперь, когда нам известно это значение, мы заключим запрос в круглые скобки `()`, а затем добавим внешний запрос и вставим необходимые дополнительные поля.

```
SELECT
   InvoiceDate,
   BillingCity,
   Total
FROM
   invoices
WHERE
   InvoiceDate >= '2013-01-01' AND total >
   (select
         max(Total)
   from
         invoices
   where
         InvoiceDate < '2013-01-01')
```

Из запроса видно, что максимальный счет был выставлен 13 ноября 2013 года.

	InvoiceDate	BillingCity	Total
1	2013-11-13 00:00:00	Prague	25.86
	1 строка получена за 2 мс		

Рис. 116

Практическое задание

✶ Сколько счетов, значения которых превышали среднюю сумму счета, было зарегистрировано 1 января 2010 года или ранее?

Подзапросы без агрегатных функций

Подзапрос не всегда содержит агрегатную функцию. Следующий запрос отображает дату конкретной транзакции.

```
SELECT
   InvoiceDate
FROM
   invoices
WHERE
   InvoiceId = 251
```

	InvoiceDate
1	2012-01-09 00:00:00
	1 строка получена за 1 мс

Рис. 117

Если необходимо узнать, получены ли какие-либо другие счета после указанного выше счета, мы добавим подзапрос, заключенный в круглые скобки, а затем добавим внешний запрос.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity
FROM
   invoices
WHERE
   InvoiceDate >
(select
   InvoiceDate
from
   invoices
where
   InvoiceId = 251)
```

	InvoiceDate	BillingAddress	BillingCity
1	2012-01-22 00:00:00	Av. Paulista, 2022	São Paulo
2	2012-01-22 00:00:00	Qe 7 Bloco G	Brasília
3	2012-01-23 00:00:00	700 W Pender Street	Vancouver
4	2012-01-24 00:00:00	1 Infinite Loop	Cupertino
5	2012-01-27 00:00:00	319 N. Frances Street	Madison
...	161 строка получена за 8 мс		

Рис. 118

Возврат нескольких значений из подзапроса

До этого момента мы использовали только подзапросы для вычисления единственного значения, которое затем передается внешнему запросу. Также можно использовать подзапросы, возвращающие несколько записей. Допустим, руко-

водство компании sTunes хочет получить только три конкретных счета. Чтобы их выбрать, нужен следующий запрос:

```
SELECT
   InvoiceDate
FROM
   invoices
WHERE
   InvoiceId IN (251, 252, 255)
```

	InvoiceDate
1	2012-01-09 00:00:00
2	2012-01-22 00:00:00
3	2012-01-24 00:00:00
...	3 строки получены за 1 мс

Рис. 119

В предыдущем запросе для возврата трех дат из таблицы `invoices` используется условие `IN`: 2012-01-09, 2012-01-22 и 2012-01-24. Теперь предположим, что нам нужна информация о покупках за эти три дня. Если необходимо выбрать все счета за эти три дня, мы можем написать новый запрос или просто использовать предыдущий в качестве подзапроса, например:

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity
FROM
   invoices
WHERE
   InvoiceDate IN
(SELECT
   InvoiceDate
from
   invoices
where
   InvoiceId in (251, 252, 255))
```

Преобразование существующего запроса в подзапрос полезно, когда вы «играете» со своими данными. Чтобы еще больше сузить область поиска, этот метод позволяет повторно использовать существующий запрос и изменять его.

	InvoiceDate	BillingAddress	BillingCity
1	2012-01-09 00:00:00	Rua Dr. Falcão Filho, 155	São Paulo
2	2012-01-22 00:00:00	Av. Paulista, 2022	São Paulo
3	2012-01-22 00:00:00	Qe 7 Bloco G	Brasília
4	2012-01-24 00:00:00	1 Infinite Loop	Cupertino
...	4 строки получены за 2 мс		

Рис. 120

Подзапросы и условие DISTINCT

Другие примеры в этой главе показывают, что подзапросы очень полезны для сценариев, где надо просмотреть или сравнить запрос по условию, для вычисления которого требуется собственный запрос. Как вы уже знаете из главы 1, в каждой таблице есть одно уникальное поле, первичный ключ, содержащее уникальный номер для каждой записи, но другие поля могут содержать избыточную информацию. Для удобства работы с избыточной информацией стоит отфильтровать данные, чтобы они отображали только уникальные значения. В этом случае полезно условие `DISTINCT`. Лучше понять подзапросы и использование условия `DISTINCT` нам помогут таблицы `tracks` и `invoice_items`.

Таблица `invoice_items` показывает, какие треки какому счету соответствуют. Если мы создадим запрос, отображающий поля `InvoiceId` и `TrackId`, упорядоченные по `TrackId`, мы увидим, что определенные номера треков были заказаны несколько раз в разных счетах.

```
SELECT
   InvoiceId,
   TrackId
FROM
   invoice_items
ORDER BY
   TrackId
```

Так, треки № 2 и № 8 появляются в нескольких счетах, что означает, что они были заказаны несколько раз (рис. 121). Однако для трека № 7 счет отсутствует, поэтому мы можем сделать вывод, что никто не приобретал его. Руководство sTunes хочет знать о треках, которые не продаются. Нам необходимо найти таблицу, связывающую поля `TrackId` с `InvoiceId`. Для перечисления всех треков (по композитору и названию), которые не отображаются в таблице `invoice_items`, мы можем использовать подзапросы.

	InvoiceId	TrackId
1	108	1
2	1	2
3	214	2
4	319	3
5	1	4
6	108	5
7	2	6
8	2	8
9	214	8
10	108	9
...	2240 строк получено за 15 мс	

Рис. 121

	TrackId
1	1
2	2
3	3
4	4
5	5
6	6
7	8
8	9
9	10
10	12
...	1984 строки получено за 11 мс

Рис. 122

Если мы снова запустим тот же запрос, только на этот раз с ключевым словом `DISTINCT`, мы получим список только тех треков, которые появляются в счетах, без дубликатов.

```
SELECT
   DISTINCT TrackId
FROM
   invoice_items
ORDER BY
   TrackId
```

Из примера видно, что некоторые номера `TrackId` (например, № 7) не появляются ни в одном счете, но списки треков, которые появляются в нескольких счетах, сокращены до одного. Теперь нам необходимо написать запрос, перечисляющий все треки из таблицы `tracks`, которые не входят (`NOT IN`) в созданный первым запросом список.

```
SELECT
   TrackId,
   Composer,
   Name
FROM
   tracks
WHERE
   TrackId NOT IN
         (select distinct
               TrackId
         from
               invoice_items)
```

Итак, у нас есть список песен, которых не было ни в одном счете (рис. 123). Анализируя полученные результаты, мы видим, что трек № 7 находится в самой верхней части списка непродаваемых треков. Теперь отдел продаж sTunes имеет четкое представление о том, какие песни непопулярны.

	TrackId	Composer	Name
1	7	Angus Young, Malcolm Young, Brian Johnson	Let's Get It Up
2	11	Angus Young, Malcolm Young, Brian Johnson	C.O.D.
3	17	AC/DC	Let There Be Rock
4	18	AC/DC	Bad Boy Boogie
5	22	AC/DC	Whole Lotta Rosie
6	23	Steven Tyler, Joe Perry, Jack Blades, Tommy Shaw	Walk On Water
7	27	Steven Tyler, Joe Perry, Desmond Child	Dude (Looks Like A Lady)
8	29	Steven Tyler, Joe Perry, Taylor Rhodes	Cryin'
9	33	Steven Tyler, Jim Vallance	The Other Side
10	34	Steven Tyler, Joe Perry, Desmond Child	Crazy
...	1519 строк получено за 20 мс		

Рис. 123

Примеры подзапросов из этой главы не являются исчерпывающими. Существует еще множество вариантов использования подзапросов, и все они имеют общие

особенности. Подзапросы могут выполнять сложные многоступенчатые вычисления с помощью всего одного запроса. Они позволяют вычислить конкретное условие, а затем сравнить новый набор данных с тем же условием.

ПРИМЕЧАНИЕ

Подзапросы – это альтернативный способ взаимодействия таблиц, имеющих общие ключевые поля. Но если нам нужно много работать с обеими таблицами, то вместо подзапросов эффективнее создать join-соединение. Например, гораздо эффективнее создать соединение между полем TrackId и таблицей tracks (вместо использования подзапросов), чтобы отображать всю информацию рядом.

Контрольные вопросы

1. Сколько счетов превышают среднюю сумму счетов, выставленных в 2010 году?
2. Какие клиенты получили эти счета?
3. Сколько клиентов живут в США?

Резюме

- Подзапросы позволяют выполнять в одном запросе несколько операторов SQL.
- Подзапросы состоят из двух или более отдельных операторов SQL, которые образуют внутренние и внешние запросы.
- Подзапросы обычно используются для сравнения существующих данных с данными, полученными с помощью агрегатов или других функций.
- Условие `DISTINCT` позволяет игнорировать избыточные данные в записях и искать только уникальные значения.

ГЛАВА 9

Представления

Краткое содержание

- ✓ Создание представлений
- ✓ Изменение существующих представлений
- ✓ Представления и соединения
- ✓ Удаление представлений
- ✓ Контрольные вопросы

Представление является виртуальной таблицей. Это просто сохраненный SQL-запрос, который может выполняться многократно или на него (как подзапрос) могут ссылаться другие запросы. Представления полезны, если постоянно требуется один и тот же запрос, особенно когда он сложен. Предположим, руководство компании sTunes запрашивает одни и те же данные о продажах каждую неделю или квартал. Значит, имеет смысл подготовить представление — запрос на искомую информацию. Рассмотрим варианты использования представлений в SQL.

Работа с представлениями

Все операторы SQL, показанные в предыдущих главах, можно сохранить и использовать повторно, когда нам потребуется создать представление. Рассмотрим запрос, показанный в начале главы 8.

```
SELECT
   ROUND(AVG(Total), 2) AS [Average Total]
FROM
   invoices
```

Мы можем преобразовать этот оператор в представление, добавив над верхней строкой запроса команду `CREATE VIEW V_AvgTotal AS`:

```
CREATE VIEW V_ AvgTotal AS
SELECT
   ROUND(AVG(Total), 2) AS [Average Total]
FROM
   invoices
```

Итак, мы создали представление `V_AvgTotal`.

ПРИМЕЧАНИЕ

При именовании представлений используется V_ в начале названия. После символа подчеркивания идет краткое описание того, как работает представление, при этом при необходимости используются дополнительные символы подчеркивания. Также можно назвать это представление V_AvgTotal_Rounded. Вы можете выбрать другое название.

При выполнении этого оператора мы получим следующее сообщение: `Query executed successfully CREATE VIEW V_AvgTotal AS` (Запрос успешно выполнен: `CREATE VIEW V_AvgTotal AS`). Представление `V_AvgTotal` можно увидеть в разделе Views (Представления) на вкладке Database Structure (Структура базы данных) в DB Browser.

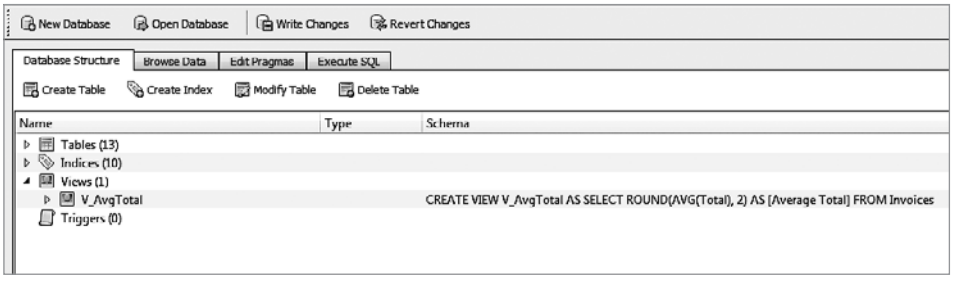


Рис. 124

Теперь, когда представление создано, мы можем выполнить несколько простых задач, просто щелкнув на нем правой кнопкой мыши. После этого появится окно меню (рис. 125).

Если выбрать пункт меню Browse Table (Просмотр таблицы) и перейти на соседнюю вкладку Browse Data (Просмотр данных), то можно просмотреть

содержимое представления (как для любой таблицы базы данных). Из этого меню мы также можем удалить представление или сгенерировать копию кода (который ввели ранее для создания представления).

