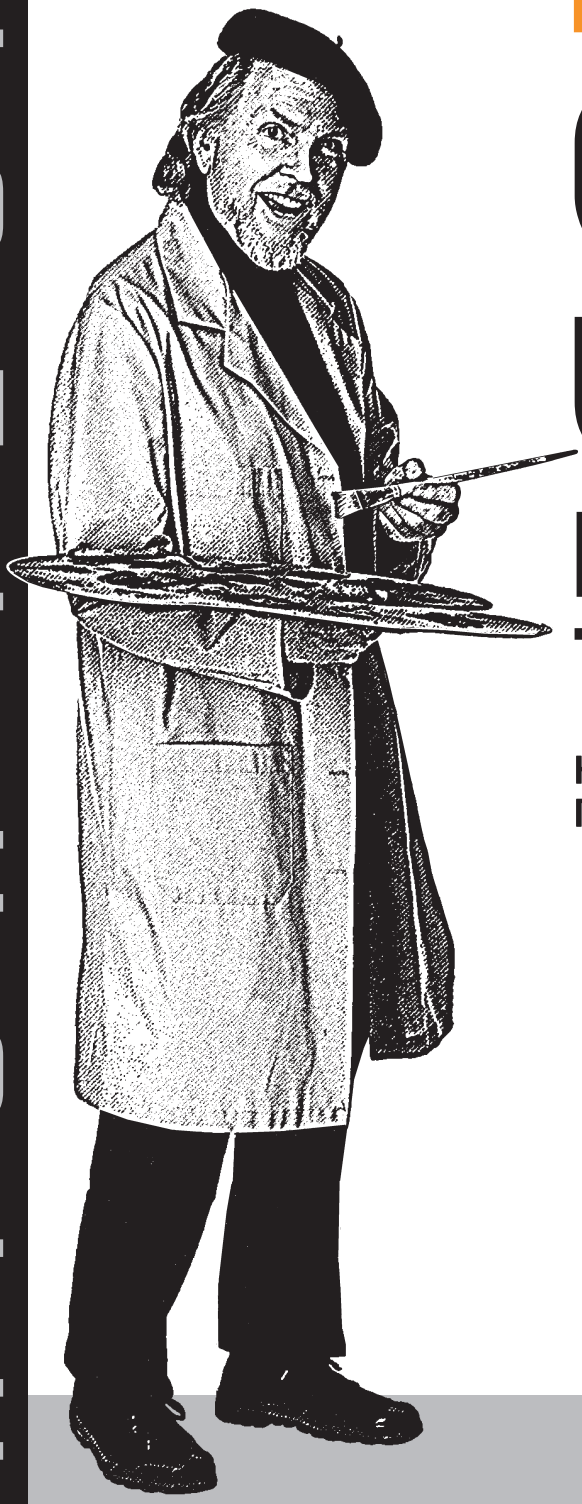
HIGH TECH

К. ДЖ. ДЕЙТ



SQL и реляционная теория

КАК ГРАМОТНО ПИСАТЬ КОД НА SQL

СИМВОЛ®
O'REILLY®

SQL and Relational Theory

How to Write Accurate SQL Code

C. J. Date

O'REILLY®

SQL и реляционная теория

Как грамотно писать код на SQL

К. Дж. Дейт

Санкт-Петербург — Москва
2010

К. Дж. Дейт

SQL и реляционная теория
Как грамотно писать код на SQL

Перевод А. Слинкина

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Редактор	*Ю. Бочина*
Корректор	*С. Минин*
Верстка	*К. Чубаров*

К. Дж. Дейт

SQL и реляционная теория. Как грамотно писать код на SQL. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 480 с., ил.

ISBN 978-5-93286-173-8

Язык SQL распространен повсеместно. Но работать с ним непросто: он сложен, запутан, при написании SQL-команд легко допустить ошибку. Понимание теории, лежащей в основе SQL, – лучший способ гарантировать, что ваш код будет написан правильно, а сама база данных надежна и легко сопровождаема.

В предлагаемой книге К. Дж. Дейт – признанный эксперт, начавший заниматься этими вопросами еще в 1970 году, – демонстрирует, как применить реляционную теорию к повседневной практике работы с SQL. Автор подробно объясняет различные аспекты этой модели, рассуждает и доказывает, приводит многочисленные примеры использования этого языка в соответствии с реляционной теорией.

Не будучи привязанным ни к какому конкретному продукту, издание опирается на многолетний опыт исследований и представляет наиболее актуальное на сегодняшний день изложение материала. Всякий, имеющий хоть какой-то опыт использования SQL – от скромного до весьма обширного, – получит от прочтения этой книги немалую пользу и удовольствие.

ISBN 978-5-93286-173-8
ISBN 978-0-596-52306-0 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 17.02.2010. Формат 70×100 $^{1}/_{16}$. Печать офсетная.
Объем 30 печ. л. Тираж 1200 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Всем считающим,
что подобное упражнение
стоит затраченного времени,
а в особенности Лексу де Хаану,
которого так недостает.

Всякий, кто полагается на практику,
не зная теории, подобен кормчему,
вступающему на судно без руля и компаса, –
он не знает, куда плывет.
Практика всегда должна опираться
на твердые теоретические основания.

Леонардо да Винчи (1452–1519)

Не в том беда, что мы чего-то не знаем,
а в том, что наши знания кажущиеся.

Джош Биллингс (1818–1885)

Языки умирают…
Математические идеи – нет.

Дж. Х. Харди

К сожалению, разрыв между теорией
и практикой в теории не настолько широк,
как на практике.

Аноним

Оглавление

Предисловие

Язык SQL распространен повсеместно. Но работать с SQL трудно: он сложен, запутан, при написании SQL-команд легко допустить ошибку – рискну предположить, что куда легче, чем можно судить по уверениям апологетов этого языка. Поэтому, чтобы уверенно писать правильный SQL-код (то есть такой, который делает в точности то, что вам нужно, не больше и не меньше), необходимо четко следовать некоторому правилу. А основной тезис настоящей книги заключается в том, что таким правилом может стать *использование SQL в соответствии с реляционной теорией*. Но что это означает? Разве SQL не является изначально реляционным языком?

Да, конечно, SQL – стандартный язык для всех реляционных баз данных, но сам по себе этот факт не делает его реляционным. Как это ни печально, SQL слишком часто отходит от принципов реляционной теории; строки-дубликаты и null-значения – два наиболее очевидных примера, которыми проблема отнюдь не ограничивается. Следовательно, он предлагает вам путь, который может привести в западню. А если вы не желаете попадать в западню, то должны понимать реляционную теорию (что она собой представляет и для чего предназначена), знать, в чем именно SQL отклоняется от этой теории, и уметь избегать проблем, которые могут из этого проистекать. Одним словом, SQL следует использовать в реляционном духе. Тогда вы сможете действовать так, будто SQL на самом деле реляционный язык, и получать все преимущества от работы с тем, что по сути является истинно реляционной системой.

В подобной книге не возникло бы необходимости, если бы все и без того использовали SQL реляционным образом, – но, увы, это не так. Напротив, в современном применении SQL я вижу проявление множества вредных тенденций. Более того, эти способы применения относятся к рекомендуемым в различных учебниках и иных публикациях авторами, которые должны были бы относиться к своей работе более ответственно (имен не привожу и на посмешище выставлять не хочу); анализ литературы в этом отношении оставляет удручающее впечатление. Реляционная модель появилась на свет в 1969 году, и вот – почти сорок лет спустя – она, похоже, так и не понята по-настоящему сообществом пользователей СУБД в целом. Отчасти поэтому в основу организации настоящей книги положена сама реляционная модель; подробно объясняются различные аспекты этой модели, и для каждого аспекта демонстрируется, как лучше всего реализовать его средствами SQL.

Предварительные требования

Я предполагаю, что вы применяете СУБД в своей работе, поэтому уже в какой-то мере знакомы с языком SQL. Точнее, я считаю, что вы имеете практическое представление либо о стандарте SQL, либо (что более вероятно) о каком-либо продукте, где применяется SQL. Однако я не рассчитываю на глубокое знание реляционной теории как таковой (хотя надеюсь, что вы все же согласны с тем, что реляционная теория – вещь хорошая, и по возможности ее следует придерживаться). Поэтому во избежание недопонимания я буду подробно описывать различные свойства реляционной модели, а также показывать, как SQL согласуется с этими свойствами. Однако я не стану пытаться обосновывать существование этих свойств, а буду предполагать, что вы достаточно подкованы в тематике баз данных, чтобы понимать, к примеру, зачем используется понятие ключа, или почему иногда приходится выполнять операцию соединения, или зачем требуется поддержка отношений многие-ко-многим. (Если бы я решил включать все определения и обоснования, то получилась бы совсем другая книга – гораздо больше по размеру, не говоря уже обо всем остальном; да и в любом случае такая книга уже написана.)

Я сказал, что ожидаю от вас достаточно близкого знакомства с SQL. Добавлю, однако, что некоторые аспекты SQL я все равно буду объяснять подробно – особенно те, что на практике применяются нечасто. (В качестве примера упомяну «потенциально недетерминированные выражения». См. главу 12.)

