Майкл Дж. Хернандес
Джон Л. Вьескас

SQL-запросы для простых смертных

Практическое руководство по манипулированию данными в SQL

Программно-независимый подход при работе с

- Access
- MS SQL Server
- Oracle
- DB2
- Informix
- Ingres

или любыми другими программами, основанными на SQL, поможет сохранить ваше время и душевное равновесие!

SQL-запросы для простых смертных

Практическое руководство
по манипулированию данными в SQL

Майкл Дж. Хернандес
Джон Л. Вьескас

Издательство "Лори"

Made with ScanKrotes

SQL Queries for Mere Mortals

A Hands-On Guide to Data Manipulation in SQL

Michael J. Hernandez
John L. Viescas

ЭЛЕКТРОННАЯ БИБЛИОТЕКА
Прочитал? Удали!
LIBRUS.RU

ADDISON-WESLEY

Boston • San Francisco • New York • Toronto • Montreal
London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Open with ScanKroms

Предисловие

Книга *SQL-запросы для простых смертных* является превосходным введением в запросы SQL и хорошо дополняет предыдущую книгу *Проектирование базы данных для простых смертных*, вышедшую в издательстве Addison-Wesley. Можно сказать, что это даже более хорошее введение, чем первая книга. Реальный программист (т. е. простой смертный) тратит больше времени на написание запросов SQL и меньше — на проектирование самой базы данных. Схемы составляют высокооплачиваемые администраторы баз данных, владеющие отдельными кабинетами и спортивными автомобилями. Большинство программистов занимаются тем, что пытаются заставить SQL работать в условиях жестко заданной схемы.

В силу своей профессии я настраиваю базы данных и преподаю расширенный SQL, поэтому могу подтвердить, что большинство текстов на SQL настолько же привлекательны для чтения, как и египетские иероглифы. Как только программа начинает работать, программист, написавший ее, переходит к решению следующей задачи, никогда не оглядываясь на то, что было сделано. И когда с этой программой что-то не так, другой человек посылает отчаянные сообщения дискуссионным группам в Интернет, где Джон и Майк спасут его несколькими мудрыми словами и переписыванием кода. Они годами помогали людям решать их проблемы, связанные с SQL. Наконец настало время изложить все это в книге, которой может воспользоваться каждый!

Совсем нетрудно и не требуется много времени, чтобы написать хорошую программу. Если вы понимаете, что делаете, то большинство проблем решается совсем просто. Прежде всего необходимо изучить основы. Данная книга предоставляет вам шанс изучить эти основы в понятной и хорошо написанной форме. Затем необходимо понять, когда и как приспособить простое решение к конкретной СУБД и конкретной реализации SQL. Когда основы будут надежно усвоены, обратитесь ко мне, и я научу вас действительно нетривиальным вещам.

Джо Келко
Атланта, штат Джорджия

Содержание

Предисловие *iv*
Предисловие и благодарности *ix*
Об авторах *xi*
Введение *xiii*

Часть I • Реляционные базы данных и SQL 1

Глава 1 • Что такое “реляционный”? 3

Типы баз данных . . . 3
Краткая история реляционной модели . . . 4
Анатомия реляционных баз данных . . . 5
Зачем все это нужно . . . 14
Итоги . . . 16

Глава 2 • Обеспечение надежности структуры базы данных 17

Почему эта глава помещена здесь . . . 17
Зачем нужна хорошо продуманная структура . . . 18
Настройка полей . . . 18
Настройка таблиц . . . 26
Установка и исправление связей . . . 36
И это все? . . . 43
Итоги . . . 43

Глава 3 • Краткая история SQL 45

Истоки SQL . . . 45
Ранние реализации . . . 47
“... а затем был Стандарт” . . . 48
Развитие стандарта ANSI/ISO . . . 49
Что готовит будущее . . . 54
Зачем изучать SQL . . . 56
Итоги . . . 57

Часть II • Основы SQL 59

Глава 4 • Создание простых запросов 61

Знакомство с SQL 61
Оператор SELECT 62
Краткое отступление: Данные в сравнении с информацией 64
Перевод запроса на SQL 66
Исключение дубликатов строк 72
Сортировка информации 74
Сохранение работы 79
Примеры операторов 79
Итоги 87
Задачи для самостоятельного решения 88

Глава 5 • Как получить нечто большее, чем просто столбцы 91

Условие SELECT, дубль два 92
За пределами азов 96
Что такое “выражение” 97
Что вы пытаетесь выразить 97
Типы выражений 100
Использование выражений в условии SELECT 109
Значение Null 117
Примеры операторов 120
Итоги 127
Задачи для самостоятельного решения 128

Глава 6 • Фильтрация данных 131

Уточнение полученного с использованием WHERE 131
Определение условий поиска 135
Использование нескольких условий 156
Повторная встреча с NULL: Предупреждающее замечание 168
Выражение условий различными способами 172
Примеры операторов 173
Итоги 179
Задачи для самостоятельного решения 179

Часть III • Работа с несколькими таблицами 183

Глава 7 • Мышление множествами 185

Что такое множество 186
Операции над множествами 186

Пересечение 187
Разность 192
Объединение 199
Операции с множествами в SQL 203
Итоги 212

Глава 8 • Внутренние соединения 214

Что такое JOIN 214
INNER JOIN 215
Применения условий INNER JOIN 231
Примеры операторов 233
Итоги 251
Задачи для самостоятельного решения 251

Глава 9 • Внешние соединения 255

Что представляет собой OUTER JOIN 255
LEFT/RIGHT OUTER JOIN 257
FULL OUTER JOIN 276
Использование операций OUTER JOIN 281
Примеры операторов 282
Итоги 295
Задачи для самостоятельного решения 296

Глава 10 • Операции UNION 298

Что представляет собой UNION 298
Запись запросов с UNION 300
Применение UNION 311
Примеры операторов 312
Итоги 322
Задачи для самостоятельного решения 322

Глава 11 • Подзапросы 325

Что представляет собой подзапрос 326
Подзапросы как выражения со столбцами 327
Подзапросы как фильтры 332
Использование подзапросов 347
Примеры операторов 349
Итоги 361
Задачи для самостоятельного решения 362

Часть IV • Суммирование данных и объединение в группы 365

Глава 12 • Простая сумма 367

Агрегатные функции 367
Использование агрегатных функций в фильтрах 381
Примеры операторов 384
Итоги 390
Задачи для самостоятельного решения 390

Глава 13 • Группирование данных 393

Зачем нужно группировать данные. 393
Условие GROUP BY 395
Наложение некоторых ограничений 404
Использование GROUP BY 408
Примеры операторов 409
Итоги 417
Задачи для самостоятельного решения 418

Глава 14 • Фильтрация сгруппированных данных 420

Сужение групп 420
Фильтры: Почувствуйте разницу 425
Использование HAVING 432
Примеры операторов 433
Итоги 441
Задачи для самостоятельного решения 441

Заключение 445

Приложения 447

Приложение А • Диаграммы Стандарта SQL 449

Приложение В • Структуры баз данных, использованных в качестве примеров 455

Приложение С • Литература, рекомендуемая для чтения 459

Предисловие и благодарности

"Язык, по самой своей сути, является средством общения;
т. е. он никогда не выражает точные вещи,
но является компромиссом — обычным
для вас, меня и всех остальных".

— Томас Эрнст Халм, *Размышления*

Изучение способов извлечения информации из базы данных — обычно дело непростое. Однако оно может оказаться относительно легкой задачей, как только станет понятен вопрос, с которым следует обратиться к базе данных. Когда вопрос понятен, его можно перевести на язык, используемый любой системой управления базами данных (СУБД),— в большинстве случаев это структурированный язык запросов, или SQL (Structured Query Language). Вам нужно преобразовать свой запрос в оператор SQL таким образом, чтобы СУБД знала, какая информация вам нужна. SQL предоставляет вам и вашей базе данных средства для обмена информацией друг с другом.

Работая многие годы консультантами по базам данных, мы обнаружили, что количество людей, которым требуется извлекать информацию из базы данных, превосходит по численности тех, кто создает программы и приложения для БД. К сожалению, ни одна из книг не сосредоточивается исключительно на извлечении информации, в частности с точки зрения "простых смертных". Мы уверены, что существует множество хороших книг по SQL, но большинство из них нацелено на программирование и разработку БД.

С учетом этого мы решили, что настало время написать книгу, которая поможет научиться правильно и эффективно составлять запросы к базе данных. Результат этого решения — в ваших руках. Эта книга уникальна среди книг по SQL тем, что она *рассматривает только ту часть SQL, которая связана с запросами.* Дочитав ее до конца, вы приобретете навыки, необходимые для извлечения любой нужной информации.

Написание такой книги, как эта, всегда требует совместных усилий. Всегда существуют редакторы, коллеги, друзья и родственники, желающие оказать поддержку и дать ценный совет. Эти люди постоянно нас поддерживали, помогали концентрироваться и служили стимулом для того, чтобы довести этот проект до конца.

В первую очередь мы хотели бы поблагодарить нашего редактора Мэри О'Брайен за возможность написать эту книгу. Она видела потенциальные возможности нашей идеи и полностью посвятила себя ее достижению. Благодарим Мэри и ее ассистентку, Марианну Курафас, за их огромное терпение и неизменную поддержку в течение многих месяцев. И мы не можем забыть Мэрилин Раш и выпускающую команду — прекрасная работа, друзья!

Выражаем признательность техническим редакторам — Малькольму С. Раблу, Майклу Блэха, Александру Таразулу и Кейту В. Хейру. Малькольм, как всегда, замечательно — иметь вас в своей команде! Майкл и Александр, спасибо за все ваши заботливые комментарии и предложения. И отдельные благодарности Кейту: он исправил несколько небольших ошибок в описании истории SQL и предоставил много информации для раздела "Что готовит будущее" главы 3. Еще раз благодарим всех за потраченное время и вклад, за оказанную нам помощь в создании этого солидного трактата по запросам SQL.

Наконец, отдельная благодарность Джо Келко за предоставленное предисловие. Джо является экспертом по SQL, коллегой и хорошим другом. Мы испытываем огромное уважение к знаниям и компетенции Джо по данному вопросу и рады тому, что его мысли и комментарии приведены в начале нашей книги.

Приношу свои самые теплые благодарности моему дорогому другу и коллеге Джону Л. Вьескасу за возможность совместной работы над этой книгой. Именно у Джона появилась первоначальная идея, и однажды вечером он уговорил меня писать ее вместе с ним. Джон длительное время занимался такой деятельностью и является признанным, уважаемым автором. Для меня честь — быть его соавтором в этой работе.