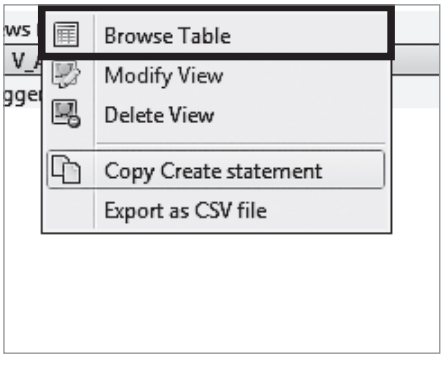


Рис. 125

ПРИМЕЧАНИЕ

На рис. 125 пункт меню Modify View (Изменить представление) на момент публикации неактивен и недоступен. Изменение существующих представлений не поддерживается данной версией DB Browser. Однако в других реализациях SQL, таких как SQL Server, можно изменять существующие представления. Позже в этой главе мы объясним, как изменить представление в SQLite.

Использование представлений

Использование представлений в базах данных полезно по ряду причин, но в первую очередь из-за удобства. Если вам постоянно нужен один и тот же запрос или вы постоянно ссылаетесь на конкретное соединение, которое показывает взаимодействие двух таблиц, такой запрос удобно сохранить как представление, и тогда он будет доступен при необходимости в любое время. Кроме того, когда запрос сохраняется как представление, его можно вызвать как подзапрос, выбрав имя представления.

Рассмотрим представление `V_AvgTotal`, созданное в начале этой главы. В главе 8 для сравнения итоговых сумм счетов со средней суммой счета мы использовали функцию расчета среднего значения в качестве подзапроса. Теперь, чтобы не писать полный подзапрос расчета среднего значения, мы можем написать наш запрос следующим образом:

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE Total <
   (SELECT
         *
   from
         V_AvgTotal)
ORDER BY
   Total DESC
```

ПРИМЕЧАНИЕ

Хотя представление – это полноценный SQL-запрос, мы все равно должны ссылаться на него в операторе SELECT, если будем использовать представление вместе с подзапросами. В операторах SELECT можно использовать символ *, чтобы представление, на которое мы ссылаемся, возвращало все строки. В рассматриваемом примере строка только одна – агрегированная сумма.

Если какой-либо запрос часто используется в качестве подзапроса, то, сохранив его в качестве представления, мы можем упростить код и сделать более понятными для других наши действия. Кто угодно, например наш коллега, может просто перейти на вкладку Database Structure (Структура базы данных) и изучить, как работают наши представления. Использование представлений сокращает время написания запроса, особенно когда запросы становятся длинными и сложными.

Изменения представлений

Как мы упоминали ранее, текущая версия DB Browser (на момент написания этой книги) не поддерживает изменение существующих представлений. В качестве альтернативы в SQLite необходимо создать новое представление и присвоить ему новое имя или удалить существующее представление. Чтобы изменить представление в SQLite, необходимо перейти на вкладку Execute SQL (Выполнить SQL-запрос), щелкнуть правой кнопкой мыши на представлении и скопировать оператор `CREATE VIEW`. Затем можно вставить результаты

на вкладку Execute SQL (Выполнить SQL-запрос), внести изменения и снова запустить оператор на выполнение.

ПРИМЕЧАНИЕ

При повторном запуске представления, если исходное представление все еще сохранено в разделе Views (Представления), необходимо переименовать новое представление (или удалить существующее). Иначе появится сообщение об ошибке, указывающее, что представление уже существует. Имя представления должно быть уникальным.

Практические задания

- Измените представление `V_AvgTotal`, чтобы в нем не было функции `ROUND()`.
- Из главы 8 выберите другой пример подзапроса, преобразуйте часть подзапроса в представление и запустите его снова.

Соединенные представления

Представления очень полезны для хранения длинных или сложных запросов. В главе 6 мы рассказывали о соединениях. Соединения помогают создавать связи между таблицами, обычно это довольно длинные запросы, поэтому их можно сохранить в качестве представлений, чтобы не вводить заново. В последней главе, посвященной подзапросам, мы использовали вместе таблицы `invoice_items` и `tracks`, чтобы выяснить, какие песни из таблицы `tracks` никогда не были заказаны. Нам бы пригодилось представление, связывающее эти две таблицы. На него можно было бы ссылаться в подзапросе либо просто сохранить как представление, чтобы использовать при необходимости.

Чтобы создать представление для этих двух таблиц, сначала необходимо решить, какое соединение мы хотим использовать. Поскольку мы ищем коррелирующие поля, то будем использовать условие `INNER JOIN` для таблиц `invoice_items` и `tracks`.

```
SELECT
   ii.InvoiceId,
   ii.UnitPrice,
   ii.Quantity,
   t.Name,
```

```
   t.Composer,
   t.Milliseconds
FROM
   invoice_items ii
INNER JOIN
   tracks t
ON ii.TrackId = t.TrackId
```

НАПОМИНАНИЕ

При создании соединения для каждой таблицы мы используем короткие псевдонимы, а затем, выявив общее поле, связываем таблицы друг с другом. В нашем случае зададим t для таблицы tracks и ii для таблицы invoice_items, так как i уже используется для таблицы invoices.

Теперь, когда соединение создано, мы можем добавить еще одну строку в начало запроса, чтобы сохранить его как представление.

```
CREATE VIEW V_Tracks_InvoiceItems AS
SELECT
   ii.InvoiceId,
   ii.UnitPrice,
   ii.Quantity,
   t.Name,
   t.Composer,
   t.Milliseconds
FROM
   invoice_items ii
INNER JOIN
   tracks t
ON ii.TrackId = t.TrackId
```

Теперь возьмем из главы 6 соединение, объединяющее таблицы `invoices`, `customers` и `employees`, и сохраним как представление.

```
CREATE VIEW V_inv_cus_emp AS
SELECT
   i.InvoiceId,
   i.InvoiceDate,
   i.Total,
   i.CustomerId,
   c.FirstName,
   c.LastName,
   c.SupportRepId,
   e.EmployeeId,
```

```
   e.LastName,
   e.FirstName,
   e.Title
FROM
   invoices AS i
INNER JOIN
   customers AS c
ON
   i.CustomerId = c.CustomerId
INNER JOIN
   employees AS e
ON
   e.EmployeeId = c.SupportRepId
ORDER BY
   InvoiceDate
```

ПРИМЕЧАНИЕ

В главе 6 мы немного изменили это соединение. Из условия SELECT мы удалили символ *. Смысл представлений – показать именно то, что необходимо. Таким образом, мы включили только необходимые поля из таблиц invoices, customers и employees.

Мы видим, что оба этих соединения сохранены как представления на вкладке Database Structure (Структура базы данных) (рис. 126).

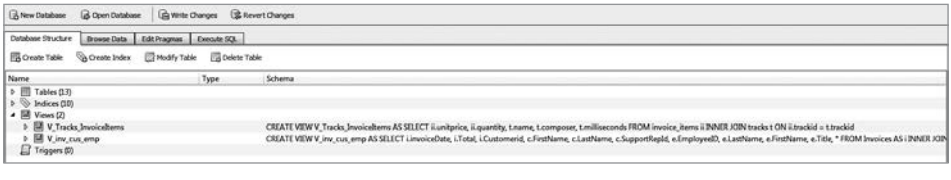


Рис. 126

Теперь, когда у нас имеются эти два представления, мы используем их во внутреннем соединении:

```
SELECT *
FROM
   V_Tracks_InvoiceItems ii
INNER JOIN
   V_inv_cus_emp ice
ON
   ii.InvoiceId = ice.InvoiceId
```

С помощью приведенного выше запроса, объединяющего пять таблиц, мы можем узнать, какие треки были проданы каждым сотрудником и какому клиенту. Агрегируя данные, мы также определим, какой трек продавался лучше всего, какой общий доход был получен от продажи трека, а также информацию о сотрудниках, осуществивших продажи. Теперь мы можем при желании сохранить это объединение как представление.

ПРИМЕЧАНИЕ

Соединенные представления будут работать только в том случае, если ключевые поля, общие для всех этих таблиц, были включены в первоначальное соединение.

Удаление представлений с помощью оператора DROP

Ранее мы уже описали один способ удаления представления — щелчок правой кнопкой мыши по имени представления на вкладке Database Structure (Структура базы данных). Представление также можно удалить с помощью команды `DROP`. Это выглядит следующим образом:

```
DROP VIEW
   V_AvgTotal
```

Она удалит представление `V_AvgTotal`. Из базы данных удаляется только представление — синтаксис `DROP VIEW` не влияет на данные.

ПРИМЕЧАНИЕ

Если вы удалите представление, на которое ссылаются другие представления, эти представления больше не будут работать.

ВНИМАНИЕ

DROP – это команда для удаления любых элементов. Она может также удалить из базы данных таблицу. В следующей главе вы изучите инструменты, с помощью которых можно редактировать или удалять данные без возможности восстановления. Как вы уже знаете, представление также можно удалить из базы данных, щелкнув по нему правой кнопкой мыши и выбрав опцию Remove View (Удалить представление).

Контрольные вопросы

Для этого задания используем запрос из главы 8:

```
SELECT
   BillingCity,
   AVG(Total) AS [City Average],
   (SELECT
         avg(total)
   from
         invoices) AS [Global Average]
FROM
   invoices
GROUP BY
   BillingCity
ORDER BY
   BillingCity
```

1. Из запроса `SELECT` возьмите внутренний запрос и создайте из него представление. Сохраните представление с именем `V_GlobalAverage`.
2. Удалите подзапрос из приведенного выше кода и замените его вновь созданным представлением `V_GlobalAverage`.
3. Сохраните этот новый запрос как представление с именем `V_CityAvgVsGlobalAvg`.
4. Удалите представление `V_GlobalAverage`. Как будет работать `V_CityAvgVsGlobalAvg`?

Резюме

- Представления — это виртуальные запросы, созданные с помощью SQL, на которые можно ссылаться в других запросах.
- Представления создаются посредством добавления в начало запроса следующего условия: `CREATE VIEW V_VIEWNAME AS`.
- Представления особенно полезны при написании длинных запросов.
- Представления можно изменять и удалять с помощью функций из вашей версии СУБД или с помощью команд SQL.

ГЛАВА 10

DML — язык управления данными

Краткое содержание

- ✓ Роль DML
- ✓ Вставка данных
- ✓ Обновление данных
- ✓ Удаление данных
- ✓ Контрольные вопросы

Все операторы SQL, которые вы уже изучили, использовались для извлечения информации из базы данных или создания производных данных на основе имеющихся значений. Эта глава познакомит вас с *языком управления данными (DML)* и с операторами SQL, которые используются для изменения данных, хранящихся в таблицах базы данных.

ВНИМАНИЕ

С этими командами лучше попрактиковаться в «песочнице», например на нашей учебной базе данных. Использование DML в рабочей базе с реальными данными может привести к фатальным последствиям.

Чем различаются анализ данных и управление базами данных

До сих пор мы говорили о том, что основная цель SQL-запросов — извлечение существующих данных из базы данных и преобразование их в полезную ин-

формацию (на примере компании sTunes). Но, как мы упоминали во введении, SQL выполняет гораздо более сложные задачи, чем просто превращение данных в значимую информацию. Существуют такие специалисты, как разработчики баз данных и администраторы баз данных, контролирующие рост, улучшение и управление базой данных компании. Функции этих специалистов различаются в зависимости от компании и реализации базы данных. Даже опытные пользователи SQL расходятся во мнениях о том, является ли язык DML (Data Manipulation Language — язык управления данными) отдельной областью изучения или его следует изучать совместно с операторами SQL, предназначенными только для извлечения информации.

НА МОЙ ВЗГЛЯД,

новичкам могут быть непонятны различия в ролях тех, кто работает с базами данных. Язык DML отлично подходит для администрирования и разработки баз данных. Если компания небольшая и в ней используется только одна база данных, роли аналитика, разработчика и администратора могут быть назначены одному человеку, и этим человеком можете быть вы! Так что даже если ваша основная цель – просто научиться писать необходимые запросы, важно понять работу DML.

Данными в РСУБД можно управлять с помощью следующих DML-команд: `INSERT`, `UPDATE` и `DELETE`. Как следует из названий, эти команды* могут использоваться для добавления, изменения и удаления данных из таблиц в базе данных. Для нашей базы sTunes мы продемонстрируем, как выполнить задание от руководства компании на добавление новых исполнителей к нашей музыкальной базе, добавление новых записей, а затем их удаление.

ПРИМЕЧАНИЕ

В DB Browser при внесении любых изменений в базу данных sTunes с использованием DML отображается сообщение с предложением сохранить изменения или закрыть файл базы данных без сохранения изменений. Вы можете сделать копию исходного файла базы данных, чтобы попрактиковаться в работе с базой данных, не опасаясь потерять оригинал.

* INSERT, UPDATE и DELETE могут называться как командами, так и операторами. — *Примеч. ред.*

Добавление данных в БД

Команда INSERT используется для вставки новых записей в таблицу. Существует несколько способов написать команду INSERT. Один из способов — использовать INSERT INTO и указать необходимое поле. Допустим, компания sTunes расширяет свой ассортимент и хочет, чтобы мы добавили в таблицу artists еще нескольких исполнителей.

Рассмотрим, как добавить новую запись в таблицу artists с помощью команды INSERT.