«Database in Depth»

Эта книга базируется на ранее изданной книге «Database in Depth: Relational Theory for Practitioners» (O'Reilly, 2005) и должна заменить ее. В предыдущей книге я ставил перед собой такую цель (цитата взята из предисловия):

> Посвятив много лет работе с базами данных в разном качестве, я пришел к выводу, что существует настоятельная потребность в книге для практиков (не новичков), где принципы реляционной теории излагались бы вне связи с особенностями конкретных существующих продуктов, коммерческими обычаями или стандартом SQL. Таким образом, в качестве читательской аудитории я вижу опытных пользователей СУБД, достаточно честных, чтобы сознаться в том, что они не понимают теоретических основ той области, в которой работают, в той мере, в какой могли бы и должны были бы понимать. Этой теорией, естественно, является реляционная модель, и хотя фундаментальные идеи, положенные в ее основу, очень просты, надо признать, что они чуть ли не повсеместно неверно представляются, или недооцениваются,

> или то и другое вместе. Фактически часто их не понимают вовсе. Вот, к примеру, несколько вопросов по реляционной теории[1]... На сколько из них вы сможете ответить?
>
> Что в точности означает первая нормальная форма?
>
> Какая связь существует между отношениями и предикатами?
>
> Что такое семантическая оптимизация?
>
> Что такое отношение-образ?
>
> Почему так важна полуразность?
>
> Почему отложенная проверка ограничений целостности не имеет смысла?
>
> Что такое переменная-отношение?
>
> Что такое предваренная нормальная форма?
>
> Может ли отношение иметь атрибут, значениями которого являются отношения?
>
> Является ли язык SQL реляционно полным?
>
> Почему так важен *принцип информации*?
>
> Как XML укладывается в реляционную модель?
>
> Настоящая книга дает ответы на эти и многие другие вопросы. В общем и целом, ее цель – помочь пользователям-практикам баз данных глубоко разобраться в реляционной теории и применить полученные знания в повседневной профессиональной деятельности.

Как следует из последнего предложения, я надеялся, что читатели той книги смогут применить изложенные идеи в своей работе без дальнейшей помощи с моей стороны. Но с тех пор я осознал, что вопреки устоявшемуся мнению язык SQL настолько труден, что вопрос о том, как применять его, не нарушая реляционных принципов, далеко не праздный. Поэтому я решил дополнить первоначальную книгу, включив в нее явные, конкретные рекомендации именно в этом направлении (то есть как использовать SQL в реляционном духе). Таким образом, в этой книге я преследовал ту же цель, что и раньше (помочь пользователям-практикам баз данных глубоко разобраться в реляционной теории и применить полученные знания в повседневной профессиональной деятельности), но постарался представить материал в виде, быть может, более удобном для усвоения и уж точно – для применения. Иными словами, я включил много материала, относящегося конкретно к языку SQL (и именно поэтому размер так сильно увеличился по сравнению с предыдущим вариантом).

---

[1] По причинам, которые здесь несущественны, я заменил несколько вопросов из оригинального перечня.

Дополнительные замечания о тексте

Я должен высказать еще несколько предварительных замечаний. Прежде всего, мое собственное понимание реляционной модели с годами развивалось. В этой книге изложены мои нынешние представления о предмете; поэтому если вы обнаружите технические расхождения – а они есть – между этой и другими моими книгами (в частности, и той, которую эта призвана заменить), правильным следует считать написанное здесь. Впрочем, сразу оговорюсь, что расхождения по большей части не слишком существенны; более того, я всегда соотношу новые термины и понятия со старыми, если считаю, что в этом есть необходимость.

Во-вторых, я, как и обещано, собираюсь говорить о теории, но опираясь на свою уверенность, что *нет ничего практичнее хорошей теории*. Я специально выделил этот момент, потому что многие, похоже, придерживаются противоположного мнения и считают: все, что отдает теоретизированием, на практике неприменимо. Но истина в том, что теория (по крайней мере, реляционная, а именно о ней я и веду речь) очень даже приближена к практике. Она создавалась как теория *не* ради теории, а ради построения систем, на все сто процентов практических. Каждая деталь этой теории обусловлена весьма практическими соображениями. Один из рецензентов предыдущей книги, Стефан Фарульт (Stéphane Faroult), писал: «Приобретя некоторый практический опыт, вы начинаете осознавать, что без знания теории не обойтись». Более того, эта теория не только практична, она еще и фундаментальна, проста, понятна, полезна, а иногда еще и *забавна* (как я надеюсь продемонстрировать в этой книге).

Разумеется, за самой яркой иллюстрацией вышеприведенного тезиса не нужно ходить дальше самой реляционной модели. Вообще вряд ли необходимо отстаивать мнение о практичности теории в том контексте, который мы имеем: существование многомиллиардной индустрии, целиком основанной на одной замечательной теоретической идее. Однако я предвижу и такую циничную позицию: «Ну ладно, а что эта теория сделала лично для меня в последнее время?» Иными словами, те из нас, кто действительно убежден в важности теории, должны постоянно противостоять критикам – и это еще одна причина, по которой я считаю эту книгу полезной.

В-третьих, как я уже говорил, в этой книге детально излагаются различные аспекты SQL и реляционной модели. (Я сознательно почти не затрагивал темы, не имеющие отношения собственно к реляционной теории, в частности транзакции.) Я всюду старался ясно отмечать, когда обсуждение относится только к SQL, когда – только к реляционной модели, а когда – к тому и другому. Однако должен подчеркнуть, что не стремился к исчерпывающему рассмотрению всех деталей SQL. Язык SQL очень сложен и предоставляет много разных способов решения одной

и той же задачи, в нем так много исключений и особых случаев, что попытка охватить все – даже если бы это было возможно, в чем я лично сомневаюсь, – оказалась бы непродуктивной, а книга при этом выросла бы до необозримых размеров. Поэтому я старался уделить внимание тому, что считаю действительно важным, не растекаясь при этом мыслью по древу. Хотелось бы заявить, что если вы будете делать все, что я советую, и не будете делать того, что я не советую, то это станет первым приближением к безопасной работе: вы будете использовать SQL реляционно. Но только вам судить, насколько это заявление оправдано и оправдано ли вообще.

Ко всему сказанному следует добавить, что, к сожалению, существуют ситуации, когда SQL невозможно использовать реляционно. Например, проверку некоторых ограничений целостности приходится откладывать (обычно, чтобы выиграть время), хотя реляционная модель считает такую отложенную проверку логической ошибкой. В этой книге приводятся рекомендации о том, как поступать в подобных случаях, но боюсь, что по большей части они сводятся к тривиальному совету: *делайте так, как считаете наиболее правильным*. Надеюсь, по крайней мере, что вы будете понимать, в чем состоит опасность отклонения от модели.

Должен также сказать, что некоторые предлагаемые рекомендации не относятся к реляционной теории, а имеют общий характер, хотя иногда у них есть и «реляционные» последствия (не всегда очевидные, кстати говоря). Хорошим примером может служить совет *избегать приведения типов*.

В-четвертых, обратите внимание, что под словом *SQL* я в этой книге понимаю исключительно стандартную версию языка, а не какой-то конкретный диалект (если только противное не оговорено явно). В частности, в согласии со стандартом я подразумеваю произношение «эс кью эл», а не «сиквел»[1] (хотя на практике последнее широко распространено).

В-пятых, эту книгу следует читать в основном последовательно за немногими исключениями, оговоренными здесь и далее в самом тексте (большинство глав в той или иной мере зависят от ранее изложенного материала, поэтому старайтесь не перескакивать из одного места в другое). Кроме того, в каждую главу включены упражнения. Конечно, выполнять их необязательно, но мне кажется, что было бы разумно попробовать свои силы хотя бы на некоторых. Ответы, в которых часто содержится дополнительная информация по теме, приведены в приложении С.

И наконец, хотелось бы упомянуть о некоторых видеосеминарах, основанных на материалах этой книги. Дополнительную информацию см. по адресу *http://www.clik.to/chris_date* или *http://www.thethirdmanifesto.com*.

[1] Поэтому мы пишем «об SQL», а не «о SQL». – *Прим. перев.*

Типографские соглашения

В книге применяются следующие соглашения:

Курсив

Служит для визуального выделения. Этим шрифтом обозначаются новые термины. Кроме того, он применяется в основном тексте, когда нужно сказать, что вместо чего-то следует подставить некоторую переменную, например x.

`Моноширинный шрифт`

Применяется для примеров кода.

`Моноширинный курсив`

В примерах кода обозначает переменную или значение, вводимое пользователем.

О примерах кода

Эта книга призвана помогать вам в работе. Поэтому вы можете использовать приведенный в ней код в собственных программах и в документации. Спрашивать у нас разрешение необязательно, если только вы не собираетесь воспроизводить значительную часть кода. Например, не требуется разрешение, чтобы включить в свою программу несколько фрагментов кода из книги. Однако для продажи или распространения примеров на компакт-диске нужно получить разрешение. Можно без ограничений цитировать книгу и примеры в ответах на вопросы. Но чтобы включить значительные объемы кода в документацию по собственному продукту, нужно получить разрешение.