Наконец, еще раз благодарю мою жену Кендру за исключительное терпение, проявленное ею в то время, пока я был занят тяжким трудом по написанию книги. Ее помощь была бесценной, и к тому же мне хотелось бы отдать свой большой долг и сказать ей, что она является единственной любовью всей моей жизни, самым ближайшим наперсником и самым лучшим другом, но она не выносит каких-либо публичных проявлений чувств (и считает это излишеством, таким же, как карманные компьютеры). Поэтому я просто закончу следующим:

Кед, теперь мы снова будем жить как нормальные люди — до следующей книги!

Майкл Дж. Хернандес
Беллевью, штат Вашингтон

Вот здорово, Майк! Таким образом ты решил меня подбодрить? Не стоит при этом принижать свои заслуги. Я мог уговорить тебя написать "следующую" книгу по SQL *для простых смертных*, но ты относишься к тем, кто изобрел этот, теперь уже хорошо проверенный, формат и "голос" для аудитории. Это была забавная и интересная задача — преподнести сложный мир SQL (одна из моих излюбленных тем) такой широкой аудитории. Спасибо за то, что ты пригласил меня принять участие в этом проекте.

В отличие от твоей, моя жена Сьюзен не питает отвращения к карманным компьютерам. Мы оба должны крепко обнять ее — не только за то, что она заботилась обо мне, пока я "отсутствовал", занимаясь этой книгой, но также за исключительные, оставшиеся "за сценой", редакционные и критические комментарии нашего материала. Она — не посторонний человек, когда подходит к компьютерам, но определенно "простой смертный", когда это касается баз данных. Она была прекрасной слушательницей для тестирования черновиков. Я только должен дать ей одно обещание: когда мы следующей весной снова поедем на Гавайи, ноутбук я оставлю дома!

Джон Л. Вьескас
Остин, штат Техас

Об авторах

Джон Л. Вьескас и Майкл Дж. Хернандес

Майкл Дж. Хернандес — опытный разработчик баз данных, с более чем 13-летним стажем разработки приложений для огромного разнообразия клиентов в различных отраслях. Майкл специализируется по разработке реляционных баз данных и является автором ставшей бестселлером книги *Проектирование баз данных для простых смертных* (Database Design for Mere Mortals, Addison-Wesley, 1997). Он работал с SQL в течение всей своей карьеры, разрабатывал приложения, используя такие базы данных (основанные на SQL), как R:BASE компании Microrim, Microsoft Access и теперь — Microsoft SQL Server. Майкл также является автором многочисленных статей и техническим редактором различных книг и периодических изданий по Access.

Помимо того, что он продолжает работать над многими проектами по разработке базы данных и много пишет, Майкл путешествует по стране, преподавая Microsoft Access в компании AppDev (бывшая Application Developers Training Company), и неизменно получает самый высокий рейтинг среди преподавателей. Он также ведет четырехдневные курсы по проектированию реляционных баз данных, основанные

на методах проектирования, представленных в данной книге. Майкл выступает на различных конференциях, в частности на выставке-конференции по решениям для Microsoft Office и VBA 1999 г. и на конференции по развертыванию и разработке сред Microsoft Office 2000 г.

Раньше Майкл был музыкантом и выступал перед публикой. Его характеризует как легкий стиль инструкций, так и способность установить контакт с аудиторией, как в те дни, когда он был исполнителем. До сих пор его можно застать с гитарой, играющим самые новые рок-мелодии. Майклу доставляют удовольствие такие мелочи, как время, проведенное в Barnes & Noble за бокалом коктейля и хорошей сигарой и путешествия на горных велосипедах со своей женой Кендрой.

Вы можете посетить Web-страницу Майкла по адресу: *www.datatexcg.com* или связаться с ним по электронной почте: *mjhernandez@msn.com*.

Джон Вьескас является независимым консультантом с более чем 35-летним опытом. Свою карьеру он начал как системный аналитик, проектируя большие приложения баз данных для систем на универсальных вычислительных машинах IBM. Он провел шесть лет в компании Applied Data Research в Далласе, где руководил персоналом из более чем 30 человек и отвечал за исследования, разработку продуктов и поддержку клиентов для продуктов баз данных для универсальных вычислительных машин IBM. Работая в Applied Data Research, Джон *с отличием* окончил Техасский университет в Далласе и получил степень по финансовому управлению.

В 1988 г. Джон перешел на работу в компанию Tandem Computers, Inc., где отвечал за разработку и внедрение программ по маркетингу баз данных Tandem в западных штатах США. Он разрабатывал и проводил технические семинары по реляционной СУБД компании Tandem, NonStop SQL, от Гавайских островов до Колорадо и от Аляски до Аризоны. Джон написал свою первую книгу *Краткое справочное руководство по SQL* (A Quick Reference Guide to SQL, Microsoft Press) в качестве исследовательского проекта для документирования сходных элементов в синтаксисе стандарта SQL ANSI 86, IBM DB2, Microsoft SQL Server, Oracle и Tandem NonStop SQL. Первый вариант книги *Работа с Microsoft Access* (Running Microsoft Access, Microsoft Press) он написал во время ежегодного отпуска и с тех пор выпустил еще четыре издания.

В 1993 г. Джон учредил свою собственную компанию. Он консультировал различные предприятия в Техасе по вопросам управления информационными системами, специализируясь на Microsoft Access и продуктах управления базами данных SQL Server. Офис компании находится в Остине, штат Техас. Он написал множество статей для технических изданий, таких как *Smart Access* и *Access Advisor*. Джон выступал с докладами на конференциях и на собраниях групп пользователей по всему миру, включая закрытые заседания на конференциях Microsoft Tech*Ed и European WinSummit. Начиная с 1993 г. служба поддержки продуктов Microsoft присуждает ему титул "Самый ценный профессионал" за помощь в решении технических вопросов на форумах общественной поддержки.

Вы можете посетить Web-сайт Джона на *www.viescas.com* или связаться с ним по электронной почте: *johnv@viescas.com*.

Введение

"Полагаю, что вы человек и можете ошибаться!"

— Джеймс Ширли, *Куртизанка*

Если вы пользуетесь компьютером постоянно, то вам наверняка приходилось иметь дело со структурированным языком запросов, или SQL (Structured Query Language), хотя, возможно, вы даже не догадывались об этом. Каждый раз, импортируя данные в электронную таблицу или выполняя слияние в программе обработки текста, вы, скорее всего, использовали в той или иной форме SQL. Когда вы выходите в интерактивном режиме на сайт электронной торговли в Web и размещаете заказ на любой продукт, весьма вероятно, что программа, применяемая на Web-сайте, осуществляет доступ к своим базам данных через SQL. Если вам необходимо получать информацию от базы данных, использующей SQL, то, прочитав эту книгу, вы расширите свое понимание языка.

Являетесь ли вы "простым смертным"

Вы можете спросить "А я — простой смертный или нет?". Ответ не так уж прост. Когда мы начинали писать эту книгу, мы считали себя "экспертами" по языку баз данных SQL. По пути мы открыли, что в некоторых областях мы также были "простыми смертными". Мы очень хорошо понимали несколько конкретных реализаций SQL, но нам пришлось пройти много сложных лабиринтов языка, пока мы изучали, как он используется во многих коммерческих продуктах. Поэтому, если вы подходите под следующие описания, вы также простой смертный!

- Если вы пользуетесь компьютерными приложениями, которые предоставляют доступ к информации из базы данных, то вы, вероятно, "простой смертный". В первый раз, когда вы не получите ожидаемую информацию с помощью средств запросов, встроенных в приложение, вам потребуется исследовать исходные операторы SQL для обнаружения причины.
- Если вы недавно начали работать с любым из множества существующих приложений баз данных для настольных компьютерных систем, но затрудняетесь с определением и запросом необходимых данных, то вы — "простой смертный".

- Если вы — программист баз данных, которому требуется "продумать все до конца" для решения каких-то сложных проблем, то вы — "простой смертный".
- Если вы считаете себя "гуру" по той или иной СУБД, а теперь столкнулись с интегрированием данных из существующей системы в другую систему, которая поддерживает SQL, то вы — "простой смертный".

Короче, *любой*, кому приходится работать с базами данных, поддерживающими SQL, может найти в этой книге много полезного. Начинающим пользователям эта книга предоставит все необходимые азы и многое другое. "Эксперту", неожиданно столкнувшемуся с необходимостью решения сложных проблем или с интеграцией нескольких систем, поддерживающих SQL, книга поможет разобраться в средствах достижения цели при использовании возможностей языка баз данных SQL.

Несколько слов о книге

Все, что вы прочтете в данной книге, основано на текущем стандарте Американского национального института стандартов (ANSI) (документ ANSI X3.135-1992), который в настоящее время реализован в большинстве популярных коммерческих систем баз данных. Документ ANSI также был принят в качестве стандарта ISO/IEC (International Organization for Standardization/International Electrotechnical Commission) и опубликован как документ ISO/IEC 9075:1992. Следовательно, это действительно международный стандарт. Излагаемый здесь SQL *не является* диалектом языка для какого-либо определенного программного продукта.

Стандарт SQL определяет как больше, так и меньше того, что реализовано в большинстве коммерческих продуктов баз данных. Поставщики баз данных еще должны реализовать многие из более новаторских возможностей, но большинство поддерживает ядро данного стандарта.

Когда вы столкнётесь с теми частями ядра языка, которые не поддерживаются некоторыми главными продуктами, мы предупредим вас об этом и покажем альтернативные способы изложения запросов к базе данных в стандартном SQL. Если значительные части стандарта поддерживаются только несколькими поставщиками, мы кратко описываем синтаксис, а затем предполагаемые альтернативы.

Книга состоит из четырех разделов:

- Часть I, "Реляционные базы данных и SQL", поясняет, как современные базы данных основываются на строгих математических моделях. Здесь описывается краткая история языка запросов к базам данных, который развился в то, что теперь нам известно как SQL.
- Часть II, "Основы SQL", знакомит с оператором SELECT, с созданием выражений и сортировкой информации в условии ORDER BY.
- Часть III, "Работа с несколькими таблицами", показывает, как формулировать запросы для извлечения данных из более чем одной таблицы.

- Часть IV, "Суммирование и объединение данных в группы", описывает, как получить общую информацию и сгруппировать и отфильтровать обобщенные данные.

В приложениях в конце этой книги приведены синтаксические диаграммы для всех элементов SQL, структуры примеров баз данных и книги, рекомендуемые для дальнейшего изучения SQL. На сайте издательства "Лори" вы найдете все примеры баз данных, используемые в книге.

Как работать с книгой

Главы книги составлены так, чтобы читать их по порядку, и каждая последующая глава построена на концепциях, изложенных в предыдущих главах.

В конце многих глав даны примеры задач, их решения и примеры наборов результатов. Мы рекомендуем изучить некоторые из них, а затем попытаться решить остальные примеры самостоятельно.