```
INSERT INTO
artists (Name)
VALUES ('Bob Marley')
```

Эта вставка состоит из трех элементов: таблицы, поля и значения. Таблица artists указана после команды INSERT INTO, где названа таблица, которую необходимо изменить. За ним следует имя поля, заключенное в круглые скобки (). В данном случае мы хотим добавить нового исполнителя, добавив его имя в поле Name. Затем следует ключевое слово VALUES, после которого в круглых скобках указано значение, которое необходимо добавить в таблицу artists. В нашем примере это Bob Marley. Bob Marley — это текстовое значение, поэтому оно заключено в одинарные кавычки.

НАПОМИНАНИЕ

Мы можем проанализировать тип данных поля Name в таблице artists, просмотрев вкладку Database Structure (Структура базы данных). В нашем случае тип данных – NVARCHAR(120), или символьный тип данных с установленным ограничением в 120 символов.

Практические задания

- Проанализируйте результат выполнения команды INSERT.
- Напишите оператор SELECT, чтобы найти новое добавленное значение.
- В таблицу artists добавьте значение Peter Tosh.
- Какой ID у добавленной записи Bob Marley?

ПРИМЕЧАНИЕ

В таблице artists имеется второй столбец – ArtistId. Его не надо указывать в команде INSERT, так как это столбец с автоматическим приращением. Это означает, что в столбце автоматически создается новый номер для новых добавляемых записей.

Еще один способ добавить значения в таблицу — перечислить значения последовательно по имени поля. В этом случае не нужно указывать столбцы, в которых должны храниться данные. Однако в этом случае при написании команды `INSERT` необходимо быть внимательным, чтобы гарантировать, что порядок, в котором данные указываются в команде `INSERT`, совпадает с порядком перечисления полей в таблице.

Рассмотрим пример добавления новой записи в таблицу `employees`.

```
INSERT INTO
employees
VALUES ('9', 'Martin', 'Ricky', 'Sales Support Agent', '2', '1975-02-
07', '2018-01-05', '123 Houston St', 'New York', 'NY', 'United States',
'11201', '(347) 525-8588', '', 'rmartin@gmail.com')
```

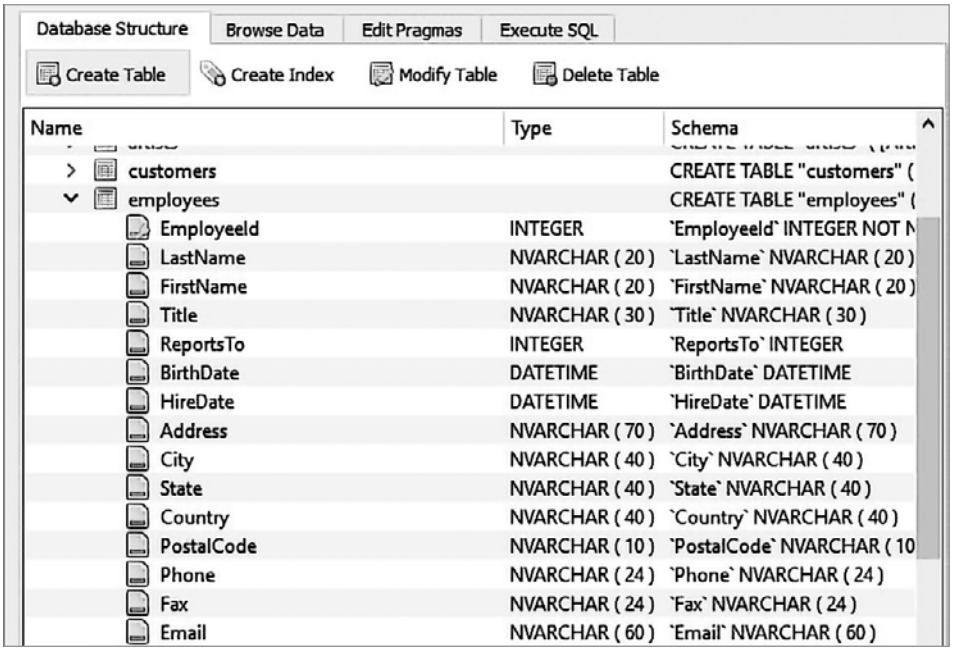


Рис. 127

Очень важно проследить, чтобы порядок, в котором данные указываются в команде INSERT, соответствовал порядку следования полей в целевой таблице. Поэтому рекомендуется сначала изучить целевую таблицу, и обратите внимание на порядок, в котором появляются поля.

На вкладке Database Structure (Структура базы данных) мы видим структуру таблицы employees. При использовании команды INSERT поля от EmployeeId до Email должны быть указаны именно в таком порядке.

Практические задания

- Проанализируйте результат выполнения команды INSERT.
- У нового сотрудника, Ricky Martin, нет номера факса. Как это будет выражено в условии INSERT?
- Вставьте новую запись (сотрудника) в таблицу employees.
- Выполните запрос еще раз, но используйте существующий первичный ключ. Какое сообщение об ошибке вы получите?

ВНИМАНИЕ

При добавлении новых записей, в которых для какого-либо поля не существует значения, вам необходимо добавить пару пустых кавычек '' для этого несуществующего значения. В приведенном выше примере мы пропустили номер факса при помощи пары пустых кавычек в таблице employees.

Обновление данных и ключевое слово SET

Команда UPDATE используется для изменения существующих данных в таблице. Как правило, команда UPDATE применяется с условием WHERE. Условие WHERE необходимо для указания строк, которые будут обновлены. Без условия WHERE оператор UPDATE обновит все содержащиеся в таблице строки.

Следующий оператор обновляет запись о сотруднике, которую мы добавили в предыдущем разделе:

```
UPDATE
employees
SET PostalCode = '11202'
WHERE
   EmployeeId = 9
```

ВНИМАНИЕ

Если сотрудники еще не добавлены, выполнение этого кода приведет к ошибке `No Such Column` (Столбец не существует).

Анализируя предыдущий синтаксис, мы видим, что таблица `employees` указывается после ключевого слова `UPDATE`. Затем следует ключевое слово `SET`, и именно здесь указан столбец в таблице `employees`, который необходимо обновить. В нашем примере это поле `PostalCode`. За ним следует знак равенства `=` и новое значение, которое заменит старое значение. В нашем примере это значение '11202' (заключено в одинарные кавычки, так как почтовый индекс является строковым значением). После ключевого слова `SET` следует условие `WHERE`, которое позволяет нам точно указать, какую запись о сотруднике мы собираемся обновить. Запись `EmployeeId=9` означает, что обновляется только эта запись и никакая другая.

Практические задания

- Проанализируйте результат выполнения команды `UPDATE`.
- Составьте еще один оператор `UPDATE` и измените номер телефона для записи Ricky Martin.
- Сколько строк будет затронуто при выполнении оператора `UPDATE`, если в нем не будет условия `WHERE`?

ВНИМАНИЕ

Особое внимание следует уделить добавлению условия WHERE в UPDATE. Если условие WHERE не включено и в нем не указаны необходимые для изменения записи, вы можете обновить записи, которые вы не собирались обновлять, что может привести к серьезным проблемам.

Удаление данных

Оператор `DELETE` используется для удаления существующих записей в таблице. Этот оператор, как и `UPDATE`, обычно используется с условием `WHERE`. Если условие `WHERE` отсутствует, оператор `DELETE` удалит все строки в таблице.

ПРИМЕЧАНИЕ

Как и в случае с UPDATE, не стремитесь сразу выполнять оператор DELETE, сначала напишите оператор SELECT, используя те же таблицы и условие

WHERE. Таким образом вы сможете предварительно проанализировать, что вы собираетесь удалить. Как только оператор SELECT вернет ожидаемые записи, вы можете выполнить оператор DELETE.

Сначала напишем оператор `SELECT`, чтобы проверить данные, которые мы собираемся удалить.

```
SELECT * FROM
   employees
WHERE
   EmployeeId = 9
```

Рассмотрим пример удаления записи, обновленной в предыдущем разделе.