Мы высоко ценим, хотя и не требуем, ссылки на наши издания. В ссылке обычно указываются название книги, имя автора, издательство и ISBN, например: «SQL and Relational Theory, C. J. Date», Copyright 2009 C. J. Date, 978-0-596-52306-0.

Если вы полагаете, что планируемое использование кода выходит за рамки изложенной выше лицензии, пожалуйста, обратитесь к нам по адресу *permissions@oreilly.com*.

Замечания и вопросы

Я очень старался, чтобы в книге не было ошибок, но что-то мог упустить. Если вы найдете какой-то огрех, просьба уведомить издательство по адресу:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

800-998-9938 (для США и Канады)
707-829-0515 (международный или местный)
707-829-0104 (факс)

Можно также посылать сообщения по электронной почте. Чтобы подписаться на рассылку или заказать каталог, отправьте письмо на адрес:

info@oreilly.com

Замечания и вопросы технического характера следует отправлять по адресу:

bookquestions@oreilly.com

Для этой книги создана веб-страница, на которой выкладываются примеры и списки замеченных ошибок (ранее отправленные извещения об ошибках и исправления доступны для всеобщего обозрения). Адрес страницы:

http://www.oreilly.com/catalog/9780596523060

Дополнительную информацию об этой и других книгах можно найти на сайте издательства O'Reilly:

http://www.oreilly.com

Доступность на Safari

Если на обложке вашей любимой книги присутствует значок Safari® Enabled, это означает, что книга доступна в сетевой библиотеке Safari издательства O'Reilly.

У Safari есть преимущество перед обычными электронными книгами. Это виртуальная библиотека, которая позволяет легко находить тысячи технических книг, копировать примеры программ, загружать отдельные главы и быстро получать точную и актуальную информацию. Бесплатный доступ по адресу *http://safari.oreilly.com*.

Благодарности

О том, чтобы переработать предыдущую книгу, включив в нее дополнительный материал по SQL, я подумывал довольно давно, но последним толчком, заставившим меня наконец засесть за эту работу, явилось мое присутствие в 2007 году на одном курсе, предназначенном для практических пользователей СУБД. Курс вел Тоон Коппелаарс (Toon Koppelaars) на базе книги, которую он написал совместно с Лексом де Хааном (Lex de Haan) (рецензентом этой книги), очень неплохой, кстати. Но поразительным оказалось то, насколько беспомощно выглядели попытки слушателей применить реляционные и логические принципы к привычному для них использованию SQL. Конечно, у слушателей были какие-то знания по этой теме – в конце концов, все они рабо-

тали с базами данных на практике, – но мне показалось, что их определенно нужно направить в сторону применения этих идей в повседневной деятельности. Поэтому я и написал эту книгу. И потому я благодарен в первую очередь Тоону и Лексу, которые придали импульс, необходимый для начала работы над проектом. Я благодарен также рецензентам Хербу Эдельштейну (Herb Edelstein), Шеери Ктитцеру (Sheeri Ktitzer), Энди Ораму (Andy Oram), Питеру Робсону (Peter Robson) и Бэрону Шварцу (Baron Schwartz) за замечания по первым черновикам рукописи и Хью Дарвену (Hugh Darwen) и Джиму Мелтону (Jim Melton) за техническую помощь иного характера. Кроме того, я как всегда благодарен своей жене Линди за неизменную многолетнюю поддержку этого и прочих моих проектов. Наконец, спасибо всему коллективу издательства O'Reilly и особенно Изабель Канкл (Isabel Kunkle), Энди Ораму (Andy Oram) и Адаму Уитверу (Adam Witwer) за ценные советы и за то, что они ободряли и поддерживали меня на протяжении всего времени работы над книгой.

К. Дж. Дейт

Хилдсбург, штат Калифорния,
2008

1

Введение

Реляционный подход к SQL – вот тема или одна из тем настоящей книги. Разумеется, чтобы должным образом раскрыть эту тему, мне придется рассматривать вопросы, касающиеся как реляционной теории, так и самого языка SQL. И хотя это замечание очевидным образом применимо ко всей книге, к первой главе оно относится в особенности. Поэтому здесь мы почти не будем говорить о языке SQL как таковом. Моя цель сейчас – дать обзор материала, большая часть которого вам, скорее всего, уже известна. Я хочу обозначить некую отправную точку, то есть заложить фундамент, на котором можно будет возводить здание книги. Но, даже искренне рассчитывая на то, что вы в основном знакомы с тем, о чем я собираюсь рассказать в этой главе, я все же предлагаю не пропускать ее. Вы должны знать то, что знать надлежит (надеюсь, вы понимаете, что я хочу сказать); в частности, вы должны владеть всем, что понадобится для понимания материала следующих глав. На самом деле я – со всем уважением – порекомендовал бы вообще не пропускать рассмотрение тем, которые вам кажутся знакомыми. Например, так ли вы уверены, что знаете, что в реляционной терминологии понимается под ключом? Или что такое соединение?[1]

---

[1] По крайней мере один ученый муж этого не понимает. Следующий отрывок взят из документа, который (как и эта книга!) имеет целью наставить пользователей SQL: «Не применяйте соединения... Подход Oracle и SQL Server к этой концепции принципиально различен... Вы можете получить неожиданные результирующие наборы... Необходимо понимать основные типы соединений... Экви-соединение строится путем выборки всех данных двух отдельных источников и создания из них одной большой таблицы... В случае внутреннего соединения две таблицы соединяются по внутренним столбцам... В случае внешнего соединения две таблицы соединяются по внешним столбцам... (*да-да, именно так написано в оригинале. – Прим. перев.*). В случае левостороннего соединения две таблицы соединяются по левым столбцам. В случае правостороннего соединения две таблицы соединяются по правым столбцам».

Реляционная модель очень плохо понята

Профессионалы в любой области должны знать лежащие в ее основе фундаментальные идеи. Поэтому профессионал в области баз данных должен знать реляционную теорию, поскольку именно реляционная модель является фундаментом (уж во всяком случае здоровенной частью фундамента), на котором покоится вся эта отрасль. В наше время любой курс по управлению базами данных, академический или коммерческий, на словах привержен идее преподавания реляционной модели, но в большинстве случаев преподавание поставлено очень плохо, если судить по результатам; безусловно, в сообществе пользователей баз данных эта модель недопонята. Перечислим некоторые возможные причины такого положения вещей.

- Модель преподносится в отрыве от всего, в «вакууме». Поэтому начинающему очень трудно уловить значимость материала или понять, какие проблемы модель призвана решить, или то и другое одновременно.
- Сами преподаватели не до конца понимают или не в состоянии оценить важность материала.
- Чаще всего на практике модель как таковая не рассматривается вовсе – вместо нее преподается язык SQL или какой-то его диалект, например принятый в Oracle.

Поэтому эта книга обращена к тем людям, которые на практике работают с базами данных и в особенности с языком SQL и каким-то боком связаны с реляционной моделью, но знают о ней не так много, как должны или хотели бы знать. Она определенно *не рассчитана* на начинающих, однако не является курсом повышения квалификации. Конкретно, я уверен, что вы что-то знаете о языке SQL, но – не сочтите за обиду – если ваши знания о реляционной модели проистекают исключительно из знания SQL, то, боюсь, вы не понимаете ее так хорошо, как следовало бы, и очень может статься, что какие-то «ваши знания неверны». Не устану повторять: *SQL и реляционная модель – вовсе не одно и то же*. В качестве иллюстрации приведу некоторые вопросы реляционной теории, которые в SQL трактуются недостаточно ясно (и это еще мягко сказано):

- Что в действительности представляют собой базы данных, отношения и кортежи
- В чем разница между отношениями-значениями и переменными-отношениями
- Значимость предикатов и высказываний
- Важность имен атрибутов
- Важнейшая роль ограничений целостности

и так далее (список далеко не полный). Все эти и многие другие вопросы рассматриваются в настоящей книге.

Еще раз повторю: если ваши знания о реляционной модели проистекают исключительно из знания SQL, то ваши знания могут оказаться неверными. Отсюда, в частности, следует, что, читая эту книгу, вы, возможно, обнаружите, что кое-что из уже известного следует забыть, а переучиваться, к сожалению, всегда трудно.

Некоторые замечания о терминологии

Вероятно, вы сразу же обратили внимание на то, что в перечне вопросов реляционной теории из предыдущего раздела я употребил формальные термины «отношение», «кортеж» и «атрибут». Но в SQL эти термины не используются, там применяются более «дружественные» слова: таблицы, строка и столбец. Вообще говоря, я с пониманием отношусь к идее употребления понятных пользователю слов, если это помогает усвоению идей. Но в данном случае мне кажется, что как раз усвоению идей такая терминология не способствует – как это ни печально; напротив, она лишь искажает суть и оказывает дурную услугу пониманию истинного смысла.