Когда отдельный запрос возвращает десятки строк в наборе результатов, мы показываем в книге только несколько первых строк, чтобы дать представление о том, как должен выглядеть ответ. Возможно, однако, что в своей системе вы не увидите точно тот же результат, поскольку каждая СУБД, поддерживающая SQL, располагает своим собственным оптимизатором, который вычисляет самый быстрый способ разрешения запроса. К тому же первые несколько строк, возвращенные вашей базой данных, могут не соответствовать нескольким первым строкам, показанным нами, если только запрос не содержит условие ORDER BY, которое требует возврата строк в определенной последовательности.

В книгу включен полный набор задач для самостоятельного решения, которые располагаются сразу же после раздела "Итоги" в каждой главе.

Полный набор диаграмм SQL в приложении A является ценным справочником для всех показанных методов SQL. Также можно воспользоваться примерами схем баз данных в приложении B, которые помогут вам проектировать свои собственные базы данных.

Как читать диаграммы

Множество диаграмм иллюстрирует соответствующий синтаксис операторов, термины и фразы, используемые при работе с SQL. Каждая диаграмма дает четкую картину общей структуры элемента SQL, обсуждаемого в настоящий момент. Вы можете использовать любую из этих диаграмм в качестве образца для создания своего собственного оператора SQL.

Все диаграммы строятся из набора основных элементов, которые можно разделить на две категории: *операторы* и *определяемые термы*. Оператор всегда является основной операцией SQL, как, например, оператор SELECT. Определяемые термы всегда являются компонентами, используемыми для построения части

оператора, как, например, *выражение*, *условие поиска* или *условное выражение*. Единственное различие между синтаксической диаграммой для оператора и синтаксической диаграммой для определяемого терма является способ, используемый для начала и конца основных синтаксических линий. На рис. 1 показаны начальные и конечные точки для каждой из категорий диаграмм. В остальном диаграммы строятся из одних и тех же элементов. На рис. 2 представлены примеры каждого из типов синтаксических диаграмм и дается краткое пояснение каждого элемента.

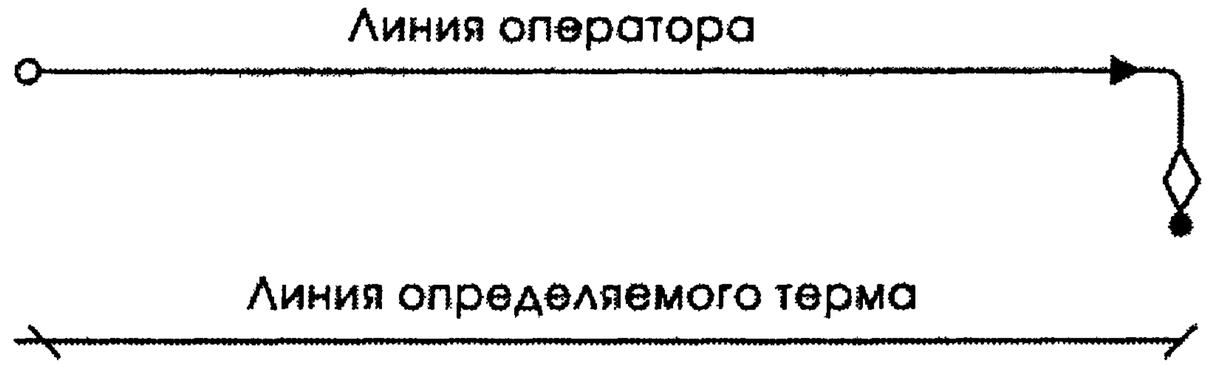


Рис. 1. *Конечные точки синтаксических линий для операторов и определяемых термов*

1. *Начальная точка оператора* — обозначает начало главной синтаксической линии для оператора. Любой элемент, появляющийся непосредственно на главной синтаксической линии, является обязательным элементом, а любой элемент, который появляется ниже, считается необязательным элементом.
2. *Главная синтаксическая линия* — определяет порядок всех обязательных и необязательных элементов для оператора или определяемого терма.

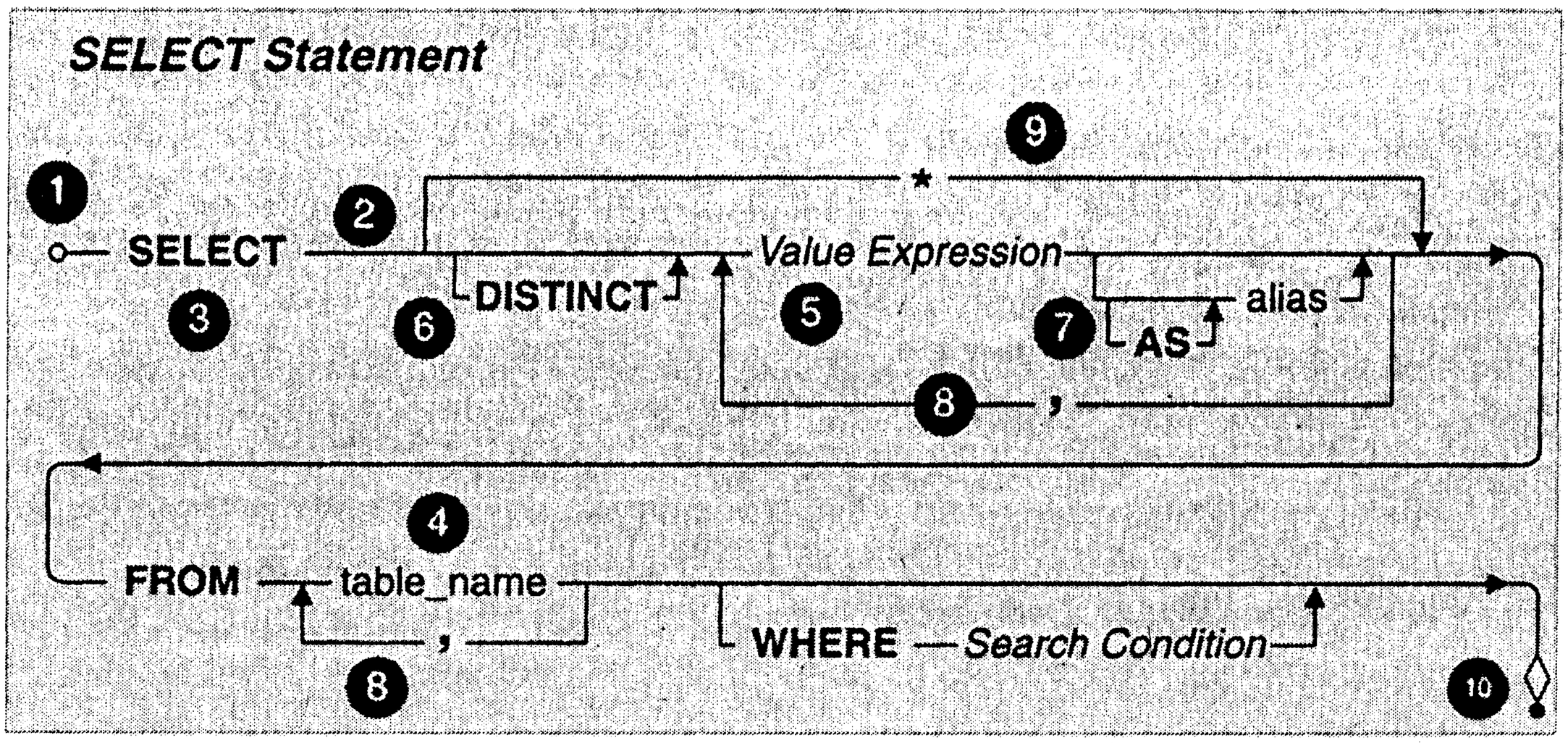


Рис. 2. *Пример диаграмм оператора и определяемого терма*

Чтобы построить синтаксис оператора или определяемого терма, нужно следовать по этой линии слева направо (или в направлении стрелок).

3. *Ключевое слово* (*слова*) — основное слово в грамматике SQL, которое является обязательной частью синтаксиса для оператора или определяемого терма. На диаграмме ключевые слова представлены заглавными буквами и жирным шрифтом. (При реальном вводе оператора в вашей программе для базы данных не обязательно вводить ключевые слова заглавными буквами, но это облегчает чтение оператора.)

4. *Буквенная запись* — имя значения, которое явно вводится в оператор. Буквенная запись представляется в виде слова или фразы, указывающих тип значения, которое требуется ввести. На диаграмме она представляется любыми буквами нижнего регистра.

5. *Определяемый терм* — Слово или фраза, представляющие некоторую операцию, которая возвращает окончательное значение, используемое в этом операторе. Мы представляем диаграмму для каждого определяемого терма, который требуется знать при работе с данной книгой. Определяемые термы всегда выделяются курсивом.

6. *Необязательный элемент* — Любой элемент или группа элементов, которые располагаются ниже основной синтаксической линии. Необязательным элементом может быть оператор, ключевое слово, определяемый терм или литеральное значение, которые для большей ясности помещаются на своей собственной линии. В некоторых случаях можно определить ряд значений для указанной опции, причем каждое значение выделено запятой (см. п. 8). Несколько необязательных элементов имеют множество дополнительных необязательных элементов (см. п. 7). В общем случае синтаксическая линия для необязательных элементов читается слева направо, как и основная синтаксическая линия. Всегда следуйте стрелкам, указывающим направление, и все будет в порядке. Обратите внимание, что некоторые опции позволяют определить несколько значений или вариантов, так что стрелки могут указывать направление справа налево. Тем не менее, как только вы ввели все необходимые элементы, направление движения возвращается к обычному: слева направо. К счастью, все необязательные элементы действуют одинаково.

7. *Дополнительный необязательный элемент* — Любой элемент или группа элементов, которые появляются ниже необязательного элемента. Эти элементы позволяют выполнить тонкую настройку операторов, чтобы можно было решить наиболее сложные проблемы.

8. *Разделитель списка опций* — Указывает, что для этой опции можно определить больше одного значения и что каждое значение должно быть отделено запятой.

9. *Альтернативная опция* — Указывает ключевое слово или определяемый терм, который можно использовать как альтернативный вариант для одного или нескольких необязательных элементов. Синтаксическая линия для альтернативной опции будет обходить синтаксические линии необязательных элементов, подразумевая возможность замены.

10. *Конечная точка оператора* — Обозначает конец главной синтаксической линии для оператора.

11. *Начальная точка определяемого терма* — Обозначает начало основной синтаксической линии для определяемого терма.

12. *Конечная точка определяемого терма* — Обозначает конец главной синтаксической линии для определяемого терма.