```
DELETE FROM
   employees
WHERE
   EmployeeId = 9
```

Синтаксис оператора `DELETE` начинается с двух ключевых слов, `DELETE FROM`, за которыми следует имя таблицы, откуда мы собираемся удалить запись. В нашем примере мы удаляем запись из таблицы `employees`. После имени таблицы следует условие `WHERE`, которое позволяет нам точно указать, какую запись о сотруднике необходимо удалить. Запись `EmployeeId=9` означает, что будет удалена только эта запись и никакая другая.

Практические задания

- ✶ Проанализируйте результат выполнения оператора `DELETE`.
- ✶ Составьте еще один оператор `DELETE` и удалите имена всех сотрудников, которые работают в службе поддержки клиентов.
- ✶ На сколько записей повлияло выполнение оператора `DELETE` из предыдущего задания?

ВНИМАНИЕ

Особое внимание следует уделить добавлению условия WHERE к оператору DELETE. Если условие WHERE не включено и в нем не указаны необходимые для удаления записи, вы можете удалить не те записи, что вызовет серьезные проблемы.

Контрольные вопросы

1. Добавьте в базу данных нового клиента.
2. Создайте счет для этого клиента.
3. Удалите этого клиента из базы данных.

Резюме

- Операторы, рассмотренные в этой главе, изменяют информацию в базах данных. Перед использованием операторов внимательно проверьте правильность и необходимость их использования.
- Данные в РСУБД управляются с помощью следующих DML-команд: `INSERT`, `UPDATE` и `DELETE`.
- Оператор `INSERT` используется для добавления новых записей в базу данных.
- Оператор `UPDATE` используется для изменения существующих записей.
- Оператор `DELETE` используется для удаления записей.

Заключение

Главное – задавать правильные вопросы

Если вы дочитали эту книгу до конца, то наверняка научились использовать язык структурированных запросов и теперь можете превращать обычные записи в базе данных в значимую информацию, которую можно использовать для принятия сложных решений. В этой книге я показал, как создавать эффективные запросы для реальных сценариев, и продемонстрировал методы, которые я считаю полезными и использую постоянно. Я по возможности старался избегать технических деталей и сложной жаргонной лексики. Для начинающих: я надеюсь, вам понравилась моя книга и мне удалось поделиться своими знаниями в SQL. Для тех, кто уже изучал SQL: хочется верить, что эта книга помогла вам упорядочить и пополнить свои знания. В заключение я хотел бы рассказать о еще нескольких практических выводах и уроках, которые я извлек за восемнадцать лет работы в сфере информационных технологий. Я надеюсь, что все это поможет вам выбрать правильное направление изучения SQL. Кроме того, здесь я отвечу на некоторые часто задаваемые вопросы и кратко расскажу о дальнейшем образовании и получении сертификата по SQL.

Как найти свой путь

Я надеюсь, что благодаря этой книге вы познакомились со множеством различных применений SQL. Некоторые вам пригодятся, а другие — нет. Когда восемнадцать лет назад я начал самостоятельно изучать удивительный мир баз данных, компьютерные технологии стремительно развивались. В то время я работал с Visual Basic и уже разрабатывал приложения (внешний интерфейс или визуальный интерфейс программных систем). В конце концов мне пришлось в эти приложения включить базы данных. Когда я начал изучать, как Microsoft Access позволяет наглядно представлять данные и связь между таблицами, я четко понял, как работают данные, и это расширило мое представление о базах данных. Со временем я понял, что данные — это основа моей работы. Более глубокое изучение мира больших данных стало для меня целью. Это может произойти и с вами, и я надеюсь, что это действительно случится. Я призываю своих студентов изучать все смежные дисциплины программирования, которые им могут понадобиться. Подумайте, какая специальность подходит вам лучше

всего! Ваш путь никогда не будет прямым. Все трудности, ошибки и промахи на этом пути помогут вам понять, что для вас истинно ценно и важно.

Выбор специальности для работы с базами данных

Хотя мы рассказывали в основном о работе аналитика баз данных (именно здесь пригодятся навыки в составлении запросов, применении операторов и получении ответов на практические вопросы), также весьма востребованы проектировщики баз данных. Вы когда-нибудь задумывались, кто решает, какие поля будут содержаться в той или иной таблице или как таблицы будут связаны друг с другом? Это работа проектировщика, или разработчика, баз данных. Например, в главе 6 мы рассказывали о предотвращении избыточности данных в нескольких таблицах. Управление базами данных также связано с ограничениями доступа, резервным копированием и аварийным восстановлением. Эти темы не рассматриваются в моей книге. Если вам интересно узнать больше о том, как создавать и обслуживать базы данных, то, возможно, вам стоит заняться проектированием баз данных.

Все ли дело в деньгах?

Часто, когда я обсуждаю со студентами возможность карьеры в области баз данных, они задают мне вопрос: какая профессия или сфера деятельности, связанная с базами данных, лучше всего оплачивается? С тех пор как я начал обучать людей, встречаясь с ними в том числе в неформальной обстановке, я понял, что многих мотивируют именно деньги. Это неплохо. Это разумная отправная точка, но не должно стать главной целью. Любое хорошо оплачиваемое занятие потребует от вас больших затрат времени и сил.

Однако, как показывает мой личный опыт, настанет момент, возможно, поздно вечером, когда вы окажетесь один на рабочем месте, в окружении лишь пустых кофейных чашек, и вы спросите себя: это действительно то, что я хотел? Это вообще того стоит? Будут моменты, когда вам начнет казаться, что ваши возможности на пределе, и вы засомневаетесь в правильности своего выбора. Путем проб и множества ошибок я обнаружил, что моя главная мотивация в эти тяжелые моменты — это страсть к обучению, и в первую очередь именно она и привела меня в эту отрасль. Я понял, что помощь другим, а точнее, наблюдение за тем, как лица моих учеников светятся, когда они достигают своей цели, приносит мне большее удовлетворение, чем карьера как таковая. Начать свой

бизнес и помогать людям, мотивировать их на учебу — это для меня огромное удовольствие. Но это мой опыт. Вас ждет ваш собственный путь, и он будет уникальным.

Не стоит гоняться за максимальной выгодой, просто каждый день напоминайте себе, что в первую очередь привлекло вас к этому занятию. Другими словами, какая конкретная область работы с базами данных вас привлекает или дает вам возможность постоянно развиваться? Существует способ монетизировать свои увлечения. Диапазон велик, поэтому независимо от того, интересуетесь ли вы медициной, спортом, путешествиями или политикой, в любой из этих областей вы сможете заниматься анализом данных.

SQL — это универсальный язык

Что необходимо знать, прежде чем использовать свои навыки с SQL другой версии? Часто этот вопрос задают мне студенты, ищущие сферу применения полученных навыков. Возможно, в требованиях о приеме на работу указано, что требуется опыт работы с конкретной реализацией SQL, например с SQL Server. И в каждой реализации базы данных имеются различия. Я призываю моих студентов не разочаровываться и не зацикливаться на знаниях конкретной версии SQL. Основные принципы, изложенные в этой книге, помогут вам работать с любой базой данных. Преимущества SQL в том, что это универсальный язык данных. Не беспокойтесь, если вам придется работать не с SQLite. Каждая реализация содержит такие же основные атрибуты, как и те, о которых вы узнали в этой книге. В каждой версии есть панель для ввода запросов, кнопка для выполнения операторов, форма обратной связи по вашему запросу и панель, где указано время обработки. Полученный результат всегда представлен в столбцах и строках. Структура реляционной базы данных, о которой вы узнали в главе 1, является отраслевым стандартом, и база данных будет организована в соответствии со стандартами независимо от того, работаете ли вы с Oracle, IBM, Microsoft SQL Server или любыми другими реализациями. Различные реализации (СУБД) — это как марки разных автомобилей. Кнопки, переключатели и подстаканники могут быть расположены немного в другом месте, но основные механизмы, тормоз и педаль газа, будут в одном и том же месте и работать одинаково во всех моделях. Цель SQL — выполнять запросы. Если вы настроитесь именно на составление запросов, все остальное станет на свои места и будет более понятным. Дополнительную информацию о реализациях SQL вы можете найти на сайте https://db-engines.com/en/ranking. На сайте вы также узнаете, какие реализации баз данных в настоящее время используются наиболее часто.

Смена карьеры

Если основная часть моего профессионального опыта связана с областью, далекой от SQL или программирования, как мне убедить работодателя дать мне шанс попробовать себя в качестве аналитика данных? Этот вопрос я слышу от многих моих студентов, у которых совершенно другой профессиональный опыт, и они только начинают свой путь в области анализа данных. Данные так или иначе связаны с вашей предыдущей профессией, хоть вы можете этого не замечать! Допустим, вы были водителем автобуса. Сначала вы подумаете, что такая профессия далека от профессии аналитика данных, но сейчас мы попробуем убедить вас в обратном. В течение каждой поездки вы видите, на каких остановках толпится больше людей, какие маршруты наиболее эффективны и т. д. В любой профессии всегда присутствуют данные. Вы можете начать собирать информацию о каждой поездке на автобусе и предоставлять эту информацию своему работодателю. Даже если вы далеки от технической сферы, вы можете использовать навыки, полученные благодаря этой книге. Собирайте данные о любой сфере жизни и формируйте базу! Если базы данных не существует, вы всегда можете ее создать!

Как продавать свои навыки

Зачастую студенты задают мне вопросы о том, как убедить их компанию предоставить им возможность доступа к базе данных, особенно если они начали работу в компании на другой должности. Студенты объясняют работодателю, что их запросы никоим образом не изменят базу данных, но те по-прежнему не хотят предоставлять им даже доступ для чтения. Как их убедить? Компания может ограничить доступ к базе данных. Ограничения могут возникать по разным причинам, но в таких случаях важно преподать правильно свои стремления и настаивать на ценности преобразования данных в значимую информацию, как мы показали в этой книге. Объясните, как ваша работа поможет компании сэкономить деньги. В большинстве случаев работа с базами данных разделяется на три этапа: разработка, тестирование и эксплуатация. Вы можете запросить доступ к среде разработки или, если база данных небольшая, запросить копию. Но существует и более важный вопрос. Не позволяйте препятствиям мешать вам в достижении вашей цели. Данные есть везде. Как сказал Боб Марли, «когда одна дверь закрыта, не сомневайтесь — другая открыта». Возможно, вы сможете найти аналогичную базу данных в той же области, где работает и ваша компания. Существуют общедоступные хранилища данных (например, data.gov для США), которые помогут вам практиковать ваши навыки.

Визуализация данных

Наука о данных развивается гораздо быстрее, чем SQL. Для студентов, желающих расширить свои навыки и выйти за рамки написания запросов в текстовом браузере SQL, визуализация данных считается хорошим вариантом. Получение практически значимой информации для многих людей бывает проблемой. Как вы уже знаете из использования DB Browser и реализации SQLite, SQL все еще существует в очень функциональном, но визуально непривлекательном мире скриптовых языков программирования.

Программное обеспечение для визуализации данных (также известное как программное обеспечение для бизнес-аналитики) — это быстроразвивающаяся область, которая сейчас очень востребована. Визуализация может дать новую жизнь простым операторам SQL. Из этой книги вы узнали, что, например, представления помогают сохранить часто используемые запросы и представить информацию в более организованном виде. Программное обеспечение для визуализации данных дает возможность добавлять гистограммы, сводные таблицы и другие способы отображения данных. Программное обеспечение для визуализации также позволяет отображать данные в реальном времени, поэтому ваши поля и производные визуальные элементы, такие как диаграммы или графики, автоматически обновляются по мере изменения данных. Этот способ более эффективен, чем старый способ копирования данных в программу для работы с электронными таблицами, такую как Excel. На рис. 128 перечислены не все, но наиболее популярные программные пакеты для визуализации данных.

Рис. 128

Советы для успешного собеседования

В интернете есть множество статей, где предлагается перечень технических тем, которые вы «должны изучить» перед собеседованием, устраиваясь на любую работу, связанную с SQL. Зачастую эти «десять основных технических навыков, которые вам необходимо знать» примерно так же полезны, как списки «десяти самых опасных животных, которые вас могут убить». Я не считаю, что

на технических собеседованиях надо просить соискателя запомнить синтаксис и применить его к очень конкретному сценарию. Я полагаю, что гораздо лучше выяснить, есть ли у него общее представление о предмете, решает ли он технические задачи и может ли описать этапы, необходимые для достижения желаемого результата. Если возможность карьерного роста зависит только от того, как вы способны запоминать информацию, это может показывать, как компания относится к созданию новых технических решений, и должно настораживать. Намного лучше, если работодатель будет больше заинтересован тем, как вы решаете проблемные вопросы, чем тем, как хорошо вы сможете запомнить синтаксис. Некоторые интервьюеры могут сосредоточиться на конкретном инструменте, таком как синтаксис представления, или могут попросить вас решить задачу с помощью определенного оператора SQL. Этот метод интервьюирования, на мой взгляд, недальновиден, он фокусируется на механической памяти. Гораздо важнее увидеть, насколько креативен потенциальный сотрудник при применении представления или другого SQL-решения для ответа на поставленный вопрос.

Сертификация по SQL

Существует множество различных программ сертификации по SQL и администрированию баз данных. Примеры наиболее распространенных сертификатов — это Microsoft Certified Solutions Associate (MCSA) и Microsoft Certified Solutions Expert (MCSE). Однако Microsoft — не единственный вариант. Существуют и другие платформы баз данных, такие как Oracle и IBM, которые также являются крупными игроками в сфере баз данных, предлагающими сертификаты. Действительно ли сертификаты необходимы? Сертификация — не единственный путь в вашей карьере в области SQL. Я верю, что вы можете получить столько же знаний, если не больше, когда станете практиком. Для меня способность использовать язык на практике более важна, чем сертификация. Если ваша компания использует IBM, пройдите сертификацию. Но если вы не уверены, просто практикуйтесь в решении реальных вопросов с любой необходимой реализацией SQL.

Напутственные слова

Я искренне надеюсь, что вам понравилась книга и что я смог вас увлечь этой темой. Дополнительную информацию о деятельности моей компании по визуализации данных и об учебных курсах, которые я предлагаю, вы можете найти на http://datadecided.com. Мне было приятно сопровождать вас в этом увлекательном путешествии в мире баз данных.

СКАЧИВАЙТЕ БЕСПЛАТНО!

- Ссылки для скачивания программного обеспечения SQL + инструкции
- Видеоуроки
- Операторы SQL. Справочное руководство
- Образец базы данных – внимательно читайте книгу!

ДВА СПОСОБА ДОСТУПА К ЦИФРОВЫМ РЕСУРСАМ

Чтобы получить доступ к цифровым ресурсам, используйте приложение камеры на своем мобильном телефоне, чтобы отсканировать QR-код, или перейдите по ссылке ниже.

SCAN ME

или

www.clydebankmedia.com/programming-tech-vault

VISIT URL

ПРИЛОЖЕНИЕ I

Контрольные вопросы и ответы на них

Глава 3. Контрольные вопросы

Используя вкладку Database Structure (Структура базы данных) и вкладку Browse Data (Просмотр данных), ответьте на следующие вопросы.

Вопрос 1. Сколько таблиц в нашей базе данных?

Решение. В DB Browser перейдите на вкладку Database Structure (Структура базы данных), где количество таблиц указано в скобках (). В нашей базе данных тринадцать таблиц.

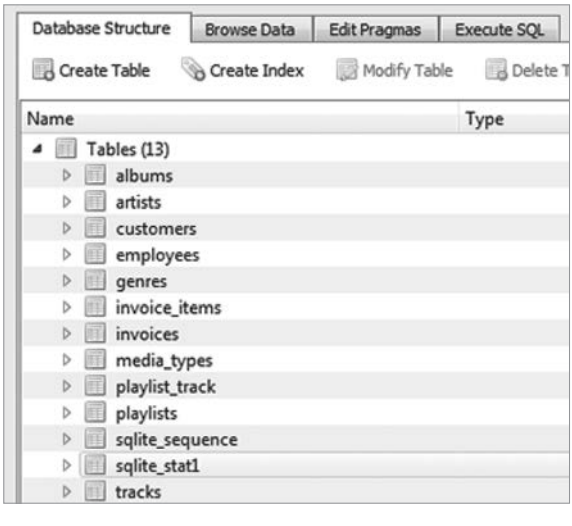


Рис. 129

Вопрос 2. Сколько полей в таблице `tracks`?

Решение. Для любой из перечисленных таблиц мы можем щелкнуть на расположенном справа маленьком треугольнике, чтобы отобразить столбцы этой таблицы.

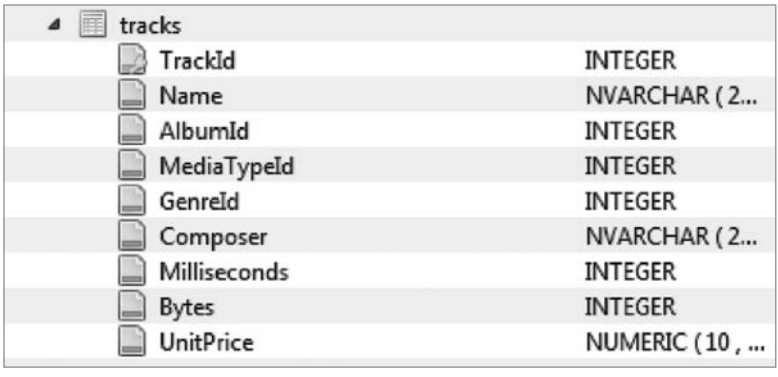


Рис. 130

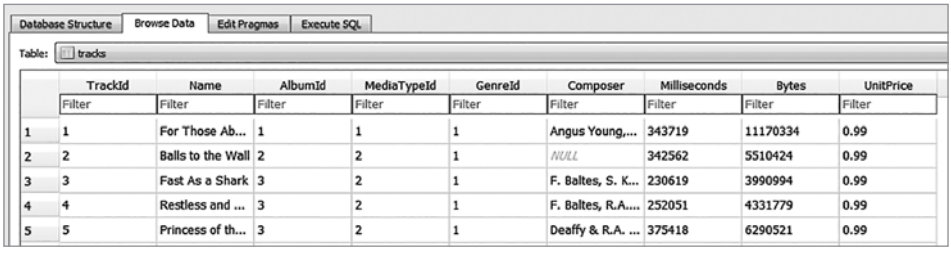
Из примера следует, что таблица `tracks` состоит из девяти столбцов.

Вопрос 3. Какие типы данных указаны в этой таблице?

Решение. На рис. 130 видно, что столбец `TrackId` принимает данные типа `INTEGER`, а столбец `Name` принимает данные типа `NVARCHAR`. Остальные столбцы также принимают типы `INTEGER` и `NVARCHAR` за исключением `UnitPrice`, который принимает тип данных `NUMERIC`.

Вопрос 4. Как выглядят данные в таблице?

Решение. Перейдите на вкладку Browse Data (Просмотр данных) и проанализируйте таблицу. Необходимо в раскрывающемся меню выбрать таблицу `tracks`. Анализ данных в таблице показывает, почему тип данных `INTEGER` используется для таких столбцов, как `TrackId` и `AlbumId`, в то время как символьный тип данных имеет больше смысла для столбцов `Name` и `Composer`. Наконец, для `UnitPrice` необходим тип данных с десятичными знаками, а целочисленного типа данных для этого столбца недостаточно.



	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	For Those Ab...	1	1	1	Angus Young,...	343719	11170334	0.99
2	2	Balls to the Wall	2	2	1	*NULL*	342562	5510424	0.99
3	3	Fast As a Shark	3	2	1	F. Baltes, S. K...	230619	3990994	0.99
4	4	Restless and ...	3	2	1	F. Baltes, R.A....	252051	4331779	0.99
5	5	Princess of th...	3	2	1	Deaffy & R.A. ...	375418	6290521	0.99

Рис. 131

Глава 4. Контрольные вопросы

Вопрос 1. Напишите запрос, чтобы узнать количество клиентов, фамилии которых начинаются с буквы *B*.

Решение. Чтобы ответить на этот вопрос, сначала надо написать запрос для отображения конкретной информации, которую мы ищем. В данном случае нас интересуют фамилии. Фамилии содержатся в таблице клиентов в поле `LastName`.

Следующий запрос отображает все фамилии.