А истина заключается в том, что отношение – это не таблица, кортеж – не строка, а атрибут – не столбец. И хотя в неформальном контексте такое словоупотребление может показаться приемлемым – я и сам так часто говорю, – я настаиваю, что приемлемо оно лишь в том случае, когда мы ясно понимаем, что эти дружественные термины – не более чем приближение к истине, они совершенно не ухватывают смысл того, что происходит на самом деле. Скажу по-другому: если вы понимаете истинное положение вещей, то благоразумное употребление дружественных терминов может оказаться здравой идеей, но, чтобы понять и оценить это истинное положение, необходимо освоить более формальную терминологию. Поэтому в этой книге я буду, как правило, пользоваться формальными терминами – по крайней мере тогда, когда говорю о реляционной модели, противопоставляя ее SQL, – и в нужном месте приведу их строгие определения. Напротив, в контексте SQL я буду употреблять присущую ему терминологию.

И еще одно замечание о терминологии: упомянув, что SQL пытается упростить один набор терминов, я должен добавить, что он сделал все возможное для усложнения другого. Я имею в виду использование терминов *оператор*, *функция*, *процедура*, *подпрограмма* и *метод*; все они обозначают по существу одно и то же (быть может, с незначительными вариациями). В этой книге я всюду буду употреблять слово *оператор*.

Раз уж зашла речь об SQL, позвольте напомнить, что (как отмечалось в предисловии) под этим я понимаю исключительно стандартную вер-

сию языка[1], если не считать нескольких мест, где по контексту требуется иное.

- Иногда я употребляю терминологию, отличающуюся от стандартной. Например, я пишу *табличное выражение* (table expression) вместо принятого в стандарте *выражение запроса* (query expression), потому что (а) значением такого выражения является таблица, а не запрос, и (б) запросы – не единственный контекст, в котором такие выражения используются. (На самом деле, в стандарте применяется термин *табличное выражение*, но в гораздо более узком смысле; конкретно, он обозначает то, что следует за фразой SELECT в выражении SELECT.)
- Продолжая предыдущую мысль, должен добавить, что не все табличные выражения допустимы в SQL в любом контексте, где их использование можно было бы предположить. В частности, результат явной операции JOIN, хотя он, безусловно, обозначает таблицу, не может встречаться в качестве «отдельно стоящего» табличного выражения (то есть на самом внешнем уровне вложенности), а также не может выступать в качестве табличного выражения в скобках, которое составляет подзапрос (см. главу 12). *Обратите внимание, что подобные замечания применимы ко многим обсуждениям в основном тексте книги, но повторять их каждый раз было бы скучно, поэтому я этого делать не буду.* (Однако все это сведено в БНФ-грамматике в главе 12.)
- Я игнорирую те аспекты стандарта, которые можно считать чрезмерно эзотерическими, особенно если они не являются частью того, что в стандарте называется Core SQL (Базовый SQL), или имеют мало общего с реляционной обработкой как таковой. В качестве примеров упомяну так называемые аналитические или оконные функции (OLAP), динамический SQL, рекурсивные запросы, временные таблицы и детали типов, определенных пользователем.
- По причинам, отчасти связанным с типографским набором, я применяю для комментариев стиль, отличающийся от стандартного. Точнее, комментарии печатаются курсивом и обрамлены ограничителями «/*» и «*/».

Не забывайте, что все реализации SQL включают средства, не описанные в стандарте. Типичный пример – идентификаторы строк. Моя общая рекомендация относительно таких средств такова: пользуйтесь ради бога, но только если они не нарушают реляционных принципов (в конце концов, эта книга посвящена описанию *реляционного* подхода к SQL). Например, применение идентификаторов строк, скорее всего, нарушит так называемый *принцип взаимозаменяемости* (см. гла-

[1] International Organization for Standardization (ISO): *Database Language SQL*, Document ISO/IEC 9075:2003 (2003).

ву 9), и если это так, я бы точно не стал их использовать. Но и в этом, и во всех остальных случаях действует универсальное правило: можете делать все, что хотите, если только знаете, что делаете.

Принципы, а не продукты

Стоит потратить немного времени на вопрос о том, почему профессионалу в области баз данных необходимо (как я уже отмечал выше) знать реляционную модель. Причина в том, что реляционная модель не связана ни с каким конкретным продуктом; она имеет дело только с принципами. Что я понимаю под принципами? Словарь дает такие определения:

> **Принцип** – основное, исходное положение теории, учения, мировоззрения; убеждение, взгляд на вещи; основная особенность в устройстве чего-либо.[1]

Важнейшая особенность принципов состоит в их долговечности. Продукты и технологии (и язык SQL в том числе) все время изменяются – принципы остаются постоянными. Допустим, к примеру, что вы знаете СУБД Oracle; предположим даже, что вы крупный специалист по Oracle. Но если ваши знания ограничены только Oracle, то они могут оказаться непригодными, скажем, в среде DB2 или SQL Server (а могут даже затруднить освоение новой среды). Однако если вы знаете основополагающие принципы – иными словами, реляционную модель, – ваши знания и навыки *переносимы*: вы сможете применить их в любой среде, и они никогда не устареют.

Поэтому в этой книге мы будем иметь дело с принципами, а не с продуктами, с основами, а не с преходящими увлечениями. Но я понимаю, что в реальном мире иногда приходится идти на компромиссы. Например, иногда имеются веские причины проектировать базу данных способом, далеким от теоретически оптимального. В качестве другого примера снова обратимся к SQL. Хотя нет сомнений, что SQL можно использовать реляционно (по крайней мере, в большинстве случаев), иногда обнаруживается – ведь существующие реализации так далеки от совершенства, – что это влечет за собой существенное падение производительности... и тогда вы более или менее вынуждены делать что-то не «истинно реляционное» (например, писать запрос неестественным образом, чтобы заставить реализацию воспользоваться индексом). Однако я абсолютно убежден, что такие компромиссы всегда следует оценивать с *концептуальных позиций*. Это означает, что:

- Идя на такой компромисс, вы должны понимать, что делаете.
- Вы должны понимать, как выглядит теоретически правильное решение, и иметь основательные причины для отхода от него.

---

[1] Перевод дан по изданию Толкового словаря русского языка С. И. Ожегова. – *Прим. перев.*

- Вы должны документировать эти причины; тогда в случае, если в будущем они отпадут (например, из-за того, что в новой версии продукта какая-то функция реализована более удачно), можно будет отказаться от первоначального компромисса.

Следующий афоризм – который приписывается Леонардо да Винчи (1452–1519), и, стало быть, ему уже около 500 лет – великолепно описывает эту ситуацию:

> Всякий, кто полагается на практику, не зная теории, подобен кормчему, вступающему на судно без руля и компаса, – он не знает, куда плывет. *Практика всегда должна опираться на твердые теоретические основания.*

(Курсив добавлен мной.)

Обзор оригинальной модели

В этом разделе я хочу задать отправную точку для последующего обсуждения; здесь приводится обзор некоторых базовых аспектов реляционной модели в том виде, в котором она была определена изначально. Обратите внимание на уточнение – «была определена изначально»! Бытует широко распространенное заблуждение, будто реляционная модель – вещь абсолютно неизменная. Это не так. В этом отношении она напоминает математику: математика тоже не статична, а изменяется со временем. Да ведь и сама реляционная модель – это тоже отрасль математики и как таковая эволюционирует по мере доказательства новых теорем и получения новых результатов. Более того, новый вклад может быть внесен любым компетентным специалистом. И, подобно математике, реляционная модель, хотя и была первоначально изобретена одним человеком, ныне стала плодом совместных усилий и принадлежит всему человечеству.

Кстати говоря, если вы не в курсе, этим человеком был Э. Ф. Кодд, в то время работавший исследователем в корпорации IBM (Э – это Эдгар, Ф – Фрэнк, но он всегда подписывался одними инициалами; а для друзей, к которым я с гордостью отношу и себя, он был просто Тедом). В конце 1968 года Кодд, математик по образованию, впервые понял, как применить математику для закладывания строгих принципов в основание области знания – управления базами данных, – которой в то время таких качеств остро недоставало. Его первоначальное определение реляционной модели появилось в научно-исследовательском отчете IBM в 1969 году, и я еще вернусь к этой работе в приложении D.

Структурные свойства

В оригинальной модели было три основных компонента – структура, целостность и средства манипулирования, и я коротко опишу каждый из них. Но сразу прошу учесть, что все даваемые мной «определе-

ния» очень нестроги; я уточню их в последующих главах, когда это будет уместно.

Итак, начнем со структуры. Конечно, главным структурным свойством является само отношение, и, как всем известно, отношения принято изображать на бумаге в виде таблиц (пример на рис. 1.1 не нуждается в пояснениях). Отношения определены над *типами* (их называют также *доменами*). По существу, тип представляет собой концептуальное множество значений, которые могут принимать фактические атрибуты фактических отношений. Обращаясь к простой базе данных об отделах и служащих, которая показана на рис. 1.1, мы можем выделить тип DNO («номера отделов»), представляющий все допустимые номера отделов, и тогда атрибут DNO в отношении DEPT и атрибут DNO в отношении EMP будут принимать значения из соответствующего ему концептуального множества. (Кстати, атрибуты не обязательно – хотя иногда это и разумно – называть так же, как соответствующий тип, и часто так не делают. Ниже мы увидим много контрпримеров.)