После ознакомления с этими элементами вы сможете читать все синтаксические диаграммы. Когда диаграмма потребует дополнительных пояснений, мы предоставим информацию, которая поможет легко ее прочитать. Чтобы помочь лучше понять, как работает диаграмма, покажем пример оператора SELECT, построенного с использованием приведенной диаграммы.

```
SELECT FirstName, LastName, City, DOB AS DateOfBirth
FROM Students
WHERE City = 'El Paso'
```

Оператор SELECT извлекает четыре столбца из таблицы Students, как указано в SELECT и условии FROM. Следуя вдоль основной синтаксической линии слева направо, можно увидеть, что необходимо указать по крайней мере одно *выражение*. Оно может представлять собой имя столбца, и можно указать столько столбцов, сколько необходимо, с *разделителем списка опций* выражения значений. Это касается использования четырех имен столбцов из таблицы Students. Нас беспокоило, что некоторые люди, просматривавшие информацию, возвращенную этим оператором SELECT, могли не знать, что означает DOB, и поэтому мы присвоили *псевдоимя* столбцу DOB в подопции AS выражения значений. Наконец, мы воспользовались условием WHERE для уверенности в том, что оператор SELECT покажет только тех студентов, которые живут в Эль-Пасо. (Если сейчас вы мало что поняли, не беспокойтесь: в дальнейшем все будет подробно разъяснено.)

Полный набор синтаксических диаграмм можно найти в приложении A. В них представлен полный и надлежащий синтаксис для всех операторов и определяемых термов, обсуждаемых в этой книге. Однако, если во время чтения вам потребуется ссылка на эти диаграммы, вы заметите небольшое несоответствие между некоторыми диаграммами в тексте и соответствующими диаграммами в приложении. Просто диаграммы в главе являются упрощенными версиями диаграмм в приложении.

Примеры баз данных, используемых в книге

На сайте издательства "Лори" (www.lory-press.ru) находится пять учебных баз данных, используемых для примеров запросов в данной книге. Также в приложение В включены диаграммы структур баз данных: схема примеров баз данных.

1. **Заказ на закупку.** Это типичная база данных записей заказов для магазина, продающего мотоциклы и аксессуары.
2. **Агентство эстрадных мероприятий.** Мы сформировали эту базу данных для того, чтобы руководить эстрадными артистами, агентами, клиентами и продажей билетов. Подобную схему можно использовать для управления продажей билетов на мероприятия или для бронирования мест в гостинице.
3. **Расписание учебного заведения.** Можно воспользоваться этой схемой для регистрации учеников в средней школе или в государственном колледже. Эта база данных отслеживает не только регистрацию по классам, но и преподавателей, назначенных для каждого класса, и оценки учеников.
4. **Лига игроков в боулинг.** Эта база данных отслеживает команды игроков в боулинг, участников команд, сыгранные матчи и результаты.
5. **Рецепты**. Эту базу данных можно использовать для сохранения и управления всеми своими рецептами. Мы даже добавили несколько, которые вы, возможно, захотите попробовать.

В корневой папке на сайте вы найдете все пять примеров баз данных в трех вариантах. Вследствие большой популярности СУБД Microsoft Access, предназначенной для настольных ПК, мы создали один набор баз данных (с расширением файла .mdb), используя Microsoft Access 2000 (версия 9.0). Мы выбрали версию 9, потому что она более тесно поддерживает текущий стандарт ANSI для SQL, чем любая предыдущая версия. Второй вариант набора состоит из файлов базы данных (с расширением файла .mdf) для Microsoft SQL Server версии 7. Мы также включили командные файлы SQL (с расширением файла .sql), которые можно использовать для присоединения примеров к каталогу Microsoft SQL Server. В качестве дополнительного приза мы включили пробную версию Microsoft SQL Server версии 7, которую можно инсталлировать на любой компьютер с работающими Microsoft Windows 95, 98, 2000 или рабочей областью NT. Вы можете воспользоваться этой пробной версией для работы со всеми примерами в книге. В корневом каталоге, а также в каталогах для каждой из глав имеется файл README.TXT, содержащий все необходимые указания о том, как загрузить примеры баз данных. Для успешной загрузки примеров очень важно прочитать эти файлы.

В текстовых файлах можно найти третью версию, содержащую команды SQL для построения баз данных и загрузки данных в свою систему. Хотя мы очень старались использовать наиболее общий и простой синтаксис для команд CREATE TABLE

и INSERT, вам (или администратору базы данных), возможно, потребуется немного изменить эти файлы, чтобы заставить их работать в вашей системе базы данных. Мы не включали какие-либо команды для создания индекса для всех таблиц, потому что синтаксис для CREATE INDEX значительно изменяется в зависимости от системы базы данных. Может оказаться, что многие примеры операторов выполняются очень медленно вследствие отсутствия индексов. Обратитесь к документации по базе данных за информацией о том, как создавать индексы для повышения производительности. При работе с системой базы данных на удаленном сервере, возможно, потребуется получить необходимые полномочия у своего администратора БД для построения примеров из предложенных команд SQL.

Также на сайте можно найти вложенную папку, содержащую раздел "Примеры операторов". Примеры данных помогут облегчить поиск примеров для каждой главы. В некоторых случаях мы изменили данные примеров, чтобы все примеры операторов возвращали хотя бы одну строку в наборе результатов. В этих вложенных папках можно непосредственно использовать Microsoft Access 2000 или присоединить версии Microsoft SQL Server. Если вы работаете с другой базой данных, то найдете один набор текстовых файлов, содержащих SQL для загрузки данных примера, и другой набор файлов с SQL для примеров операторов.

"Иди по дороге из желтого кирпича"

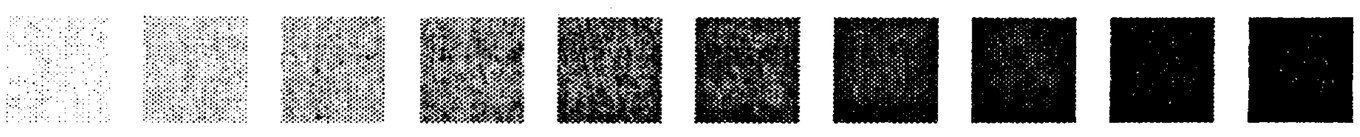
— советует Жевун Дороти ("Волшебник страны Оз")

Теперь можно перейти к изучению SQL, не так ли? В это мгновение вы все еще в домике, его все еще швыряет торнадо, и вы пока еще в Канзасе.

Прочтите первые три главы. В главе 1 мы расскажем о том, как возникли реляционные базы данных и как они выросли до наиболее широко используемых сегодня в отрасли типов баз данных. Это позволит вам приобрести общее понятие о СУБД, используемой вами в настоящее время. В главе 2 показано, как осуществить тонкую настройку своей структуры данных, чтобы данные были достоверны и точны. Возможно, вам трудно работать с некоторыми операторами SQL, если структуры данных были плохо спроектированы.

Собственно "дорога из желтого кирпича" начинается в главе 3. В ней изучается происхождение SQL и процесс развития до текущего состояния, а также говорится о некоторых людях и компаниях, которые помогли проложить дорогу языку, и об особенностях SQL. Наконец, в этой главе объясняется, как SQL стал национальным и международным стандартом и какой вид он примет в ближайшем будущем.

Как только вы прочтете эти главы, считайте, что вы на прямом пути в страну Оз. Просто следуйте по проложенной нами дороге через все оставшиеся главы. Когда вы закончите чтение книги, вы поймете, что нашли Волшебника — и сможете увидеть его в зеркале.



Часть I

Реляционные базы данных и SQL

1

Что такое "реляционный"?

"Знание — это лишь та небольшая часть незнания,
которую мы систематизировали
и классифицировали".

— Амброуз Бирс

Вопросы, рассматриваемые в данной главе:

- Типы баз данных
- Краткая история реляционной модели
- Анатомия реляционной базы данных
- Зачем все это нужно
- Итоги

Прежде чем погрузиться в изучение сути SQL, нужно усвоить некоторую вводную информацию о реляционной БД. Эта информация обеспечит азы, необходимые для действительного понимания того, что такое SQL, и поможет разъяснить, как можно использовать SQL для достижения поставленных целей с наибольшей выгодой.

Типы баз данных

Что же такое "база данных"? Как, возможно, вам известно база данных представляет собой упорядоченный набор данных, используемый для моделирования некоторого типа организации или организационного процесса. В сущности, совсем неважно, используется ли для сбора и хранения данных бумага или компьютерная программа. Когда осуществляется сбор и хранение данных некоторым упорядоченным образом для конкретной цели, получается база данных. Мы будем использовать для сбора и хранения данных компьютерную программу.

Практически все БД делятся на два типа: *операционные* и *аналитические*.

Операционные базы данных являются сегодня стержнем многих компаний, организаций и учреждений. Этот тип баз данных главным образом используется для повседневного сбора, изменения и сохранения данных. Сохраняемые данные являются *динамическими*. Это означает, что они постоянно изменяются и всегда отражают самую

последнюю информацию. Такие организации, как розничные магазины, производственные компании, больницы, клиники и издательства используют операционные базы данных, потому что их данные находятся в состоянии постоянного изменения.

В отличие от них аналитические базы данных сохраняют и отслеживают ретроспективные данные и динамику их изменения во времени. Аналитическая база данных является ценным активом для отслеживания тенденций, просмотра статистических данных за длительный период времени или для создания тактических, стратегических бизнес-планов. Тип сохраняемых данных — *статический*. Это означает, что данные никогда (или очень редко) не изменяются. Информация, собранная из аналитической базы данных, отражает моментальный снимок данных для определенного момента времени и обычно не является самой последней. Химические лаборатории, геологические компании и фирмы, занимающиеся маркетинговыми исследованиями, являются примерами организаций, использующих аналитические базы данных.

Краткая история реляционной модели

Существует несколько типов моделей баз данных. Некоторые из них, например иерархические и сетевые, крайне неудобны в работе, тогда как другие, такие как реляционные, получили широкое признание. В книгах также обсуждаются “объектные” или “объектно-реляционные” модели. Поскольку эти более новые модели подвергаются в настоящее время дальнейшему исследованию и разработке, мы полагаем, что они, скорее всего, будут реализованы как расширения реляционной модели. Фактически, большая часть работы комитета ANSI над следующей версией стандарта SQL включает расширение SQL, охватывающее объектно-ориентированные концепции. Но для наших целей мы сосредоточимся только на реляционной модели.

В самом начале....

Реляционная база данных впервые появилась на свет в 1969 г. и бесспорно стала сегодня наиболее широко используемой моделью базы данных в системах управления базами данных. Отец реляционной модели, д-р Эдгар Ф. Кодд, был в конце 60-х годов ученым-исследователем и занимался поисками новых способов управления большими объемами данных. Неудовлетворенность моделями и продуктами баз данных того времени заставила его задуматься о способах применения математических дисциплин и структур для решения несметного числа проблем, с которыми он сталкивался. Математик по профессии, он полагал, что сможет применить специальные разделы математики для решения таких проблем, как дублирование данных, плохая целостность данных и сверхзависимость структур баз данных от их физической реализации.