```
SELECT
   LastName
FROM
   customers
```

В результате данного запроса отображаются все фамилии, но они не расположены в нужном порядке. Чтобы расположить их по алфавиту, мы можем использовать оператор `ORDER BY`. Обратите внимание, что нам не нужно указывать `A – Z`, так как по умолчанию данные отображаются в порядке возрастания. Если бы нам требовалось получить имена, начинающиеся с буквы `Z`, мы использовали бы оператор `DESC`.

```
SELECT
   LastName
FROM
   customers
ORDER BY
   LastName ASC
```

	LastName
1	Almeida
2	Barnett
3	Bernard
4	Brooks
5	Brown
6	Chase
7	Cunningham
8	Dubois
9	Fernandes
10	Francis
...	

Рис. 132

Теперь наши результаты расположены в алфавитном порядке, и мы можем видеть, что четыре записи начинаются с буквы *B*. Обратите внимание, что мы все еще посредством визуального наблюдения определяем, сколько записей начинается с буквы *B*. Другие способы мы рассмотрим ниже.

Вопрос 2. Какая компания при сортировке в порядке убывания появляется в верхней строке таблицы `customers`?

Решение. На этот раз мы ищем поле `Company`. Как мы говорили в предыдущем вопросе, все, что нам необходимо сделать, это изменить последнюю часть нашего запроса, чтобы указать порядок убывания.

```
SELECT
   Company
FROM
   customers
ORDER BY
   Company DESC
```

Получим следующий результат.

	Company
1	Woodstock Discos
2	Telus
3	Rogers Canada
4	Riotur
5	Microsoft Corporation
6	JetBrains s.r.o.
7	Google Inc.
8	Embraer - Empresa Brasileira de Aeronáutica S.A.
9	Banco do Brasil S.A.
10	Apple Inc.
...	

Рис. 133

Мы видим, что компания Woodstock Discos — первая в списке по убыванию.

Вопрос 3. Какое количество клиентов не указали почтовый индекс?

Решение. Мы могли бы ответить на этот вопрос, прокручивая данные в разделе Browse Data (Просмотр данных), но есть более эффективный способ. Используя

условие SELECT, мы можем перечислить все данные в порядке возрастания. Но на этот раз нам необходимо перечислить более одного столбца, чтобы мы могли видеть, каким именам клиентов не соответствуют данные почтового индекса. Таким образом, мы выберем поля FirstName, LastName и PostalCode, а затем отсортируем результаты по PostalCode.

```
SELECT
   FirstName,
   LastName,
   PostalCode
FROM
   customers
ORDER BY
   PostalCode
```

В результате отобразятся четыре записи, которые не имеют почтовых данных, что обозначено значением null в столбце PostalCode (рис. 134).

	FirstName	LastName	PostalCode
1	João	Fernandes	NULL
2	Madalena	Sampaio	NULL
3	Hugh	O'Reilly	NULL
4	Luís	Rojas	NULL
5	Stanisław	Wójcik	00-358
6	Lucas	Mancini	00192
7	Terhi	Hämäläinen	00530
8	Eduardo	Martins	01007-010
9	Alexandre	Rocha	01310-200
10	Bjørn	Hansen	0171
...			

Рис. 134

ПРИМЕЧАНИЕ

Если поля перечислить в порядке убывания, придется прокрутить вниз полосу прокрутки, чтобы увидеть нулевые значения.

Глава 5. Контрольные вопросы

Вопрос 1. Создайте запрос для таблицы `invoices`, включающий оператор `CASE`, который будет отмечать все продажи из США — страны, откуда выставлен счет, как Domestic Sales (Продажи на внутреннем рынке), а все другие продажи — как Foreign Sales (Продажи за рубежом). После оператора `END AS` создайте новое поле `SalesType`.

Решение. Чтобы отобразить эту информацию, необходимо применить фильтрацию, используя оператор `CASE`. Поскольку мы классифицируем наш оператор `CASE` по стране, где был выставлен счет, необходимо добавить это поле в условие `SELECT`.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   BillingCountry,
   Total,
   CASE
   WHEN BillingCountry = 'USA' THEN 'Domestic Sales'
   ELSE 'Foreign Sales'
   END AS SalesType
FROM
   invoices
```

	InvoiceDate	BillingAddress	BillingCity	BillingCountry	Total	PurchaseType
1	1/1/2009 0:00	Theodor-Heuss-Straße 34	Stuttgart	Germany	1.98	Foreign Sales
2	1/2/2009 0:00	Ullevålsveien 14	Oslo	Norway	3.96	Foreign Sales
3	1/3/2009 0:00	Grétrystraat 63	Brussels	Belgium	5.94	Foreign Sales
4	1/6/2009 0:00	8210 111 ST NW	Edmonton	Canada	8.91	Foreign Sales
5	1/11/2009 0:00	69 Salem Street	Boston	USA	13.86	Domestic Sales
6	1/19/2009 0:00	Berger Straße 10	Frankfurt	Germany	0.99	Foreign Sales
7	2/1/2009 0:00	Barbarossastraße 19	Berlin	Germany	1.98	Foreign Sales
8	2/1/2009 0:00	8, Rue Hanovre	Paris	France	1.98	Foreign Sales
9	2/2/2009 0:00	9, Place Louis Barthou	Bordeaux	France	3.96	Foreign Sales
10	2/3/2009 0:00	3 Chatham Street	Dublin	Ireland	5.94	Foreign Sales
...						

Рис. 135

Вопрос 2. Отсортируйте эти данные по новому полю `SalesType`.

Решение. Чтобы отобразить все внутренние продажи в одной группе и все зарубежные продажи в другой группе, необходимо к уже существующему запросу добавить условие `ORDER BY` (используя новое поле):

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   BillingCountry,
   Total,
   CASE
   WHEN BillingCountry = 'USA' THEN 'Domestic Sales'
   ELSE 'Foreign Sales'
   END AS SalesType
FROM
   invoices
ORDER BY
   SalesType
```

На рис. 136 показаны результаты данного запроса, где сначала отображаются Domestic Sales (Продажи на внутреннем рынке). Если вы запустите этот запрос и прокрутите вниз, вы увидите, что все страны, кроме США, отмечены как Foreign Sales (Продажи за рубежом).

	InvoiceDate	BillingAddress	BillingCity	BillingCountry	Total	SalesType
1	1/11/2009 0:00	69 Salem Street	Boston	USA	13.86	Domestic Sales
2	2/19/2009 0:00	1600 Amphitheatre Parkway	Mountain View	USA	0.99	Domestic Sales
3	3/4/2009 0:00	1 Microsoft Way	Redmond	USA	1.98	Domestic Sales
4	3/4/2009 0:00	1 Infinite Loop	Cupertino	USA	1.98	Domestic Sales
5	3/5/2009 0:00	801 W 4th Street	Reno	USA	3.96	Domestic Sales
6	3/6/2009 0:00	319 N. Frances Street	Madison	USA	5.94	Domestic Sales
7	4/14/2009 0:00	1 Infinite Loop	Cupertino	USA	13.86	Domestic Sales
8	6/6/2009 0:00	1 Microsoft Way	Redmond	USA	3.96	Domestic Sales
9	6/7/2009 0:00	801 W 4th Street	Reno	USA	5.94	Domestic Sales
10	6/10/2009 0:00	1033 N Park Ave	Tucson	USA	8.91	Domestic Sales
...						

Рис. 136

Вопрос 3. Сколько счетов от продаж на внутреннем рынке превышают сумму $15?

Решение. Чтобы включить числовые и текстовые параметры, в существующий запрос можно добавить условия `WHERE` и `AND`.

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   BillingCountry,
   Total,
   CASE
   WHEN BillingCountry = 'USA' THEN 'Domestic Sales'
   ELSE 'ForeignSales'
   END AS SalesType
FROM
   invoices
Where
   SalesType = «Domestic Sales» AND Total > 15
```

	InvoiceDate	BillingAddress	BillingCity	BillingCountry	Total	SalesType
1	3/21/2010 0:00	162 E Superior Street	Chicago	USA	15.86	Domestic Sales
2	5/29/2011 0:00	319 N. Frances Street	Madison	USA	18.86	Domestic Sales
3	8/5/2012 0:00	2211 W Berry Street	Fort Worth	USA	23.86	Domestic Sales

Рис. 137

Глава 6. Контрольные вопросы

Вопрос 1. Используя DB Browser и вкладку Browse Data (Просмотр данных) или ER-диаграмму (рис. 65), проанализируйте таблицу `tracks`. Определите, какие поля в этой таблице будут внешними ключами в другой таблице. На основании определенных вами внешних ключей определите, какие таблицы связаны с таблицей `tracks`.