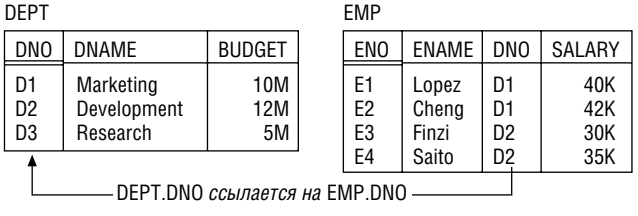


Рис. 1.1. *База данных об отделах и служащих – тестовые значения*

Как я уже сказал, таблицы, подобные представленной на рис. 1.1, изображают *отношения*, точнее *n*-арные отношения. *N*-арные отношения можно нарисовать в виде таблицы с *n* столбцами; столбцы будут соответствовать *атрибутам* отношения, а строки – *кортежам*. Здесь *n* – произвольное неотрицательное целое число. 1-арное отношение называют *унарным*, 2-арное – *бинарным*, 3-арное – *тернарным* и т. д.

Реляционная модель поддерживает также различные виды *ключей*. Начнем с того – и это крайне важно! – что в каждом отношении есть по меньшей мере один *потенциальный ключ*[1]. Потенциальный ключ – это просто уникальный идентификатор; иными словами, это комбинация атрибутов – часто, но не всегда, «комбинация» состоит всего из одного атрибута, – такая, что значения этой комбинации во всех кортежах отношения различны. Так, на рис. 1.1 у каждого отдела имеется уникаль-

[1] Строго говоря, это предложение должно выглядеть так: «Каждая *переменная-отношение* имеет по меньшей мере один потенциальный ключ» (см. раздел «Отношения и переменные-отношения» ниже). Аналогичное замечание относится и к другим местам в этой главе. См. упражнение 1.1 в конце главы.

ный номер отдела, а у каждого служащего – уникальный номер служащего, поэтому мы можем сказать, что {DNO} – потенциальный ключ отношения DEPT, а {ENO} – потенциальный ключ отношения EMP. Кстати, обратите внимание на фигурные скобки; повторю, что потенциальные ключи всегда являются комбинациями, или множествами, атрибутов – даже если конкретное множество состоит всего из одного атрибута, а множества на бумаге традиционно изображают в виде заключенного в фигурные скобки списка элементов, перечисленных через запятую.

Отступление

Здесь я впервые употребил термин *список* (разделенный запятыми) (*comma-list*), который далее будет встречаться очень часто. Его можно определить следующим образом. Пусть *xyz* – некоторая синтаксическая конструкция (например, «имя атрибута»). Тогда *список xyz* означает последовательность из нуля или более элементов *xyz*, в котором каждая пара соседних элементов разделена запятой (дополнительно перед и после запятой может находиться один или несколько пробелов). Так, если *A*, *B* и *C* – имена атрибутов, то каждая из показанных ниже конструкций представляет собой список имен атрибутов:

A , B , C

C , A , B

A , C

B

Списком является и пустая последовательность имен атрибутов. Кроме того, если список заключен в фигурные скобки и, следовательно, обозначает множество, то (а) порядок следования элементов в списке не существенен (так как множества по определению неупорядочены) и (б) если элемент встречается более одного раза, то считается, что он встретился только один раз (так как множества не содержат дубликатов).

Далее, *первичным* ключом называется такой потенциальный ключ, который по каким-то причинам интерпретируется специальным образом. Если рассматриваемое отношение имеет только один потенциальный ключ, то его с равным успехом можно назвать первичным. Но если потенциальных ключей несколько, то обычно один из них выбирается в качестве первичного, то есть считается, что он в каком-то отношении «равнее других». Предположим, к примеру, что у каждого служащего имеется как уникальный номер, так и уникальное имя – пример, пожалуй, не слишком реалистичен, но вполне пригоден для иллюстрации, – тогда и {ENO}, и {ENAME} являются потенциальными ключами EMP. В этом случае мы можем назначить, например, {ENO} первичным ключом.

Обратите внимание: я сказал, что первичный ключ *обычно* выбирается. Обычно – но не обязательно. Если имеется всего один потенциальный ключ, то вопрос о выборе вообще не встает; но если таких ключей два или больше, то выбор одного из них в качестве первичного от-

дает произвольностью (по крайней мере, мне так кажется). Разумеется, бывают ситуации, когда нет веских оснований отдать предпочтение конкретному кандидату. В этой книге я, как правило, буду выбирать какой-то первичный ключ – и на рисунках, подобных рис. 1.1, обозначать составляющие первичный ключ атрибуты двойным подчеркиванием, – но хочу особо подчеркнуть, что с точки зрения реляционной теории важность представляют именно потенциальные, а не первичные ключи. Отчасти по этой причине я в дальнейшем под словом *ключ*, без уточнения, буду понимать любой потенциальный ключ. (Если вам интересно, то скажу, что «специальная интерпретация» первичных ключей по сравнению с потенциальными носит в основном синтаксический характер; она не является фундаментальной особенностью и вообще не очень существенна.)

Наконец, *внешним* ключом называется множество атрибутов одного отношения, значения которых должны совпадать со значениями некоторого потенциального ключа в каком-то другом (или в том же самом) отношении. Так, на рис. 1.1 {DNO} – внешний ключ в отношении EMP, значения которого должны совпадать со значениями потенциального ключа {DNO} в отношении DEPT (что я и попытался отразить на рисунке посредством соответственно надписанной стрелочки). Говоря «должны совпадать», я имею в виду, что если отношение EMP содержит, к примеру, кортеж, в котором DNO имеет значение D2, то и DEPT должно содержать кортеж, в котором DNO имеет значение D2, – иначе в отношении EMP окажется служащий, который работает в несуществующем отделе, и база данных перестанет быть «верной моделью реальности».

Свойства целостности

Ограничение целостности (или, для краткости, просто *ограничение*) по существу представляет собой булево выражение, которое должно принимать значение TRUE. Например, для отделов и служащих мы могли бы ввести ограничение, которое гласит, что значение атрибута SALARY (зарплата) должно быть больше нуля. Вообще говоря, на любую базу данных налагается много разнообразных ограничений, однако все они по необходимости специфичны для конкретной базы и потому должны выражаться в терминах отношений в этой базе. Но в оригинальную реляционную модель включены также два *обобщенных* ограничения целостности – обобщенных в том смысле, что они применимы к любой базе данных (если говорить неформально). Одно касается первичных ключей, другое – внешних.

Правило целостности сущностей

Атрибуты, входящие в состав первичного ключа, не могут принимать null-значений.

Правило ссылочной целостности

Не должно быть внешних ключей, не имеющих соответствия.

Сначала я объясню смысл второго правила. Говоря о *внешнем ключе, не имеющем соответствия*, я имею в виду значение внешнего ключа, для которого не существует соответствующего ему потенциального ключа с таким же значением; например, в базе данных об отделах и служащих правило ссылочной целостности было бы нарушено, если бы в отношении EMP встретилось, к примеру, значение D2 атрибута DNO, но в отношении DEPT не оказалось бы кортежа с таким же значением DNO. Таким образом, правило ссылочной целостности просто выражает семантику внешних ключей, а название «ссылочная целостность» напоминает о том факте, что значение внешнего ключа можно рассматривать как *ссылку* на кортеж с таким же значением соответственного потенциального ключа. По существу, правило говорит: если *B* ссылается на *A*, то *A* должно существовать.

Что же касается правила целостности сущностей, то я, скажем так, испытываю затруднение. Дело в том, что я полностью отвергаю концепцию «null-значений»; по моему глубокому убеждению, *им не место в реляционной модели*. (Кодд считал иначе, очевидно, но и у меня есть сильные аргументы в защиту собственной позиции.) Поэтому, чтобы объяснить смысл правила целостности сущностей, мне придется (по крайней мере, на время) забыть о своем неверии. Что я и сделаю... но имейте в виду, что я еще вернусь к вопросу о null-значениях в главах 3 и 4.

По сути своей, null – это «маркер», говорящий, что *значение неизвестно* (подчеркну – и это принципиально важно, что null-значение – и не значение вовсе, а только *маркер*, или *флаг*). Допустим, к примеру, что мы не знаем величину зарплаты служащего E2. Тогда вместо того, чтобы вводить какое-то реальное значение SALARY в кортеж, описывающий этого служащего в отношении EMP, – по определению, мы не можем этого сделать, так как не знаем нужного значения, – мы *помечаем* атрибут SALARY в этом кортеже флагом null:

ENO	ENAME	DNO	SALARY
E2	Cheng	D1	

Важно понимать, что этот кортеж не содержит *ничего* в позиции SALARY. Но очень трудно изобразить «ничто» на рисунке! На рисунке выше я попытался показать, что позиция SALARY пуста, затенением ячейки, но было бы точнее не показывать ее вовсе. Так или иначе, я и дальше буду придерживаться того же соглашения: обозначать пустые позиции затенением, памятуя о том, что затенение говорит об отсутствии всякого значения. Если вам так удобнее, можете считать, что она представляет собой маркер, или флаг null.