Д-р Кодд официально представил свою новую реляционную модель в эпохальной работе, озаглавленной “A Relational Model of Data for Large Shared Databanks (Communications of the ACM, June 1970, 377-87) — ”Реляционная модель данных для больших совместно используемых банков данных", в июне 1970 г. В основу своей

новой модели он положил два раздела математики: теорию множеств и исчисление предикатов первого порядка. В самом деле, само наименование новой модели происходит от термина "отношение" (relation), заимствованного из теории множеств. (Широко распространено заблуждение, что свое наименование реляционная модель якобы получила из-за того, что таблицы в рамках реляционной базы данных могут быть связаны (related) одна с другой. Теперь, когда вам известна правда, вы будете мирно и спокойно спать по ночам!) К счастью, вам не требуется что-либо знать о теории множеств и логике предикатов первого порядка для проектирования и использования реляционной базы данных. Если используется хорошая методика проектирования базы данных — как представленная в книге Майкла Хернандеса *Проектирование баз данных для простых смертных*,— то можно разработать надежную и эффективную структуру БД для сбора и сохранения любых данных. (Но для того, чтобы решать более сложные задачи, требуются базовые познания в теории множеств. Мы приведем необходимый объем информации в главе 7.)

Программное обеспечение для реляционных баз данных

С момента своего появления реляционная модель является основой программных продуктов для баз данных, известных как системы управления реляционными базами данных (СУРБД — Relational Database Management System, RDBMS). Выпускаемые различными поставщиками, они спустя несколько лет получили признание в различных отраслях и организациях и используются в средах различного типа. В 70-х годах мэйнфреймы (большие универсальные ЭВМ) использовали такие программы, как *System R* компании IBM и *INGRES*, разработанную в Калифорнийском университете в Беркли. В 80-е годы СУРБД для мэйнфреймов получили развитие в таких программных продуктах, как *Oracle* одноименной компании и *DB2* компании IBM. Бурное развитие персональных компьютеров в середине 80-х вызвало появление таких программ, как *dBase* от Ashton Tate, *Paradox* от Ansa Software и *R:BASE* от Microrim. Когда в начале 90-х годов стала очевидной потребность в совместном использовании данных персональными компьютерами, родилась клиент-серверная вычислительная модель вместе с идеей централизованно расположенных общих данных, которыми было бы легко управлять и обеспечивать их безопасность. Эта концепция дала начало таким продуктам, как *Oracle 8i* и Microsoft *SQL Server 7*. Приблизительно с 1996 г. были приложены более согласованные усилия к обеспечению доступности баз данных в Интернете. Поставщики программного обеспечения серьезно отнеслись к этим усилиям и теперь предлагают продукты "с расширенной поддержкой Web", такие как *Cold Fusion* от Allaire, *Sybase Enterprise Application Studio* от Sybase и *Visual InterDev* от Microsoft.

Анатомия реляционных баз данных

В соответствии с реляционной моделью данные в реляционной базе данных сохраняются в отношениях, или *связях*, которые воспринимаются пользователем как таблицы. Каждое отношение состоит из *кортежей* (записей) и *атрибутов* (полей):

Customers

CustomerID	FirstName	LastName	StreetAddress	City	State	ZipCode
1010	Michael	Davolio	672 Lamont Ave	Houston	TX	77201
1011	Margaret	Peacock	667 Red River Road	Austin	TX	78710
1012	Estella	Pundt	2500 Rosales Lane	Dallas	TX	75260
1013	Mark	Rosales	323 Advocate Lane	El Paso	TX	79915
1014	Consuelo	Maynez	3445 Cheyenne Road	El Paso	TX	79915
1015	Ryan	Ehrlich	455 West Palm Ave	San Antonio	TX	78284

Записи

Поля

Рис. 1.1. *Пример таблицы*

("Кортеж" — это термин теории множеств, обозначающий упорядоченную последовательность нескольких величин — *Прим. пер.*) Реляционная база данных имеет некоторые другие характеристики, которые обсуждаются ниже.

Пример таблицы представлен на рис. 1.1.

Таблицы

Таблицы являются основными структурами в базах данных. Каждая таблица описывает отдельный предмет. Логический порядок записей и полей в таблице совершенно не имеет значения. Каждая таблица содержит хотя бы одно поле (называемое *первичным ключом*), которое однозначно идентифицирует каждую из записей. Например, на рис. 1.1 CustomerID является первичным ключом таблицы Customers. Фактически, вследствие последних двух характеристик таблицы данные в реляционной базе данных могут существовать независимо от способа их физического хранения в компьютере. Это очень удобно для пользователей, поскольку от них не требуется знания физического расположения записей для того, чтобы извлекать данные.

Предмет, который представляет данная таблица, может быть либо *объектом*, либо *событием*. Когда предмет является объектом, таблица описывает что-то реальное, например человека, место или вещь. Независимо от своего типа, каждый объект имеет характеристики, которые могут сохраняться в качестве данных. Эти данные могут затем обрабатываться почти бесконечным числом способов. Летчики, изделия, машины, студенты, здания и оборудование — это все примеры объектов, которые можно представить в виде таблицы. На рис. 1.1 показан один из наиболее общих примеров таблиц такого типа.

Когда предметом таблицы является событие, таблица представляет что-то, что происходит в заданный момент времени и обладает характеристиками, которые требуется записать. Эти характеристики можно сохранить как данные, а затем обрабатывать как информацию таким же образом, как и таблицу, которая представляет некоторый конкретный объект. К примерам событий, которые можно записать, относятся слушания дел в суде, распределение денежных средств, результаты

Patient Visit

Patient ID	Visit Date	Visit Time	Physician	Blood Pressure	Temperature
92001	1998-05-01	10:30	Hernandez	120 / 80	98.8
97002	1998-05-01	13:00	Piercy	112 / 74	97.5
99014	1998-05-02	09:30	Rolson	120 / 80	98.8
96105	1998-05-02	11:00	Hernandez	160 / 90	99.1
96203	1998-05-02	14:00	Hernandez	110 / 75	99.3
98003	1998-05-02	09:30	Rolson	120 / 80	98.8

Рис. 1.2. *Таблица, представляющая "событие"*

лабораторных исследований и геологических съемок. На рис. 1.2 приведен пример таблицы, представляющей событие, с которым все мы сталкиваемся несколько раз — прием у врача.

Поля

Поле является атомарной структурой в базе данных, и оно представляет собой характеристику предмета таблицы, к которой оно относится. Поля реально используются для хранения данных. Данные из этих полей можно затем извлекать и представлять как информацию почти в любой вообразимой форме. Запомните, что качество информации, получаемой из таблицы, прямо пропорционально времени, посвященному обеспечению структурной целостности и целостности данных в самих полях. Совершенно невозможно недооценить важность полей.

Каждое поле в базе данных, спроектированной надлежащим образом, содержит одно и только одно значение, а имя поля идентифицирует тип хранимого в нем значения. Это само по себе делает ввод данных в поле интуитивным. Если поля имеют такие имена, как FirstName, LastName, City, State и ZipCode, то не нужно гадать, значения какого типа заносятся в каждое поле. Также облегчается процесс сортировки данных по состоянию или поиск кого-либо с фамилией Hernandez.

Записи

Запись представляет собой уникальный экземпляр предмета таблицы. Она составляется из полного набора полей в таблице независимо от того, содержат ли эти поля какие-либо значения или нет. Вследствие способа определения таблицы каждая запись идентифицируется в базе данных уникальным значением в поле первичного ключа этой записи.

На рис. 1.1, например, каждая запись представляет уникального клиента в рамках таблицы, а поле CustomerID идентифицирует указанного клиента в базе данных. В свою очередь, каждая запись включает все поля таблицы, а каждое поле описывает некоторый аспект клиента, представленного этой записью. Записи являются ключевым фактором в понимании отношений между таблицами, поскольку необходимо знать, как запись в одной таблице связана с записями в другой таблице.

Ключи

Ключи являются специальными полями, которые играют совершенно особую роль в таблице. Тип ключа определяется его предназначением. Хотя таблица может содержать несколько типов ключей, наше обсуждение ограничится двумя наиболее важными: *первичным ключом* и *внешним ключом*.

Первичный ключ — это поле (или группа полей), которое уникальным образом идентифицирует каждую запись в таблице. Когда первичный ключ состоит из двух или более полей, он называется составным первичным ключом. Первичный ключ является наиболее важным из всех по двум причинам: его *значение* идентифицирует *конкретную запись* из всей базы данных, а его *поле* идентифицирует *указанную таблицу* во всей базе данных. Первичные ключи также обеспечивают целостность на уровне таблиц и помогают установить связи с другими таблицами. Каждая таблица в базе данных должна иметь первичный ключ.