Решение. Анализируя таблицу `tracks`, мы видим три поля с целочисленными значениями, которые могут быть внешними ключами.

Поля `AlbumId`, `MediaTypeId` и `GenreId` соответствуют таблицам `albums`, `media_types` и `genres` соответственно.

	TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	For Those Ab...	1	1	1	Angus Young,...	343719	11170334	0.99
2	2	Balls to the Wall	2	2	1	*NULL*	342562	5510424	0.99
3	3	Fast As a Shark	3	2	1	F. Baltes, S. K...	230619	3990994	0.99
4	4	Restless and ...	3	2	1	F. Baltes, R.A....	252051	4331779	0.99
5	5	Princess of th...	3	2	1	Deaffy & R.A. ...	375418	6290521	0.99
6	6	Put The Finge...	1	1	1	Angus Young,...	205662	6713451	0.99
7	7	Let's Get It Up	1	1	1	Angus Young,...	233926	7636561	0.99
8	8	Inject The Ve...	1	1	1	Angus Young,...	210834	6852860	0.99
9	9	Snowballed	1	1	1	Angus Young,...	203102	6599424	0.99
10	10	Evil Walks	1	1	1	Angus Young,...	263497	8611245	0.99

Рис. 138

Вопрос 2. Создайте внутреннее соединение между таблицами `albums` и `tracks` и отобразите названия альбомов и названия треков в едином наборе результатов.

Решение:

```
SELECT
   t.composer AS "Artist Name",
   a.title AS "Album Title",
   t.Name AS "Track Name"
FROM
   albums a
INNER JOIN
   tracks t
ON
   a.AlbumId = t.AlbumId
```

Вопрос 3. Создайте третье внутреннее соединение — с таблицей `genres`, которую вы нашли, отвечая на вопрос 1. Включите в ваш набор результатов поле `Name` из этой таблицы.

Решение:

```
SELECT
   g.name AS Genre,
   t.composer AS "Artist Name",
   a.title AS "Album Title",
   t.Name AS "Track Name"
FROM
   albums a
INNER JOIN
   tracks t
ON
   a.AlbumId = t.AlbumId
```

```
INNER JOIN
   genres g
ON
   g.GenreId = t.GenreId
```

Глава 7. Контрольные вопросы

Вопрос 1. Создайте однострочный список рассылки для всех клиентов из США, включая полные имена, написанные заглавными буквами, и полные адреса с пятизначными почтовыми индексами, в следующем формате:

FRANK HARRIS 1600 Amphitheatre Parkway, Mountain View, CA 94043

Решение. В приведенном выше формате имя и фамилия должны быть написаны в верхнем регистре, поэтому для этих двух полей мы воспользуемся функцией `UPPER()`. Для объединения остальных полей используем символ `||`, добавляя, где необходимо, пробелы и запятые.

```
SELECT
   UPPER(FirstName) || ' ' || UPPER(LastName) || ' '
   || Address || ', ' || City || ', ' || State || ' '
   || SUBSTR(PostalCode,1,5) AS [MailingAddress]
FROM
   customers
WHERE
   Country = 'USA'
```

	MailingAddress
1	FRANK HARRIS 1600 Amphitheatre Parkway, Mountain View, CA 94043
2	JACK SMITH 1 Microsoft Way, Redmond, WA 98052
3	MICHELLE BROOKS 627 Broadway, New York, NY 10012
4	TIM GOYER 1 Infinite Loop, Cupertino, CA 95014
5	DAN MILLER 541 Del Medio Avenue, Mountain View, CA 94040
1	KATHY CHASE 801 W 4th Street, Reno, NV 89503
2	HEATHER LEACOCK 120 S Orange Ave, Orlando, FL 32801
3	JOHN GORDON 69 Salem Street, Boston, MA 2113
4	FRANK RALSTON 162 E Superior Street, Chicago, IL 60611
5	VICTOR STEVENS 319 N. Frances Street, Madison, WI 53703
...	

Рис. 139

Вопрос 2. Каковы средние годовые продажи клиентам из США согласно имеющимся данным за все годы?

Решение. Если мы просто ищем агрегатную функцию для одной страны, то можем выбрать страну, где был выставлен счет, и среднее значение от общей суммы, используя условие `WHERE`, чтобы ограничить наши результаты для одной страны — США.

```
SELECT
   BillingCountry,
   AVG(Total)
FROM
   invoices
WHERE
   BillingCountry = 'USA'
```

	BillingCountry	AVG(Total)
1	USA	5.7479121

Рис. 140

НАПОМИНАНИЕ

Чтобы сократить количество возвращаемых знаков после запятой, мы можем использовать функцию ROUND() вне функции AVG().

Вопрос 3. Каков общий объем продаж компании за все время?

Решение. Поскольку в данном запросе задается общая сумма счетов, условие `SELECT` выглядит довольно просто:

```
SELECT
   SUM(Total)
FROM
   invoices
```

	SUM(Total)
1	2328.6

Рис. 141

Вопрос 4. Кто входит в десятку лучших клиентов с точки зрения совершенных ими покупок? *Подсказка*: чтобы ответить на этот вопрос, необходимо использовать соединение (глава 6).

Решение. Значение общей суммы найдено. Теперь необходимо отобразить первую десятку клиентов, приносящих наибольшую прибыль. Поскольку мы ищем данные из одной таблицы, которые соответствуют данным из другой таблицы во взаимно однозначном отношении, мы используем внутреннее соединение.

```
SELECT
   SUM(Total)AS [Revenue Total],
   c.FirstName,
   c.LastName
FROM
   invoices i
INNER JOIN
   customers c
ON
   i.CustomerId = c.CustomerId
GROUP BY c.CustomerId
ORDER BY SUM(Total) DESC
```

Глава 8. Контрольные вопросы

Вопрос 1. Сколько счетов превышает среднюю сумму счетов, выставленных в 2010 году?

Решение. Чтобы ответить на этот вопрос, необходимо решить две задачи. Во-первых, следует найти среднюю сумму счета-фактуры, сгенерированную в 2010 году. Во-вторых, необходимо сравнить это значение с каждым счетом в таблице, чтобы увидеть, сколько из них превышает среднюю стоимость счета-фактуры за 2010 год.

Сначала напишем следующий подзапрос:

```
select
   avg(total)
from
   invoices
where
   InvoiceDate between '2010-01-01' and '2010-12-31'
```

В результате выполнения данного запроса мы получим среднее значение $5,80. Теперь необходимо написать внешний запрос для выбора счетов, превышающих средний показатель за 2010 год.

```
SELECT
   InvoiceDate,
   Total
FROM
   invoices
WHERE
Total >

(SELECT
   avg(total)
from
   invoices
where
   InvoiceDate between '2010-01-01' and '2010-12-31')
ORDER BY
   Total DESC
```

	InvoiceDate	Total
1	11/13/2013 0:00	25.86
2	8/5/2012 0:00	23.86
3	2/18/2010 0:00	21.86
4	4/28/2011 0:00	21.86
5	1/18/2010 0:00	18.86
6	5/29/2011 0:00	18.86
7	1/13/2010 0:00	17.91
8	9/5/2012 0:00	16.86
9	10/6/2012 0:00	16.86
10	3/21/2010 0:00	15.86
...		

Рис. 142

В результате выполнения данного запроса получено 179 строк.

ПРИМЕЧАНИЕ

Если бы требовалось получить только фактическое количество возвращенных счетов-фактур, во внешнем запросе можно было бы изменить поле Total, указав COUNT(Total).

Вопрос 2. Какие клиенты получили эти счета?

Решение. Чтобы связать данные о клиентах из таблицы `customers` с таблицей `invoices`, необходимо использовать повторное объединение. Сам вопрос подразумевает однозначную связь между таблицей `customers` и таблицей `invoices`. Мы уже выбрали интересующие нас счета, поэтому теперь нам необходимо получить информацию о клиентах, которым были выставлены эти счета. При решении данного вопроса воспользуемся внутренним соединением. Это решение очень похоже на решение вопроса 1. Все, что мы добавили, — это раздел внутреннего соединения, поэтому у нас также имеется доступ к именам клиентов.

```
SELECT
   i.InvoiceDate,
   i.Total,
   c.FirstName,
   c.LastName
FROM
   invoices i
INNER JOIN
   customers c
ON
   i.CustomerId = c.CustomerId
WHERE
Total >

(SELECT
   avg(total)
from
   invoices
where
   InvoiceDate between '2010-01-01' and '2010-12-31')
ORDER BY
   Total DESC
```

Вопрос 3. Сколько клиентов живут в США?

Решение. Мы можем изменить решение вопроса 2, включив оператор `AND` в конец условия `WHERE` внешнего запроса.

```
SELECT
   InvoiceDate,
   Total,
   BillingCountry
FROM
   invoices
WHERE
Total >
```

```
(SELECT
   avg(total)
from
   invoices
where
   InvoiceDate between '2010-01-01' and '2010-12-31')
AND BillingCountry = 'USA'
ORDER BY
   Total DESC
```

	InvoiceDate	Total	BillingCountry
1	11/13/2013 0:00	25.86	USA
2	8/5/2012 0:00	23.86	USA
3	2/18/2010 0:00	21.86	USA
4	4/28/2011 0:00	21.86	USA
5	1/18/2010 0:00	18.86	USA
6	5/29/2011 0:00	18.86	USA
7	1/13/2010 0:00	17.91	USA
8	9/5/2012 0:00	16.86	USA
9	10/6/2012 0:00	16.86	USA
10	3/21/2010 0:00	15.86	USA
...			

Рис. 143

В результате выполнения данного запроса получено 40 строк.

ПРИМЕЧАНИЕ

При необходимости получения точного количества результатов можно использовать функцию SUM().

Глава 9. Контрольные вопросы

В данном разделе необходимо преобразовать запрос, в котором сравнивается средняя сумма счетов по городу со средним глобальным показателем, в представление.

```
SELECT
   BillingCity,
   AVG(Total) AS [City Average],
```

```
  (select
        avg(total)
   from
        invoices) AS [Global Average]
FROM
   invoices
GROUP BY
   BillingCity
ORDER BY
   BillingCity
```

Вопрос 1. Из запроса `SELECT` возьмите внутренний запрос и создайте из него представление. Сохраните его с именем `V_GlobalAverage`.

Если вы выполняли примеры из главы, возможно, вы уже сохранили функцию, рассчитывающую среднее значение, в качестве представления. Убедитесь, что новому представлению присвоено новое имя.

Решение. Во внутреннем запросе в первую строку добавим синтаксис представления.

```
CREATE VIEW V_GlobalAverage AS
select
   avg(total)
from
   invoices AS [Global Average]
```

Вопрос 2. Удалите подзапрос из приведенного выше кода и замените его вновь созданным представлением `V_GlobalAverage`.