Возвращаясь к соотношению EMP, правило целостности сущностей говорит (неформально выражаясь), что у любого служащего может быть неизвестно имя, отдел или зарплата, но не номер служащего, поскольку

ку, если неизвестен номер, то мы вообще не знаем, о каком служащем (сущности) идет речь.

Вот и все, что я хочу сказать о null-значениях на данный момент. И пока забудем о них.

Средства манипулирования

Раздел модели, относящийся к манипулированию, состоит из двух частей:

- *Реляционная алгебра*, в которой определяется набор операторов, например разность (или MINUS), применимых к отношениям.
- Оператор *реляционного присваивания*, который позволяет присвоить значение реляционного выражения (например, *r1 MINUS r2*, где *r1* и *r2* – отношения) какому-то отношению.

Оператор реляционного присваивания по существу описывает способ выполнения обновлений в реляционной модели, и я еще вернусь к нему ниже в разделе «Отношения и переменные-отношения». *Примечание*: в этой книге я придерживаюсь традиционного соглашения о том, что общим термином *обновление* (*update*) обозначается совокупность реляционных операторов INSERT, DELETE и UPDATE (а также присваивания). Когда речь идет о конкретном операторе UPDATE, я записываю его заглавными буквами.

Что касается реляционной алгебры, то она состоит из набора операторов, которые – говоря очень нестрого – позволяют порождать «новые» отношения из «старых». Каждый оператор принимает на входе одно или несколько отношений и на выходе возвращает новое отношение; например, оператор разности (MINUS) принимает на входе два отношения и «вычитает» одного из другого, порождая на выходе третье отношение. Очень важно, что результатом является отношение: это хорошо известное свойство *замкнутости* реляционной алгебры. Именно свойство замкнутости позволяет записывать вложенные реляционные выражения; так как результат любой операции – объект того же вида, что и входные операнды, то результат одной операции можно подать на вход другой, – например, разность *r1 MINUS r2* может быть объединена с отношением *r3*, затем результат пересечен с отношением *r4* и т. д.

Можно определить сколько угодно операторов, удовлетворяющих простому условию: «одно или несколько отношений на входе, единственное отношение на выходе». Ниже я кратко опишу те операторы, которые принято считать исходными (по существу, те, что были определены Коддом в ранних работах)[1]; в главах 6 и 7 я остановлюсь на них бо-

[1] Исключение составляет дополнительно определенный Коддом оператор деления *divide*. В главе 7 я объясню, почему опустил его здесь.

лее подробно и определю некоторые дополнительные операторы. На рис. 1.2 показано графическое представление исходных операторов. Примечание: если вы незнакомы с этими операторами и не понимаете, что означают картинки, не расстраивайтесь; как я уже сказал, в последующих главах они будут описаны детально, с множеством примеров.

Ограничение[1]

Возвращает отношение, содержащее все кортежи заданного отношения, которые удовлетворяют заданному условию. Например, можно ограничить отношение EMP только кортежами, для которых атрибут DNO имеет значение D2.

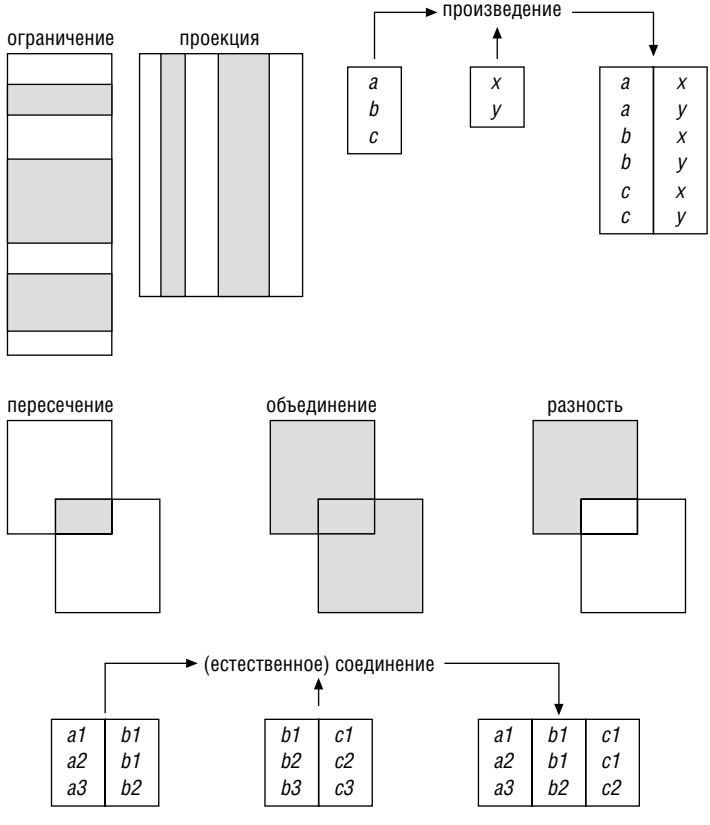


***Рис. 1.2.** Оригинальная реляционная алгебра*

[1] Слова *constraint* (ограничение целостности) и *restriction* (выборка) переводятся на русский язык одним словом *ограничение*. К счастью, из контекста всегда понятно, о чем именно идет речь. *–Прим. перев.*

Проекция

Возвращает отношение, содержащее все (под)кортежи заданного отношения, которые остались после исключения из него некоторых атрибутов. Например, можно спроецировать отношение EMP на атрибуты ENO и SALARY (исключив тем самым атрибуты ENAME и DNO).

Произведение

Возвращает отношение, содержащее все возможные кортежи, которые являются комбинацией двух кортежей, принадлежащих соответственно двум заданным отношениям. Этот оператор известен также как *декартово произведение* (иногда он называется *расширенным декартовым произведением*), *перекрестное произведение*, *перекрестное соединение* или *декартово соединение*; по сути дела, это просто частный случай операции соединения, как будет показано в главе 6.

Пересечение

Возвращает отношение, содержащее все кортежи, которые принадлежат одновременно двум заданным отношениям. (На самом деле, пересечение также является частным случаем соединения, как будет показано в главе 6.)

Объединение

Возвращает отношение, содержащее все кортежи, которые принадлежат либо одному из двух заданных отношений, либо им обоим.

Разность

Возвращает отношение, содержащее все кортежи, которые принадлежат первому из двух заданных отношений, но не принадлежат второму.

Соединение

Возвращает отношение, содержащее все возможные кортежи, которые представляют собой комбинации двух кортежей, принадлежащих двум заданным отношениям, при условии, что в комбинируемых кортежах присутствуют одинаковые значения в одном или нескольких общих для исходных отношений атрибутах (причем эти общие значения появляются в результирующем кортеже один раз, а не дважды). *Примечание*: Соединение такого вида изначально называлось *естественным*. Однако, поскольку естественное соединение несомненно является наиболее важным, то теперь принято называть его просто соединением, опуская уточняющий квалификатор; я тоже буду придерживаться такого соглашения.

И еще одно заключительное замечание: как вы, возможно, знаете, существует еще так называемое реляционное исчисление. Его можно счи-

тать альтернативой реляционной алгебре, то есть с равным успехом можно сказать, что манипуляционная часть реляционной модели состоит из реляционной алгебры (и оператора реляционного присваивания) или из реляционного исчисления (и оператора реляционного присваивания). Они эквивалентны и взаимозаменяемы в том смысле, что для каждого алгебраического выражения существует логически эквивалентное ему выражение реляционного исчисления и наоборот. Я еще буду говорить о реляционном исчислении, в основном в главах 10 и 11.

Сквозной пример

Я завершу этот краткий обзор описанием примера, который будет основой при обсуждении большинства, если не всех, тем в этой книге: всем известной – если не сказать навязшей в зубах – базы данных о поставщиках и деталях. (Прошу прощения за то, что еще раз вывел этого престарелого боевого коня из стойла, но я полагаю, что использование одного и того же примера в разных публикациях способствует, а не мешает обучению.) На рис. 1.3 приведены тестовые значения.

S

SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SNO	PNO	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

P

PNO	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12.0	London
P2	Bolt	Green	17.0	Paris
P3	Screw	Blue	17.0	Oslo
P4	Screw	Red	14.0	London
P5	Cam	Blue	12.0	Paris
P6	Cog	Red	19.0	London

Рис. 1.3. *База данных о поставщиках и деталях – тестовые значения*

Уточним понятия.

Поставщики

Отношение S описывает поставщиков (точнее, поставщиков по контракту). У каждого поставщика имеется уникально определяющий его номер (SNO) (как видите, я сделал {SNO} первичным ключом), название (SNAME), необязательно уникальный (пусть даже на рис. 1.3 все значения SNAME различаются) статус (STATUS), представляющий некое свойство, позволяющее предпочесть одного поставщика другому, и местоположение (CITY).

Детали

Отношение P представляет детали (точнее, виды деталей). У каждой детали есть уникальный номер (PNO) ({PNO} – первичный ключ), одно наименование (PNAME), один цвет (COLOR), один вес (WEIGHT) и одно местоположение, то есть склад, где хранятся детали этого вида (CITY).