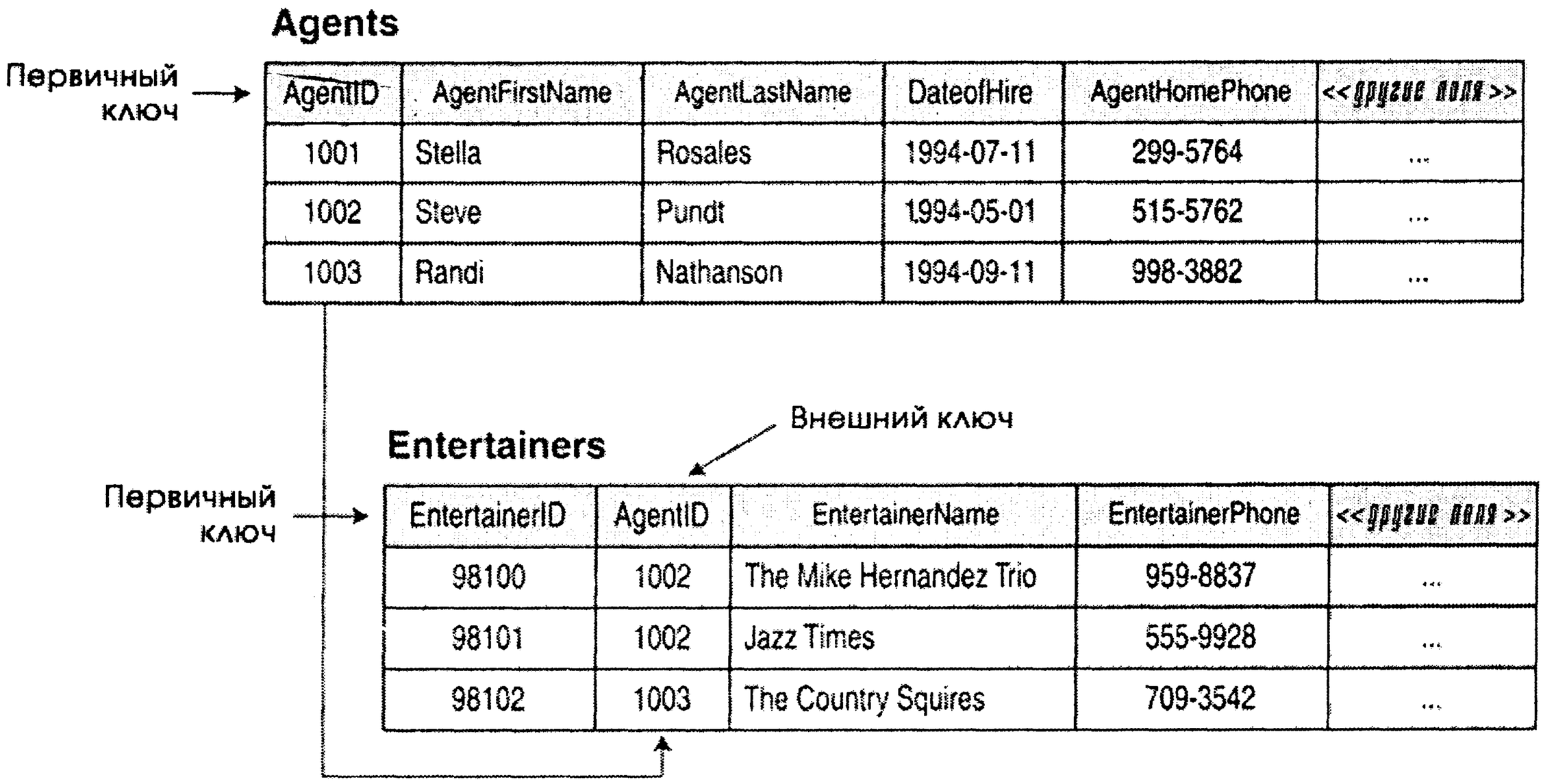


Рис. 1.3. *Первичный и внешний ключи*

Поле AgentID на рис. 1.3 является хорошим примером первичного ключа, потому что оно уникальным образом идентифицирует каждого агента в таблице Agents и помогает обеспечить целостность на уровне таблицы, гарантируя отсутствие дублирующих записей. Оно также используется для установления связей между таблицей Agents и другими таблицами в базе данных, например таблицей Entertainers, показанной в примере.

При установлении связи двух таблиц друг с другом обычно берут копию первичного ключа из первой таблицы и вставляют ее во вторую таблицу, где она становится внешним ключом. (Поскольку во второй таблице уже имеется собственный первичный ключ, то первичный ключ, вносимый из первой таблицы, будет “внешним” для второй таблицы.)

На рис. 1.3 приведен пример внешнего ключа, в котором AgentID является первичным ключом таблицы Agents и внешним ключом в таблице Entertainers. Как можно видеть, в таблице Entertainers уже имеется первичный ключ — EntertainerID. В этом взаимоотношении AgentID является полем, устанавливающим связь между Agents и Entertainers.

Внешний ключ важен не только потому, что он помогает установить связь между двумя таблицами, но он также обеспечивает целостность на уровне связей (т. н. ссылочную целостность) базы данных. Это означает, что записи в обеих таблицах всегда будут правильно связаны, потому что значения внешнего ключа *должны* быть скопированы из значений первичного ключа, на который он ссылается. Внешние ключи также помогают избежать опасных "висящих в воздухе записей", классическим примером которых является запись заказа, не связанная с клиентом. Если не известно, кто сделал заказ, то его невозможно обработать и, что очевидно, выставить счет. Это вызовет спад квартальных продаж!

Представления

Представление является виртуальной таблицей, которая составлена из полей одной или нескольких таблиц базы данных. Таблицы, из которых составляют представление, называются *базовыми*. В реляционной модели представление считается виртуальной таблицей, поскольку оно извлекает данные из базовых таблиц, а не хранит все данные в себе. На самом деле единственной информацией о представлении, которая сохраняется в базе данных, является ее структура.

Представления позволяют рассматривать информацию в базе данных с различных точек зрения, обеспечивая тем самым большую гибкость в работе с данными. Представление можно создать разными способами. Они особенно полезны, когда основываются на нескольких связанных таблицах. Например, можно создать представление, которое суммирует такую информацию, как общее количество часов, отработанное каждым столяром в пределах деловой части Эль-Пасо. Или можно создать представление, которая группирует данные по конкретным полям. Представление такого типа отображает общее количество служащих в каждом городе в пределах каждого штата указанного ряда регионов.

Пример типичного представления показан на рис. 1.4.

Во многих СУРБД представление обычно реализовано в виде *хранимого запроса* или, более кратко, *запроса*. В большинстве случаев запрос имеет все характеристики представления, а его единственное отличие состоит в том, что на него ссылаются по другому имени. Важно отметить, что некоторые поставщики начинают называть запрос его настоящим именем. Независимо от того, как он называется в ваших СУРБД, вы, несомненно, будете использовать представления в своей базе данных.

Customers

CustomerID	CustFirstName	CustLastName	CustPhone	<<другие поля>>
10001	Sally	Callahan	555-2671	...
10002	Ann	Fuller	555-2496	...
10003	James	Leverling	555-2501	...
<<-More Rows Here ----------->>				

Engagements

EngagementNumber	CustomerID	StartDate	End Date	StartTime	<<другие поля>>
1	52113	1999-07-01	1199-07-04	13:00	...
2	54223	1999-07-01	1999-07-05	13:00	...
3	52233	1999-07-10	1999-07-15	13:00	...
<<-More Rows Here ----------->>					

Customer_Engagements *(Представление)*

EngagementNumber	CustFirstName	CustLastName	StartDate	EndDate
1	Mark	Rosales	1999-07-01	1999-07-04
2	Thomas	Fuller	1999-07-01	1999-07-05
3	Sally	Callahan	1999-07-10	1999-07-15
<<-More Rows Here ----------->>				

Рис. 1.4. *Пример представления*

Связи

Если записи указанной таблицы могут быть связаны некоторым образом с записями в другой таблице, то говорят, что между таблицами имеется *связь*. Способ установления этой связи зависит от типа связи. Между двумя таблицами может существовать три типа связей: один-к-одному, один-ко-многим, многие-ко-многим. Уяснение связей является решающим для понимания работы производных таблиц, а также проектирования и использования многотабличных запросов SQL (подробнее см. в части III).

Связь "один-к-одному"

Пара таблиц связана отношением один-к-одному, если одна запись в первой таблице связана *только с одной* записью во второй таблице, а одна запись во второй таблице связана *только с одной* записью в первой таблице. В таком типе связи одна таблица называется *первичной*, а другая — *вторичной*. При связывании берется первичный ключ из первичной таблицы и вставляется во вторичную таблицу,

Agents

AgentID	AgentFirstName	AgentLastName	DateofHire	AgentHomePhone	<<другие поля>>
1001	Stella	Rosales	1994-07-11	299-5764	...
1002	Steve	Pundt	1994-05-01	515-5762	...
1003	Randi	Nathanson	1994-09-11	998-3882	...

Compensation

AgentID	HourlyRate	CommissionRate	<<другие поля>>
1001	21.50	4.5 %	...
1002	25.75	3.0 %	...
1003	20.00	4.5 %	...

Рис. 1.5. *Пример связи один-к-одному*

где он становится внешним ключом. Это особый тип связи, поскольку во многих случаях внешний ключ также является первичным ключом вторичной таблицы.

Пример типичной связи один-к-одному представлен на рис. 1.5, где Agents (Агенты) является первичной таблицей, а Compensation (Заработная плата) — вторичной таблицей. Связь между этими таблицами такова, что отдельная запись в таблице Agents может быть связана только с одной записью в таблице Compensation, а отдельная запись в таблице Compensation — только с одной записью в таблице Agents. AgentID является не только первичным ключом в обеих таблицах, но и внешним ключом во вторичной таблице.

Выбор таблицы, которая будет играть роль первичной в этом типе связи, совершенно произвольный. Связь один-к-одному не является очень распространенной и обычно встречается в случаях, когда таблица разбивается на две части в целях обеспечения конфиденциальности.

Связь "один-ко-многим"

Когда пара таблиц связана между собой по типу один-ко-многим, то одна запись в первой таблице может быть связана с *несколькими* записями во второй таблице, но одна запись во второй таблице может быть связана *только с одной* записью в первой таблице. При установлении такой связи берется первичный ключ в таблице со стороны "один" и вставляется в таблицу со стороны "многие", в которой он становится внешним ключом.

Типичная связь один-ко-многим показана на рис. 1.6. В этом примере одна запись в таблице Artists может быть связана с *несколькими* записями в таблице Engagements, но одна запись в таблице Engagements может быть связана *только с одной* записью в таблице Artists. Как можно уже предположить, ArtistID является внешним ключом в таблице Engagements.

Artists

ArtistID	ArtistFirstName	ArtistLastName	GroupName	<<другие поля>>
67001	Mike	Hernandez	The Mike Hernandez Trio	...
67002	Zachary	Ehrlich	Jazz Times	...
67003	Gerry	Greer	The Country Squires	...

Engagements

EngagementID	ArtistID	CustomerID	StartDate	EndDate	<<другие поля>>
1001	67003	701	1998-09-10	1998-09-12	...
1002	67001	625	1998-09-11	1998-09-12	...
1003	67001	712	1998-09-15	1998-09-19	...

Рис. 1.6. *Пример связи один-ко-многим*

Связь "многие-ко-многим"

Две таблицы связаны отношением многие-ко-многим, когда одна запись в первой таблице может быть связана с *несколькими* записями во второй таблице, а одна запись во второй таблице может быть связана с *несколькими* записями в первой таблице. Чтобы установить эту связь, необходимо создать так называемую *связывающую таблицу*, которая обеспечит легкий способ сопоставления записей из одной таблицы с записями другой и поможет гарантировать отсутствие проблем при добавлении, удалении или модификации любых связанных данных. При определении связывающей таблицы копия первичного ключа каждой таблицы используется для образования структуры новой таблицы. Эти поля в действительности выполняют две различные роли: вместе они образуют составной первичный ключ связывающей таблицы, а по отдельности каждое из них является внешним ключом.

Связь многие-ко-многим, которая не была установлена надлежащим образом, обычно называется неразрешенной. На рис. 1.7 представлен понятный пример неразрешенной связи многие-ко-многим. В этом случае одна запись в таблице Artists может быть связана с несколькими записями в таблице Recordings, *а* отдельная запись в таблице Recordings может быть связана с *несколькими* записями в таблице Artists.