Решение. При использовании представления в условии `SELECT` мы используем символ *.

```
SELECT
   BillingCity,
   AVG(Total) AS [City Average],
(select
   *
from
V_GlobalAverage) AS [Global Average]
FROM
   invoices
GROUP BY
   BillingCity
ORDER BY
   BillingCity
```

Вопрос 3. Сохраните этот новый запрос как представление с именем `V_CityAvgVsGlobalAvg`.

Решение. Копируем код из вопроса 2 и в верхнюю часть запроса добавляем оператор `CREATE VIEW`.

```
CREATE VIEW V_CityAvgVsGlobalAvg AS
SELECT
   BillingCity,
   AVG(Total) AS [City Average],
(select
   *
from
V_GlobalAverage) AS [Global Average]
FROM
   invoices
GROUP BY
   BillingCity
ORDER BY
   BillingCity
```

Вопрос 4. Удалите представление `V_GlobalAverage`. Как будет работать `V_CityAvgVsGlobalAvg`?

Решение. Чтобы удалить наше представление, воспользуемся условием `DROP VIEW`. Кроме того, на вкладке Database Structure (Структура базы данных) в DB Browser мы можем щелкнуть правой кнопкой мыши по представлению и удалить его.

```
DROP VIEW V_GlobalAverage
```

Теперь, чтобы увидеть, как это повлияет на наши предыдущие операторы, необходимо написать условие `SELECT` для выбора виртуальной таблицы.

```
V_CityAvgVsGlobalAvg
SELECT
   *
FROM
   V_CityAvgVsGlobalAvg
```

В результате мы получим следующее сообщение об ошибке:

```
no such table: main.V_GlobalAverage:
```

(таблица не существует: `main.V_GlobalAverage:`)

Глава 10. Контрольные вопросы

Вопрос 1. В таблицу `customers` добавьте новую запись.

Решение. Сначала необходимо добавить новую запись в таблицу `customers`. Клиент может существовать сам по себе, не будучи упомянутым в какой-либо другой таблице (если он еще не совершил покупку). Для начала вставьте запись в таблицу `customers`.

```
INSERT INTO
customers
VALUES ('60', 'New', 'Customer', '', '123 Day Street', 'New York',
'NY', 'USA', '11201', '(347) 525-8688', '', 'nc@gmail.com', '1');
```

ПРИМЕЧАНИЕ

Некоторые поля оставим пустыми, поставив рядом две одинарные кавычки. Запустим запрос на выполнение и проанализируем результаты.

```
SELECT
   *
FROM
   customers
WHERE
   FirstName = 'New'
```

ПРИМЕЧАНИЕ

Если для нового клиента вы использовали другое имя, измените это значение в запросе соответствующим образом.

	CustomerId	FirstName	LastName	Company	Address	City	State	Country	PostalCode	Phone	Fax	Email	SupportId
1	60	New	Customer		123 Day Street	New York	NY	USA	11201	(347)525-8688		nc@gmail.com	1

Рис. 144

Вопрос 2. Создайте счет для этого клиента.

Решение. Чтобы создать запись счета для нашего нового клиента, нам необходимо обратить особое внимание на поля в таблице `invoices`, которые соответствуют таблице `customers`. Например, в наших счетах используется тот же адрес, который отображается в таблице `customers`.

```
INSERT INTO
invoices
VALUES ('413', '60', '2019-10-04 00:00:00', '123 Day Street', 'New
York', 'NY', 'USA', '10201', '50.00')
```

Вопрос 3. Удалите этого клиента из базы данных.

Решение. Как мы упоминали в главе 10, рекомендуется просмотреть данные, которые необходимо удалить. В этом случае данные, которые мы удаляем, содержатся в двух таблицах, поэтому для просмотра всех необходимых нам данных мы воспользуемся условием `INNER JOIN`.

```
SELECT
   c.FirstName,
   c.LastName,
   i.Total,
   i.InvoiceId
FROM
   invoices i
INNER JOIN
   customers c
ON i.CustomerId = c.CustomerId
WHERE c.CustomerId = 60
```

Теперь, когда мы подтвердили данные, мы можем использовать оператор `DELETE`.

```
DELETE FROM
   invoices
WHERE CustomerId = 60

DELETE FROM
   customers
WHERE CustomerId = 60
```

ПРИЛОЖЕНИЕ II

Список ключевых слов SQL по главам

Глава 4. Ключевые слова

SELECT, AS, FROM, ORDER BY, ASC, DESC, LIMIT

```
/*
Многострочные комментарии начинаются с сочетания символов /*
и заканчиваются символами */. Как правило, блок комментариев соответствует
следующему формату:

CREATED BY: <ФИО>
CREATED ON: <дата>
DESCRIPTION: <краткое описание, что делает запрос>
*/

-- В данном примере используются однострочные комментарии:

SELECT -- Указывает, какие данные или поля будут получены из базы данных
   FirstName AS 'First Name', – Имя поля
   LastName AS [Last Name], – С помощью ключевого слова AS можно
                              переименовать поле
   Company AS Co – Псевдоним, состоящий из одного слова, который нет
                   необходимости заключать в одинарные кавычки или скобки

FROM -- Указывает, какие таблицы содержит база данных
   customers – Ссылка на таблицу customers

ORDER BY -- Указывает порядок сортировки записей по определенному полю;
по умолчанию будет определена сортировка в порядке возрастания (A – Z)
   FirstName DESC – сортировка в порядке убывания (Z – A)

LIMIT -- Задает ограничение на количество записей, выбираемых из базы данных
   10; – Точку с запятой указывать необязательно
```

Глава 5. Ключевые слова

`WHERE, CASE, WHEN, THEN, ELSE, END AS, DATE()`

ПРИМЕЧАНИЕ

Операторы в SQL используются в условиях SQL.

ТИПЫ ОПЕРАТОРОВ

ОПЕРАТОРЫ СРАВНЕНИЯ	ЛОГИЧЕСКИЕ ОПЕРАТОРЫ	АРИФМЕТИЧЕСКИЕ ОПЕРАТОРЫ
= Равно	BETWEEN (Между)	+ Сложение
> Больше, чем	IN (В)	- Вычитание
< Меньше, чем	LIKE (Как)	/ Деление
>= Больше или равно	AND (И)	* Умножение
<= Меньше или равно	OR (Или)	% Остаток от деления
<> Не равно		

Рис. 145

```
SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total
FROM
   invoices
WHERE
Total = 1.98 -- Возвращает только записи, где Total = 1.98
ORDER BY
   InvoiceDate

CASE - Перебирает условия и возвращает значение, когда выполняется первое условие
   WHEN -- Используется для указания условия
   THEN -- Используется после оператора WHEN
```

```
   ELSE -- Используется, если условие не выполнено в условиях WHEN/THEN
   END AS -- Создается новое поле

SELECT
   InvoiceDate,
   BillingAddress,
   BillingCity,
   Total,
CASE -- Создается четыре условия для отображения различных ценовых диапазонов
   WHEN TOTAL < 2.00 THEN 'Baseline Purchase' - Условие 1
   WHEN TOTAL BETWEEN 2.00 AND 6.99 THEN 'Low Purchase'
   WHEN TOTAL BETWEEN 7.00 AND 15.00 THEN 'Target Purchase'
ELSE 'Top Performers' -- Ключевое слово ELSE указывает другие имеющиеся условия
   END AS PurchaseType
FROM
   invoices
ORDER BY
   BillingCity
```

ПРИМЕЧАНИЕ

Для удобства печати в предыдущем примере однострочные комментарии сокращены. Однострочные комментарии всегда должны находиться в одной строке в браузере SQL, иначе они будут ошибочно приняты за код и возникнет ошибка.

	InvoiceDate	BillingAddress	BillingCity	Total	PurchaseType
1	2009-05-10 00:00:00	Lijnbaansgracht 120bg	Amsterdam	8.91	Target Purchase
2	2010-12-15 00:00:00	Lijnbaansgracht 120bg	Amsterdam	1.98	Baseline Purchase
3	2011-03-19 00:00:00	Lijnbaansgracht 120bg	Amsterdam	3.96	Low Purchase
...	...	...	...	...	...
71	2010-03-21 00:00:00	162 E Superior Street	Chicago	15.86	Top Performers

Рис. 146

ПРИМЕЧАНИЕ

DATE() – первая рассмотренная в книге функция. Она представлена ранее других, чтобы ее можно было использовать с ключевыми словами в главе 5. Другие функции рассмотрены в главе 7.

```
/*
С помощью функции DATE() можно удалить любую информацию о временном
коде из данных, хранящихся как тип DATETIME.
*/
SELECT
   InvoiceDate,
   DATE(InvoiceDate) AS [Results of DATE Function]
FROM
   invoices
ORDER BY
   InvoiceDate
```

	InvoiceDate	Results of DATE Function
1	2009-01-01 00:00:00	2009-01-01
2	2009-01-02 00:00:00	2009-01-02
3	2009-01-03 00:00:00	2009-01-03
4	2009-01-06 00:00:00	2009-01-06
5	2009-01-11 00:00:00	2009-01-11
...		

Рис. 147

Глава 6. Ключевые слова

`INNER JOIN, ON, LEFT OUTER JOIN, RIGHT OUTER JOIN, IS, NOT`

ПРИМЕЧАНИЕ

Оператор RIGHT JOIN не поддерживается в SQLite, но поддерживается в других реализациях СУБД.

ВНУТРЕННЕЕ СОЕДИНЕНИЕ

```
SELECT
   i.InvoiceId, -- Указывает ссылку на поле в таблице
   c.CustomerId,
   c.Name,
   c.Address,
   i.InvoiceDate,
```

```
   i.BillingAddress,
   i.Total
FROM
   invoices AS i
INNER JOIN
   customers AS c
ON i.CustomerId = c.CustomerId
```

ЛЕВОЕ ВНЕШНЕЕ СОЕДИНЕНИЕ

```
SELECT
   i.InvoiceId,
   c.CustomerId,
   c.Name,
   c.Address,
   i.InvoiceDate,
   i.BillingAddress,
   i.Total
FROM
   invoices AS i
LEFT OUTER JOIN
   customers AS c
ON
   i.CustomerId = c.CustomerId
```

ПРАВОЕ ВНЕШНЕЕ СОЕДИНЕНИЕ (НЕ ПОДДЕРЖИВАЕТСЯ В SQLITE)

```
SELECT
   i.InvoiceId,
   c.CustomerId,
   c.Name,
   c.Address,
   i.InvoiceDate,
   i.BillingAddress,
   i.Total
FROM
   invoices AS i
RIGHT OUTER JOIN -- Меняет местами таблицы
    customers AS c
ON i.CustomerId = c.CustomerId

SELECT
   ar.ArtistId AS [ArtistId From Artists Table],
   al.ArtistId AS [ArtistId From Albums Table],
```

```
   ar.Name AS [Artist Name],
   al.Title AS [Album]
FROM
   artists AS ar
LEFT OUTER JOIN
   albums AS al
ON
   ar.ArtistId = al.ArtistId
WHERE
   al.ArtistId IS NULL – Можно также использовать IS NOT
```

Глава 7. Ключевые слова

GROUP BY, HAVING

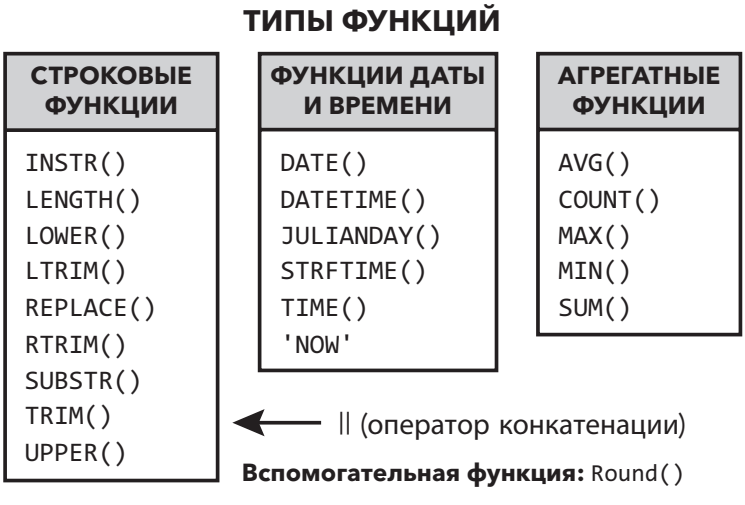


Рис. 148

ПРИМЕЧАНИЕ

SQLite распознает гораздо больше функций, чем включено в эту главу. Более подробную информацию вы можете найти на сайте https://www.sqlite.org/lang_corefunc.html.