Поставки

Отношение SP описывает поставки (то есть показывает, какие детали поставляются какими поставщиками). У каждой поставки имеется один номер поставщика (SNO), один номер детали (PNO) и одно количество (QTY). В этом примере я буду предполагать, что в любой момент времени существует не более одной поставки с данным поставщиком и данной деталью ({SNO,PNO} – первичный ключ, кроме того, {SNO} и {PNO} являются внешними ключами, сопоставляемыми первичным ключам S и P, соответственно). Обратите внимание, что в базе данных на рис. 1.3 есть один поставщик, S5, для которого нет ни одной поставки.

Модель и реализация

Прежде чем двигаться дальше, я хочу остановиться на одном моменте, существенном для всего, что будет обсуждаться ниже. Разумеется, реляционная модель – это модель данных. Но, к сожалению, последний термин имеет в мире баз данных два совершенно разных значения. Первое и наиболее фундаментальное таково:

> **Определение**: *Модель данных* (в первом смысле) – это абстрактное, независимое, логическое определение структур данных, операторов над данными и прочего, что в совокупности составляет *абстрактную систему*, с которой взаимодействует пользователь.

Именно это значение слова мы имеем в виду, когда говорим о реляционной модели. И, вооружившись этим определением, мы можем провести полезное, и важное, различие между моделью данных в первом смысле и ее реализацией, которая определяется следующим образом.

> **Определение**: *Реализацией* данной модели данных называется физическое воплощение на реальной машине тех компонентов абстрактной системы, которые в совокупности составляют модель.

Я проиллюстрирую эти определения в терминах реляционной модели. Прежде всего, само понятие *отношения* является частью модели данных: пользователи должны знать, что такое отношение, понимать, что оно состоит из кортежей и атрибутов, уметь их интерпретировать и т. д.

Все это части модели. Но пользователям необязательно знать, как отношения физически хранятся на диске, как физически кодируются отдельные значения данных, какие существуют индексы или другие пути доступа к данным – это части реализации, а не модели.

Или возьмем концепцию *соединения*: пользователи должны знать, что такое соединение, как его осуществить, как выглядит результат соединения и т. д. Опять-таки, все это части модели. Но совершенно ни к чему знать, как физически реализуется соединение, какие при этом производятся трансформации выражений, какие используются индексы или иные пути доступа к данным, какие требуются операции ввода/вывода – это части реализации, а не модели.

И еще один пример: *потенциальные ключи* (для краткости просто *ключи*) – часть модели, и пользователям определенно необходимо знать, что представляют собой ключи. На практике уникальность ключей часто гарантируется посредством так называемого уникального индекса; но индексы вообще и уникальные индексы в частности не являются частью модели, это часть реализации. Таким образом, не следует путать индекс с ключом в реляционном смысле, даже если первый применяется для реализации второго (точнее, для реализации некоторого *ограничения ключа*, см. главу 8).

Короче говоря:

- Модель (в первом смысле) – это то, что пользователь должен знать.
- Реализация – это то, что пользователю знать необязательно.

Я вовсе не хочу сказать, что пользователям запрещено знать о реализации; я лишь говорю, что это необязательно. Иными словами, все касающееся реализации должно быть, по крайней мере потенциально, *скрыто от пользователя*.

Из данных выше определений вытекают некоторые важные следствия. Прежде всего (и вопреки чрезвычайно распространенному заблуждению), все связанное с производительностью принципиально является деталью реализации, а не модели. Например, мы часто слышим, что «соединение – медленная операция». Но такое замечание лишено всякого смысла! Соединение – это часть модели, а модель как таковая не может быть ни медленной, ни быстрой; такими качествами может обладать только реализация. Поэтому допустимо сказать, что в некотором продукте X конкретная операция соединения реализована быстрее или медленнее, чем в продукте Y, – но это и все.

Я не хочу, чтобы у вас в этом месте сложилось ложное впечатление. Да, производительность принципиально является деталью реализации; однако это не означает, что хорошая реализация будет работать быстро, если модель используется неправильно. На самом деле, это как раз одна из причин, по которым вы должны знать устройство модели: чтобы не использовать ее неправильно. Написав выражение S JOIN SP, вы впра-

ве ожидать, что реализация сделает все необходимое и сделает хорошо; но если вы настаиваете на ручном кодировании соединения, например таким образом (в виде псевдокода):

```
do for all tuples in S ;
   fetch S tuple into TNO , TN , TS , TC ;
   do for all tuples in SP with SNO = TNO ;
      fetch SP tuple into TNO , TP , TQ ;
      emit tuple TNO , TN , TS , TC , TP , TQ ;
   end ;
end ;
```

то рассчитывать на достижение высокой производительности было бы глупо. **Рекомендация**: Не поступайте так. Реляционные системы не следует использовать в качестве простых методов доступа.[1]

Кстати, эти замечания по поводу производительности относятся и к SQL. Утверждения о быстроте или медленности точно так же бессмысленны в применении к SQL, как и к реляционным операторам (соединению и прочим); в этих терминах имеет смысл говорить только о реализациях. Однако использовать SQL можно и так, что это гарантированно приведет к плохой производительности. Хотя в этой книге тема производительности почти не затрагивается, иногда я все же буду отмечать последствия своих рекомендаций с этой точки зрения.

Отступление

Я хотел бы ненадолго задержаться на вопросе о производительности. Вообще говоря, приводя в этой книге ту или иную рекомендацию, я не руководствовался соображениями производительности; в конце концов, реляционная модель всегда ставила целью передать заботу о производительности от пользователя системе. Однако не стоит и говорить, что эта задача так и не была решена в полной мере, и потому (как я уже отмечал) поставленной целью – реляционное использование SQL – иногда приходится жертвовать в интересах достижения приемлемой производительности. И это еще одна причина, по которой действует универсальное правило: *можете делать все, что хотите, если только знаете, что делаете*.

Но вернемся к различию между моделью и реализацией. Второй момент, как вы, вероятно, догадались, состоит в том, что именно разделение модели и реализации позволяет добиться *физической независимости от данных*. Физическая независимость от данных – термин, по-

[1] Сразу несколько рецензентов обратили внимание на то, что это предложение лишено смысла (как можно использовать систему в качестве метода?). Что ж, если вы настолько молоды, что не знакомы с термином *метод доступа*, могу только позавидовать; но факт в том, что этот термин, пусть не очень подходящий, в прошлом широко использовался для обозначения того или иного простого механизма ввода/вывода на уровне отдельных записей.

жалуй, не слишком удачный, но он уже устоялся, – означает, что мы можем как угодно изменять способ физического хранения и доступа к данным, не внося соответствующих изменений в то, как данные представляются пользователю. Причина, по которой может возникнуть желание изменить детали хранения и доступа, как правило, связана с производительностью; а тот факт, что такие изменения можно производить, не меняя способа представления данных пользователю, означает, что существующие программы, запросы и прочее будут продолжать работать. Таким образом, – и это очень важно – физическая независимость от данных позволяет *уменьшить капиталовложения в обучение пользователей и разработку приложений* и, добавлю еще, капиталовложения в проектирование логической структуры базы данных.

Из всего сказанного следует, что (как отмечалось выше) индексы, да и вообще все виды физических путей доступа, являются составной частью реализации, а не модели; их место «под капотом», где они не видны пользователю. (Отмечу, что пути доступа как таковые вообще не упоминаются в реляционной модели.) По тем же причинам их, строго говоря, следовало бы исключить и из SQL. **Рекомендация**: Избегайте любых конструкций SQL, нарушающих эту заповедь. (Насколько я знаю, в стандарте нет ничего, вступающего в противоречие с этой рекомендацией, чего нельзя сказать о некоторых продуктах на основе SQL.)

Как бы то ни было, из приведенных выше определений видно, что различие между моделью и реализацией в действительности является частным – хотя и очень важным – случаем всем знакомого общего различия между логической и физической организацией. Однако, как это ни печально, в большинстве современных SQL-систем оно не проводится так четко, как следовало бы. Отсюда немедленно следует, что они в гораздо меньшей степени физически независимы от данных, чем хотелось бы, и не обеспечивают тех свойств, которые должны присутствовать в реляционной системе. В следующем разделе я еще вернусь к этой теме.

А теперь я хочу обратиться ко второму значению термина *модель данных*, с которым, смею предположить, вы отлично знакомы. Его можно определить следующим образом:

> **Определение**: *Моделью данных* (во втором смысле) называется модель данных (особенно сохраняемых) конкретного предприятия.

Иными словами, модель данных во втором смысле – это просто (логическая и, возможно, до некоторой степени абстрактная) структура базы данных. Например, можно говорить о модели данных банка, больницы или правительственного учреждения.

Объяснив оба смысла, я хочу привлечь ваше внимание к аналогии, которая, как мне кажется, отлично иллюстрирует соотношение между ними.

- Модель данных в первом смысле – аналог языка программирования, конструкции которого применимы для решения многих конкретных задач, но сами по себе не связаны ни с какой отдельной задачей.
- Модель данных во втором смысле – аналог конкретной программы, написанной на этом языке, – в ней используются средства, предоставляемые моделью в первом смысле, для решения конкретной задачи.