Эта связь является неразрешенной вследствие унаследованной проблемы связи многие-ко-многим. Вопрос состоит в том, как проще связать записи из первой таблицы с записями во второй таблице. Перефразируем вопрос в терминах таблиц, показанных на рис. 1.7. Как соотнести некоторого артиста с несколькими звукозаписями или конкретную звукозапись с несколькими артистами? Вставляется ли несколько полей из Artists в таблицу Recordings? Или добавляется ли несколько полей из Recordings в таблицу Artists? Любой из этих вариантов создаст множество проблем при попытке работать со связанными данными, в частности относительно

Artists

ArtistID	ArtistFirstName	ArtistLastName	GroupName	<<другие поля>>
67001	Mike	Hernandez	The Mike Hernandez Trio	...
67002	Zachary	Ehrlich	Jazz Times	...
67003	Gerry	Greer	The Country Squires	...

Recordings

RecordingID	Title	YearReleased	<<другие поля>>
1102	Jazz 'Round Midnight	1995	...
1103	Until I Return	1998	...
1104	Love Me, Don't Leave Me	1995	...
1105	Midnight Breeze	1996	...
1106	No Puede Ver	1994	...

Рис. 1.7. *Неразрешенная связь многие-ко-многим*

целостности данных. Решение этой дилеммы заключается в создании связывающей таблицы ранее описанным способом. Посредством создания и использования связывающей таблицы можно надлежащим образом разрешить связь многие-ко-многим. На рис 1.8 это решение представлено на практике.

Artists

ArtistID	ArtistFirstName	ArtistLastName	GroupName	<<другие поля>>
67001	Mike	Hernandez	The Mike Hernandez Trio	...
67002	Zachary	Ehrlich	Jazz Times	...
67003	Gerry	Greer	The Country Squires	...

Artist_Recordings *(таблица связей)*

ArtistID	RecordingID
67001	1102
67001	1105
67002	1102
67002	1106
67003	1104

Recordings

RecordingID	Title	YearReleased	<<другие поля>>
1102	Jazz 'Round Midnight	1995	...
1103	Until I Return	1998	...
1104	Love Me, Don't Leave Me	1995	...
1105	Midnight Breeze	1996	...
1106	No Puede Ver	1994	...

Рис. 1.8. *Связь многие-ко-многим, разрешенная надлежащим образом*

При создании связывающей таблицы на рис 1.8 был взят столбец ArtistID из таблицы Artists и RecordingID из таблицы Recordings, и они использовались как основа для новой таблицы. Как и для любой другой таблицы в базе данных, новой связывающей таблице присвоено собственное имя — Artist_Recordings. Реальное преимущество связывающей таблицы состоит в том, что она позволяет связывать любое количество записей из обеих связанных таблиц. Как видно из примера, теперь можно легко связать указанного артиста с любым количеством звукозаписей или определенную звукозапись с любым количеством артистов.

Понимание связей приносит большую пользу при работе с запросами SQL к нескольким таблицам, поэтому непременно возвратитесь к этому разделу, когда начнете изучать часть III книги.

Зачем все это нужно

Почему следует добиться понимания реляционных баз данных? Почему нужно к тому же заботиться о том, какой тип среды используется для работы с данными? И, кроме того, что действительно это вам дает? Именно здесь начинается образование — и удовольствие.

Время, проведенное за изучением реляционных баз данных, можно рассматривать как хорошее капиталовложение вашего ценнейшего ресурса — времени, и в ваших собственных интересах сделать это. Следует получить хорошие рабочие знания о реляционной базе данных, поскольку это наиболее широко используемая модель данных из существующих сегодня. Позабудьте прочитанное в рекламных публикациях и болтовню сотрудников — сбор, сохранение и манипулирование большинством данных, используемых в коммерческой деятельности и в организациях, осуществляется в реляционных базах данных. Да, были предложены и расширения к модели, и прикладные программы, работающие с реляционными базами данных, введен объектно-ориентированный подход, и реляционные базы данных интегрированы в некоторой степени в Web. Но совершенно все равно, как ни нарежешь — то ли ломтиками, то ли кубиками — и какими пряностями ни приправишь, это все еще реляционная база данных! Реляционным базам данных уже более 25 лет, они все еще надежны, и их замена в ближайшем обозримом будущем пока еще не предполагается.

Почти все коммерческое прикладное программное обеспечение по управлению базами данных, которое используется в настоящее время, является реляционным. (Однако такие люди, как д-р Кодд, С. Дж. Дэйт и Фабиан Паскаль могут серьезно сомневаться в таких заявлениях!) Если вы хотите получить хорошо оплачиваемую должность в области, связанной с базами данных, то следует лучше знать, как спроектировать реляционную базу данных и как реализовать ее, используя одну из популярных СУРБД. При текущем форсировании многими компаниями и корпорациями Интернет-коммерции лучше иметь некоторый опыт разработки в Web. Наличие хорошего практического знания реляционных баз данных полезно во многих случаях. По существу, чем больше вы знаете о проектировании реляционных баз данных, тем

легче будет разрабатывать приложения для конечного пользователя базы данных. Вы будете удивлены тем, насколько интуитивно понятными станут ваши БД, потому что вы будете понимать, почему предоставляются какие-либо инструментальные средства и как ими пользоваться с наибольшей выгодой. Ваши практические знания будут серьезным активом, поскольку вы изучите, как использовать SQL, который является стандартным языком для создания, поддержки и работы с реляционной базой данных.

Куда идти дальше

Зная о важности изучения реляционных баз данных, следует осознать, что существует разница между *теорией баз данных* и *их проектированием*. Теория включает в себя принципы и правила, которые определяют основу реляционной модели БД. Именно это изучается в священных залах академий, а затем быстро выбрасывается из головы в темных каморках "реального мира". Но теория, тем не менее, важна: она гарантирует, что реляционная база данных имеет надежную структуру и что все действия, предпринятые над данными в базе данных, дают предсказуемые результаты. С другой стороны, проектирование базы данных включает в себя структурированный, организованный набор процессов, которые используются для этого проектирования. Хорошая методика проектирования БД поможет обеспечить целостность, согласованность и точность данных и гарантировать, что любая извлеченная информация будет, насколько это возможно, точной и неустаревшей.

Если нужно спроектировать и создать базы данных в масштабе предприятия или разработать базы данных для Web-коммерции, или начать тщательные исследования хранилищ данных, то следует серьезно задуматься об изучении теории баз данных. Это пригодится и примется во внимание, когда вы станете консультантом по базам данных высокого уровня. Для остальных, тех, кто собирается проектировать и создавать реляционные базы данных на различных платформах (они, как мы надеемся, составляют преобладающее большинство читателей этой книги), изучение хорошей методики проектирования надежных баз данных окажется полезным. Всегда помните, что проектировать базу данных относительно легко, но *реализовать* ее для определенной СУРБД на конкретной платформе — это совсем другой вопрос.

На рынке имеется множество хороших книг о проектировании баз данных. Некоторые из них, например книга одного из авторов — Mike Hernandez *Database Design for Mere Mortals* — рассматривает только методы проектирования баз данных. Другие, как, например, C. J. Date *An Introduction to Database Systems*, соединяют как теорию, так и проектирование (предупреждаем, что книги по теории совсем не являются легким чтением). Как только принято решение, в каком направлении вы хотите двигаться, купите соответствующие книги, запаситесь напитком по своему выбору и — приступайте к раскопкам. Когда-нибудь, когда вы освоитесь с реляционными базами данных в целом, вы обнаружите, что необходимо как следует изучить SQL.

А это — предмет и задача данной книги.

Итоги

В начале данной главы кратко обсуждались различные типы баз данных, используемых в настоящее время. Организации, работающие с динамическими данными, используют операционные базы данных, гарантирующие, что извлеченная информация всегда точна и соответствует текущему моменту, насколько это возможно. Организации, работающие со статическими данными, используют аналитические базы данных.

Затем мы рассмотрели краткую историю модели реляционной базы данных. Д-р Э. Ф. Кодд создал реляционную модель, исходя из достижений некоторых разделов математики, и модель существует уже свыше 25 лет. Программное обеспечение баз данных было разработано для различных компьютерных сред и постоянно росло по мощности, производительности и возможностям с 70-х годов. От больших универсальных ЭВМ до настольных ПК и сред Web — СУРБД являются сегодня стержнем многих организаций.

Кроме того, мы описали внутреннее строение реляционной базы данных. Мы познакомили вас с ее базовыми концепциями и кратко пояснили их назначение. Вы изучили три типа связей и теперь понимаете их важность не только в терминах самой структуры базы данных, но и в свете вашего понимания SQL.

Полезно изучить реляционные базы данных и знать, как они проектируются, поскольку они являются наиболее распространенным типом баз данных, используемых в настоящее время, и почти каждая программа ПО для баз данных должна будет использоваться для поддержки реляционной базы данных.

2

Обеспечение надежности структуры базы данных

"Мы определяем облик зданий, которые мы строим: впоследствии они определяют наш облик".

— Сэр Уинстон Черчилль

Вопросы, рассматриваемые в данной главе:

- Почему эта глава помещена здесь
- Зачем нужна хорошо продуманная структура
- Настройка полей
- Настройка таблиц
- Установка и исправление связей
- Все ли это?
- Итоги

Большинство читателей этой книги, вероятно, работают с существующей структурой базы данных, реализованной в вашей излюбленной (мы надеемся) программной СУРБД. В данный момент сложно предположить, имелись ли у вас (или у лица, разработавшего базу данных) необходимые знания и навыки или время для надлежащего проектирования БД. Предположим худшее. Вероятно, имеется множество таблиц, которые могут использовать тонкую настройку. Некоторые методики помогут вам привести свою базу данных в порядок и обеспечат возможность легкого извлечения информации из ваших таблиц.

Почему эта глава помещена здесь

Не стоит удивляться тому, что мы обсуждаем вопросы проектирования БД в этой книге и что они включены в вводные главы. Причина проста. Если база данных имеет плохо проработанную структуру, то многие из операторов SQL (они включены в остальную часть книги) будет в лучшем случае трудно реализовать, а в худшем случае они будут относительно бесполезны. Однако, если структура базы данных хорошо спроектирована, навыки, приобретенные при чтении этой книги, окажутся весьма нужными.

Данная глава поможет привести базу данных в относительный порядок. Мы настоятельно рекомендуем прочитать ее, чтобы быть в состоянии проверить надежность структур своих таблиц.

> **Внимание!** Мы будем рассматривать *логическую* структуру базы данных. Мы не предполагаем объяснять, как создавать или реализовать базу данных на SQL, потому что эти вопросы не входят в задачу данной книги.