Глава 8. Ключевые слова

DISTINCT

Базовый подзапрос:

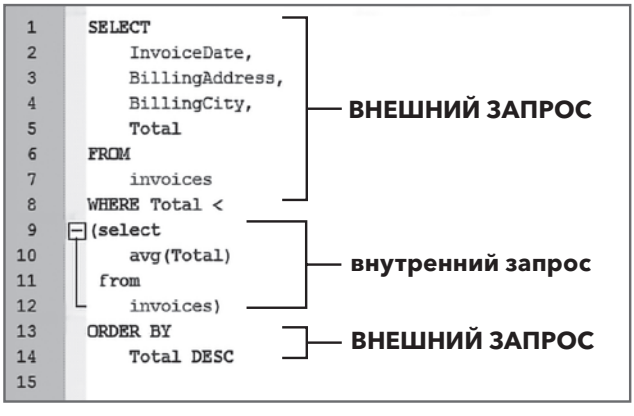


Рис. 149

	TrackId
1	1
2	2
3	3
4	4
5	5
6	6
7	8
8	9
9	10
10	12
...	1984 строки получено за 11 мс

Рис. 150

Условие DISTINCT:

```
SELECT
   DISTINCT TrackId
FROM
   invoice_items
ORDER BY
   TrackId
```

Глава 9. Ключевые слова

CREATE VIEW, DROP VIEW

```
CREATE VIEW V_ViewName AS [Alias Name]

DROP VIEW V_ViewName
```

Глава 10. Ключевые слова

INSERT INTO, UPDATE, SET, DELETE

ВНИМАНИЕ

Язык управления данными (DML) позволяет изменять базу данных. С этими командами лучше попрактиковаться в «песочнице», например на учебной базе данных. Использование DML в рабочей базе с реальными данными может привести к фатальным последствиям.

```
INSERT INTO
artists (Name)
VALUES ('Bob Marley')

UPDATE
employees
SET PostalCode = '11202'
WHERE
   EmployeeId = 9

DELETE FROM
   employees
WHERE
   EmployeeId = 9
```

Об авторе

УОЛТЕР ШИЛДС (WALTER SHIELDS)

Уолтер Шилдс работает с SQL и базами данных уже более восемнадцати лет, сотрудничая с такими организациями, как Target Corporation, NYC Transit Authority и NYC Administration for Children's Services. Он помогает им использовать и понимать свои данные с помощью SQL.

Уолтер начал обучать студентов в кафе в Трайбеке в Нью-Йорке, где у него не было ничего, кроме ноутбука, «набитого» учебными материалами по SQL. С тех пор его наставничество превратилось в бизнес — компанию SQL Training Wheels. Когда Уолтер не обучает студентов, он работает над проектом Datadecided в компании Tableau (разработчик одноименного программного обеспечения для интерактивной визуализации данных и бизнес-аналитики).

Глоссарий

Агрегатная функция

Функция, предназначенная для получения единственного результата на основе содержимого поля. Агрегатные функции могут возвращать сумму, минимальное и максимальное значение, количество или другие математические функции.

Администратор базы данных

Специалист по базам данных, ответственный за обслуживание, безопасность и целостность базы данных. В его обязанности входит решать, кто и к каким частям базы данных получит доступ, и назначать специалиста, ответственного за редактирование базы данных.

Аргументы

Параметры функции, которые, как правило, заключены в круглые скобки `()` и разделены запятой.

Арифметический оператор

Ключевое слово SQL, используемое для выполнения основных арифметических операций (сложение, вычитание, умножение, деление, остаток от деления), которое обычно используется внутри условия `WHERE`.

Атрибут

Еще один вариант представления поля.

База данных

Набор данных, упорядоченный для облегчения и скорости поиска и извлечения с помощью компьютерных технологий.

Браузер SQL

Программный интерфейс системы управления реляционными базами данных, позволяющий конечному пользователю просматривать базы дан-

ных и выполнять запросы с использованием языка структурированных запросов.

Диаграмма «сущность — связь» (ERD)

Схема базы данных, представляющая взаимосвязи между таблицами, например связь между первичным ключом в одной таблице и соответствующими ему внешними ключами в других таблицах. ER-диаграмму также можно назвать схемой.

Внешний ключ

Столбец в таблице, который является первичным ключом в другой таблице.

Данные

Информация, которая может быть записана и сохранена в базе данных.

Запись

Один полный набор информации, состоящий из одной строки и как минимум одного столбца.

Запрос

Запрос, выполненный на языке структурированных запросов, введенный в браузер SQL с запросом определенного набора информации.

Изолированная программная среда («песочница»)

Среда базы данных, изолированная от любых действующих серверов или важных данных, чтобы код можно было протестировать или попрактиковаться в его применении.

Ключевое слово

Специальное зарезервированное слово в SQL, выполняющее определенную функцию в операторе или запросе. `SELECT` — наиболее распространенное ключевое слово SQL.

Логические операторы

Ключевое слово SQL, используемое для выполнения условного выбора данных, которые отвечают определенным критериям; как правило, находится в условии `WHERE`. Например, `BETWEEN`, `IN`, `LIKE`, `AND` и `OR`.

Метаданные

Данные о структуре данных в базе данных.

Набор результатов

Выходные или результирующие данные успешно выполненного запроса, обычно представленные в виде записей из базы данных.

Нормализация

Метод, используемый при создании баз данных для уменьшения количества избыточных столбцов и, таким образом, уменьшения как размера базы данных, так и времени, необходимого для выполнения запросов.

Оператор (operator)

Специальное ключевое слово SQL, как правило, используемое в сочетании с существующим условием SQL, например условием `WHERE`. В SQL существуют следующие типы операторов: операторы сравнения, логические и арифметические операторы.

Оператор (statement)

Любой допустимый фрагмент кода, который может быть выполнен РСУБД.

Оператор сравнения

Ключевое слово SQL, используемое для сравнения значений. Обычно используется в условии `WHERE`.

Например, `=` (равно), `>` (больше), `<` (меньше), `>=` (больше или равно), `<=` (меньше или равно), `<>` (не равно).

Панель запросов

Часть браузера SQL, позволяющая пользователю вводить SQL-запросы.

Панель результатов

Часть браузера SQL, в которой отображается набор результатов или данные, возвращенные из запроса.

Панель сообщений

Часть браузера SQL, в которой отображаются сообщения о полученных результатах выполненных запросов.

Первичный ключ

Столбец, действующий как уникальный идентификатор для конкретной записи в таблице.

Поле

Пространство, выделенное для определенного типа данных. Поле может относиться к одному конкретному элементу в записи или ко всему столбцу. Иногда его называют столбцом или атрибутом.

Псевдоним

Замещающее имя для столбца базы данных, определенное пользователем в операторе `AS`. Псевдонимы используются для предоставления таблице или столбцу таблицы временного имени.

Реляционная база данных

Дизайн базы данных, в котором используется несколько таблиц, связанных друг с другом с помощью полей первичного и внешнего ключей.

Реляционная система управления базой данных

Программный пакет, позволяющий пользователю создавать, редактировать и выполнять SQL-запросы к реляционным базам данных.

РСУБД

Реляционная система управления базами данных.

Условие

Подраздел инструкции SQL, который начинается с ключевого слова и может включать дополнительные параметры и операторы.

Синтаксис

Правильное использование ключевых слов, порядок и структура операторов SQL, чтобы браузер SQL правильно интерпретировал результирующий запрос.

Синтаксическая ошибка

Сообщение об ошибке, созданное браузером SQL из-за некорректного запроса.

Соглашение о кодировании

Набор руководящих принципов, стандартов и методов, используемых в большинстве языков программирования для обеспечения читабельности кода другими заинтересованными сторонами компании.

Составной ключ

Первичный ключ, состоящий из двух или более полей, объединенных таким образом, чтобы образовать уникальный идентификатор.

Столбец

Еще один вариант представления поля.

Строка

Еще один вариант представления записи.

Строковый тип данных

Текстовые данные, хранящиеся в текстовом типе данных, таком как NVARCHAR.

Схема

Описание взаимосвязи между таблицами базы данных и их первичными и внешними ключами, которая может быть показана визуально с помощью ER-диаграммы.

Таблица

Уникальный набор записей, состоящий из строк и столбцов.

Тип данных

Атрибут поля, указывающий, какой тип данных может содержать это поле. Например, числовые и текстовые данные.

Функция

Специальное ключевое слово SQL, которое принимает определенные параметры, называемые аргументами, выполняет операцию (например, вычисление или изменение данных в поле) и возвращает результат этой операции в виде значения.

Целочисленный тип данных

Тип данных, представляющий собой целое (недесятичное) число.

Язык управления (манипулирования) данными (DML)

Подмножество ключевых слов SQL, которые используются для добавления, удаления и изменения данных в базе данных. Например, `INSERT`, `UPDATE` и `DELETE`.

Boolean

Тип данных, возвращающий значение true (истина) или false (ложь).

DB Browser

Браузер SQL, использующий СУБД SQLite.

SQL

Язык структурированных запросов. Стандартизированный набор ключевых слов, специально разработанный для создания, управления и контроля реляционных баз данных.

SQLite

Конкретная реализация SQL, также называемая реляционной системой управления базами данных.

Библиография

Введение

1. “1 Second,” Internet Live Stats, accessed January 24, 2019, http://www.internetlivestats.com/one-second/.
2. https://www.forbes.com/sites/bernardmarr/2015/09/30/big-data-20-mind-boggling-facts-everyone-must-read/#36eb197c17b1.
3. http://wikibon.org/blog/big-data-infographics/.
4. https://www.dezyre.com/article/big-data-timeline-series-of-big-data-evolution/160.
5. https://www.technologyreview.com/s/514346/the-data-made-me-do-it/.
6. https://www.glassdoor.com/Salaries/sql-developer-salary-SRCH_KO0,13.htm.

Глава 1

7. http://www.dictionary.com/browse/datum.
8. https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf.
9. “Most Widely Deployed SQL Database Engine — SQLite,” accessed February 19, 2019, https://www.sqlite.org/mostdeployed.html.

Глава 7

10. “SQLite Query Language: Core Functions,” accessed February 25, 2019, https://www.sqlite.org/lang_corefunc.html
11. “SQLite Query Language: Aggregate Functions,” accessed March 10, 2019, https://www.sqlite.org/lang_aggfunc.html.

Уолтер Шилдс

SQL: быстрое погружение

Перевел с английского А. Павлов

Руководитель дивизиона	*Ю. Сергиенко*
Руководитель проекта	*А. Питиримов*
Ведущий редактор	*Е. Строганова*
Литературный редактор	*Ю. Леонова*
Художественный редактор	*В. Мостипан*
Корректоры	*С. Беляева, Г. Шкатова*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 06.2022. Наименование: книжная продукция. Срок годности: не ограничен.

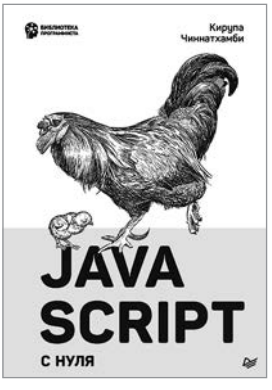
Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 14.04.22. Формат 70×100/16. Бумага офсетная. Усл. п. л. 23,220. Тираж 1000. Заказ 0000.

Кирупа Чиннатхамби

JAVASCRIPT С НУЛЯ



JavaScript еще никогда не был так прост! Вы узнаете все возможности языка программирования без общих фраз и неясных терминов. Подробные примеры, иллюстрации и схемы будут понятны даже новичку. Легкая подача информации и живой юмор автора превратят нудное заучивание в занимательную практику по написанию кода. Дойдя до последней главы, вы настолько прокачаете свои навыки, что сможете решить практически любую задачу, будь то простое перемещение элементов на странице или даже собственная браузерная игра.