Кстати говоря, из сказанного выше следует, что если мы ведем речь о моделях данных во втором смысле, то можем без опасения говорить о «реляционных моделях» во множественном числе или о «некоторой» реляционной модели. Но модель данных в первом смысле только одна, и это *та самая реляционная модель*, которую придумал Кодд. Последнюю мысль я еще разовью в приложении А.

Далее в книге я буду употреблять термин *модель данных*, или для краткости просто *модель*, исключительно в первом смысле.

Свойства отношений

Теперь вернемся к изучению основных понятий реляционной теории. В этом разделе я хочу остановиться на некоторых свойствах самих отношений. Прежде всего, у каждого отношения есть *заголовок* и *тело*. Заголовком называется множество атрибутов (здесь под *атрибутом* я понимаю пару имя-атрибута/имя-типа), а телом – множество кортежей, согласованных с заголовком. Так, у отношения «поставщики» на рис. 1.3 заголовок состоит из четырех атрибутов, а тело – из пяти кортежей. Таким образом, отношение не содержит кортежей – оно содержит тело, которое, в свою очередь, содержит кортежи, – но обычно мы для простоты говорим, что кортежи содержит само отношение.

Попутно отмечу, что хотя правильно было бы говорить, что заголовок состоит из пар имя-атрибута/имя-типа, обычно имена типов на рисунках, подобных рис. 1.3, опускаются, из-за чего создается впечатление, будто заголовок состоит только из имен атрибутов. Например, атрибут STATUS имеет тип, – допустим, INTEGER, – но на рис. 1.3 он не показан. Однако не следует забывать, что он все-таки существует!

Далее, количество атрибутов в заголовке называется *степенью* (или *арностью*), а количество кортежей в теле – *кардинальностью*. Например, для отношений S, P и SP на рис. 1.3 степень равна соответственно 4, 5 и 3, а кардинальность – 5, 6 и 12. *Примечание*: термин *степень* применяется также к кортежам. Так, все кортежи в отношении S имеют степень 4 (как и само отношение S).

Отношения *никогда* не содержат кортежей-дубликатов. Это следует из того, что тело определяется как множество кортежей, а математическое множество по определению не содержит дубликатов. В этом от-

ношении SQL, как вы, конечно, знаете, отступает от теории: таблицы в SQL могут содержать строки-дубликаты и потому в общем случае не являются отношениями. Хочу подчеркнуть, что в этой книге термином *отношение* я всегда называю истинное отношение – без кортежей-дубликатов, – а не таблицу SQL. Кроме того, вы должны понимать, что реляционные операции всегда порождают результат без кортежей-дубликатов, в полном соответствии с определением. Например, проекция отношения «поставщики» на рис. 1.3 на атрибут CITY дает результат, показанный ниже на левом, а не на правом рисунке:

CITY
London
Paris
Athens

CITY
London
Paris
Paris
London
Athens

(Изображенный на левом рисунке результат можно получить SQL-запросом SELECT DISTINCT CITY FROM S. Если опустить слово DISTINCT, то получится нереляционный результат, показанный справа. Особо отметим, что в таблице справа нет двойного подчеркивания, поскольку в ней нет никакого ключа, а уж тем более первичного.)

Далее, кортежи отношения не упорядочены. Это свойство также следует из того, что тело определено как множество, а элементы математического множества не упорядочены (стало быть, {*a*,*b*,*c*} и {*c*,*a*,*b*} в математике считаются одним и тем же множеством, и, естественно, то же справедливо и для реляционной модели). Конечно, когда мы рисуем отношение на бумаге в виде таблицы, мы должны располагать строки сверху вниз, но в таком порядке нет ничего реляционного. Так, строки отношения «поставщики» на рис. 1.3 я мог бы расположить в любом порядке, скажем, сначала S3, потом S1, потом S5, S4 и S2, и это было бы то же самое отношение. *Примечание*: Тот факт, что отношения не упорядочены, еще не означает, что в запрос нельзя включать фразу ORDER BY, но означает, что результат, возвращенный таким запросом, не является отношением. Фраза ORDER BY полезна для представления результатов, но сама по себе не является реляционным оператором.

Аналогично, не упорядочены и атрибуты отношения, поскольку заголовок также является математическим множеством. Изображая отношение в виде таблицы на бумаге, мы по необходимости располагаем столбцы в некотором порядке – слева направо, но в таком порядке нет ничего реляционного. Так, для отношения «поставщики» на рис. 1.3 я мог бы расположить столбцы в любом другом порядке, скажем, STATUS, SNAME, CITY, SNO, и с точки зрения реляционной модели это было бы то же самое отношение (это еще одна причина, по которой таблицы SQL вообще говоря не являются отношениями). Например, на обоих рисунках ниже представлено одно и то же отношение, но разные таблицы SQL:

SNO	CITY
S1	London
S2	Paris
S3	Paris
S4	London
S5	Athens

CITY	SNO
London	S1
Paris	S2
Paris	S3
London	S4
Athens	S5

(В SQL эти представления порождаются соответственно запросами SELECT SNO, CITY FROM S и SELECT CITY, SNO FROM S. Возможно, вам кажется, что различие между этими двумя запросами и таблицами несущественно, но на самом деле последствия весьма серьезны, и о некоторых из них я расскажу в последующих главах. См., например, обсуждение SQL-оператора JOIN в главе 6.)

Наконец, отношения всегда *нормализованы* (или, что то же самое, находятся *в первой нормальной форме*, 1НФ).[1] Неформально это означает, что в табличном представлении отношения на пересечении любой строки и любого столбца мы видим только одно значение. Более формально – каждый кортеж отношения содержит единственное значение соответствующего типа в позиции любого атрибута. В следующей главе я буду говорить об этом гораздо более подробно.

И в заключение я хотел бы еще раз акцентировать ваше внимание на моменте, о котором упоминал неоднократно: существует логическое различие между отношением как таковым и его изображением, например, на рис. 1.1 и 1.3. Повторю, что конструкции, показанные на рис. 1.1 и 1.3, вообще не являются отношениями, а лишь изображениями отношений, которые я обычно называю таблицами, несмотря на то, что это слово уже наделено определенной семантикой в контексте SQL. Конечно, между отношениями и таблицами есть определенное сходство, и в неформальном контексте о них обычно говорят, как об одном и том же, – и, пожалуй, это допустимо. Но если требуется точность – а как раз сейчас я пытаюсь быть точным, – то необходимо понимать, что эти два понятия не идентичны.

Попутно отмечу, что и в более общем плане существует логическое различие между произвольной сущностью и изображением этой сущности. Эта мысль замечательно проиллюстрирована на знаменитой картине Магритта. На картине изображена обычная курительная трубка, но под ней Магритт написал *Ceçi n'est pas une pipe*... – смысл, конечно, в том, что рисунок – это не трубка, а лишь изображение трубки.

Ко всему вышесказанному я хочу добавить, что на самом деле тот факт, что основной абстрактный объект реляционной модели – отношение –

[1] Мы говорим о «первой» нормальной форме, потому что, как вы наверняка знаете, можно определить последовательность нормальных форм «высших порядков» – вторую нормальную форму, третью нормальную форму и т. д., – что существенно для проектирования баз данных. См. упражнение 2.9 в главе 2, а также приложение В.

имеет такое простое представление на бумаге, составляет важное достоинство этой модели; именно наличие простого представления и делает реляционные системы простыми для понимания и использования и позволяет без труда рассуждать об их поведении. Но, к сожалению, это же свойство может приводить к неверным выводам (например, о существовании некоего порядка кортежей – сверху вниз).

И еще один момент: я сказал, что существует логическое различие между отношением и его изображением. Концепция *логического различия* вытекает из следующего афоризма Виттгенштейна:

> Любое логическое различие является существенным различием.

Это замечание необычайно полезно; как «мыслительный инструмент» оно весьма способствует четким и ясным рассуждениям и очень помогает при выявлении и анализе некоторых путаных мест, которые, к несчастью, так часто встречаются в мире баз данных. Я еще не раз вернусь к нему на страницах этой книги. А пока позвольте мне выделить те логические различия, с которыми мы уже столкнулись:

- SQL и реляционная модель
- Модель и реализация
- Модель данных (в первом смысле) и модель данных (во втором смысле)

В последующих трех разделах будут описаны и другие различия.

Базовые и производные отношения

Выше я уже объяснял, что операторы реляционной алгебры позволяют, начав с некоторого набора исходных отношений, – скажем, изображенных на рис. 1.3, – получить новые отношения (например, с помощью запросов). Исходные отношения называются *базовыми*, остальные – *производными*. Следовательно, чтобы было от чего отталкиваться, реляционная система должна предоставлять средства для определения базовых отношений. В SQL эта задача решается предложением CREATE TABLE (в SQL базовому отношению, естественно, соответствует базовая таблица). Очевидно, что базовые отношения должны быть поименованы, например:

```
CREATE TABLE S ... ;
```

Но некоторым производным отношениям, в частности так называемым представлениям, также присваиваются имена. *Представлением* (или *виртуальным отношением*) называется именованное отношение, значением которого в каждый момент времени t является результат вычисления некоторого реляционного выражения в этот момент. Вот как это может выглядеть на языке SQL:

```
CREATE VIEW SST_PARIS AS
   SELECT SNO , STATUS
```