Зачем нужна хорошо продуманная структура

Если структура вашей базы данных плохо проработана, вы столкнетесь с проблемами при извлечении из своей базы данных простой на вид информации, с вашей базой данных будет трудно работать, и вы будете вздрагивать от страха каждый раз при необходимости добавить или удалить поля в своих таблицах. И другие аспекты базы данных, например целостность данных, связи таблиц и возможность извлекать точную информацию нарушаются, если структуры спроектированы плохо. Эти вопросы являются только верхушкой айсберга. И их еще множество! Убедитесь в хорошей проработке структур для того, чтобы избежать всех этих неприятностей.

Многих из этих проблем можно избежать при правильном проектировании БД с самого начала. Даже если база данных уже спроектирована, не все еще потеряно. Все еще можно применить предложенные ниже методы и создать хорошую структуру. Однако необходимо принять во внимание, что качество конечной структуры напрямую зависит от количества времени, *затраченного* на тонкую настройку. Чем больше заботы и терпения будет уделено при применении методики, тем с большей вероятностью можно гарантировать успех.

Теперь вернемся к первоочередным задачам формирования ваших структур: займемся полями.

Настройка полей

Поскольку поля являются самыми основными структурами в базе данных, то необходимо, чтобы они находились в самом лучшем виде, прежде чем заняться тонкой настройкой таблиц в целом. Во многих случаях исправление структуры полей исключит множество проблем, связанных с данной таблицей, и поможет избежать потенциальных проблем, которые могли бы возникнуть.

Что в имени тебе моем? (Часть первая)

Как известно, поле представляет характеристику предмета таблицы, к которой оно принадлежит. Если присвоить полю соответствующее наименование, то можно будет идентифицировать характеристику, которую предположительно оно представляет. Неоднозначное, неосмысленное или непонятное имя является верным признаком проблем и предполагает, что назначение этого поля не было тщательно

продумано. Воспользуйтесь следующим контрольным списком для тестирования каждого имени поля.

- *Является ли имя описательным и осмысленным для всей вашей организации?* Если сотрудники нескольких отделов предполагают использовать эту базу данных, убедитесь в том, что используемое имя имеет ясный смысл для каждого из них. Семантика — забавная вещь, и если это слово будет иметь для разных людей разный смысл, то вы просто нарветесь на проблемы.

- *Является ли имя поля однозначным?* Поле с именем PhoneNumber (Телефонный номер) может очень легко ввести в заблуждение. Что это за номер телефона? Домашний? Рабочий? Мобильный? Давайте привыкать к точности. Если требуется записать каждый из этих типов номеров телефонов, создайте поля HomePhone, WorkPhone и CellPhone.

  Для того чтобы сделать имена полей более четкими и однозначными, позаботьтесь о том, чтобы не использовать одни и те же имена в нескольких таблицах. Предположим, что имеются три таблицы с именами Customers, Vendors и Employees. Нет сомнений, что в каждой из этих таблиц будут поля City и State и что эти поля будут одинаково называться во всех трех таблицах. Это не будет представлять собой проблему до тех пор, пока вы не обратитесь к одному конкретному полю. Но как можно будет различить, скажем, поле City в таблице Vendors, поле City в таблице Customers и поле City в таблице Employees? Ответ прост: добавьте короткий префикс к каждому имени поля. Например, используйте имя VendCity в таблице Vendors, CustCity в таблице Customers и EmpCity в таблице Employees. Теперь можно легко сделать понятную ссылку на каждое из этих полей. (Этот метод можно использовать для любого группового поля, например FirstName, LastName и Address.)

  Главное, чтобы каждое поле в вашей базе данных имело уникальное имя и чтобы оно появлялось только один раз во всей структуре базы данных. Единственным исключением из этого правила является случай, когда поле используется для установки связи между двумя таблицами.

- *Используется ли в качестве имени поля акроним или сокращение?* Если да, то замените его! Сокращение бывает трудно расшифровать и можно неверно понять. Представьте себе поле с именем CAD_SW. Извольте догадаться, что это значит! Используйте сокращения как можно реже и обрабатывайте их внимательно. Применяйте их только в том случае, если они дополняют или расширяют имя поля положительным образом. Сокращение не должно отвлекать от смысла имени поля.

- *Не используется ли имя, которое явно или неявно обозначает более одной характеристики?* Этот тип имен легко узнать, поскольку они обычно включают слова “и” или “или”. Имена полей, которые содержат обратную наклонную черту (\), дефис (-), или амперсанд (&), могут легко обмануть. Если у вас имеются поля с такими именами, как телефон/факс (Phone\Fax), область (Area) или местонахождение (Location), то пересмотрите данные, которые они содержат, и определите, требуется ли разделить их на более мелкие, отдельные поля.

Внимание! Стандарт SQL определяет обычный (регулярный) идентификатор как имя, которое должно начинаться с буквы и может содержать только буквы, цифры и символ подчеркивания; пробелы не допускаются. Он также определяет идентификатор с разделителями как имя, заключенное в двойные кавычки, которое должно начинаться с буквы и может содержать буквы, цифры, символы подчеркивания, пробелы и совершенно специфический набор специальных символов. Поскольку многие версии SQL поддерживают только регулярные идентификаторы, рекомендуется использовать для имен своих полей именно это правило.

После использования этой контрольной таблицы для изменения имен своих полей остается еще одна задача: Убедитесь, что имена полей используются в единственном числе. Поле с именем во множественном числе, например Categories (Категории), подразумевает, что в нем может содержаться два или больше значений. Имя поля используется в единственном числе, потому что оно представляет собой отдельную характеристику предмета, описываемого таблицей, к которой оно принадлежит. С другой стороны, имя таблицы имеет множественное число, поскольку она представляет собой набор подобных объектов или событий. Можно достаточно легко отличить имя таблицы от имени поля, если использовать это соглашение об именах.

Сглаживание грубых краев

После приведения в порядок имен полей сосредоточимся на структуре самих полей. Хотя вы достаточно уверены в надежности своих полей, однако существует несколько моментов, которые можно проверить, чтобы убедиться в том, что поля построены наиболее эффективным образом. Протестируйте свои поля в соответствии со следующим контрольным списком и определите, нужно ли еще поработать с ними.

- *Убедитесь, что поле представляет конкретную характеристику предмета таблицы.* Основное намерение состоит здесь в определении, действительно ли поле относится к данной таблице. Если оно не соответствует таблице, удалите его. Единственное исключение из этого правила возникает при использовании поля для установления связи между этой и другой таблицей в базе данных или при его добавлении к таблице для поддержки некоторой задачи, необходимой приложению.

Staff

StaffID	StaffFirstName	StaffLastName	StaffStreetAddress	StaffCity	StaffState	<<другие поля>>
98014	James	Leverling	722 Moss Bay Blvd.	Kirkland	WA	...
98019	Laura	Callahan	901 Pine Avenue	Portland	OR	...
98020	Albert	Buchanan	13920 S.E. 40th Street	Bellevue	WA	...
98021	Tim	Smith	30301- 166th Ave. N.E.	Seattle	WA	...
98022	Janet	Leverling	722 Moss Bay Blvd.	Kirkland	WA	...
98023	Alaina	Hallmark	Route 2, Box 203 B	Woodinville	WA	...

Classes

ClassID	Class	ClassroomID	StaffID	StaffLastName	StaffFirstName	<<другие поля>>
1031	Art History	1231	98014	Leverling	James	...
1030	Art History	1231	98014	Leverling	James	...
2213	Biological Principles	1532	98021	Smith	Tim	...
2005	Chemistry	1515	98019	Callahan	Laura	...
2001	Chemistry	1519	98233	Hallmark	Alaina	...
1006	Drawing	1627	98020	Buchanan	Albert	...
2907	Elementary Algebra	3445	98022	Leverling	Janet	...

Рис. 2.1. *Таблица с ненужными полями*

Например, в таблице Classes на рис. 2.1 поля StaffLastName и StaffFirstName являются лишними вследствие наличия поля StaffID. StaffID используется для установления связи между таблицей Classes и таблицей Staff (Преподаватели), и можно видеть данные из обеих таблиц одновременно, воспользовавшись производной таблицей или запросом Select в SQL. Если в таблицах имеются такие поля, то можно либо совсем их удалить, либо использовать как основу для новой таблицы, если они не появляются больше нигде во всей структуре базы данных.

- *Убедитесь в том, что каждое поле содержит только одно значение.* Поле, потенциально сохраняющее несколько экземпляров *одного и того же* значения, называется *многозначным* полем. Поле, которое может потенциально сохранять два или более *различных* значений, называется *составным* полем. Многозначные и составные поля могут разрушить базу данных, особенно при попытках редактирования, удаления или сортировки данных. Когда гарантируется, что каждое поле сохраняет только одно значение, вы выбираете длинный путь обеспечения гарантий целостности данных и достоверности информации.
  Но в настоящий момент просто попытайтесь идентифицировать все многозначные или составные поля и пометьте их.

- *Убедитесь в том, что в поле не сохраняется результат вычислений или объединения.* В таблице, спроектированной надлежащим образом, вычисляемых полей быть не должно. Дело в самой сути вычисляемого поля. Поле базы данных, в отличие от ячейки в электронной таблице, хранит не вычисление, а его результат. Когда значение любой части вычисления изменяется, результирующее значение, сохраненное в поле, не корректируется. Единственный способ корректировки состоит в ручном исправлении значения или в написании некоторой процедуры, которая будет выполнять это автоматически. В любом случае на пользователя или на разработчика возлагается обязанность убедиться, что значение скорректировано. Однако предпочтительным способом работы с вычислениями является включение их в оператор SELECT (см. главу 5).

- *Убедитесь в том, что поля появляются только один раз во всей базе данных.* Если вы сделаете обычную ошибку, введя одинаковое поле (например, CompanyName) в несколько таблиц в пределах базы данных, вы можете впоследствии столкнуться с проблемой противоречивости данных. Это случается, когда изменяют значение в одной таблице и забывают сделать такое же изменение в других местах появления поля.
  Чтобы такого не произошло, каждое поле должно появляться во всей структуре базы данных только один раз (единственное исключение из этого правила возникает, когда поле используется для установления связи между двумя таблицами).

Разбиение составных полей

Составные и многозначные поля нарушают целостность данных, поэтому необходимо разделить их для того, чтобы избежать любых потенциальных проблем. Решение о том, что рассматривать в первую очередь, является совершенно произвольным, поэтому начнем с составных полей.

Customers

CustomerID	CustomerName	StreetAddress	PhoneNumber	<<другие поля>>
1001	Suzanne Viescas	15127 NE 24th, #383, Redmond, WA 98052	425 555-2686	...
1002	Will Thompson	122 Spring River Drive, Duvall, Wa 98019	425 555-2681	...
1003	Gary Hallmark	Route 2, Box 203B, Auburn, WA 98002	253 555-2676	...
1004	Michael Davolio	672 Lamont Ave, Houston, TX 77201	713 555-2491	...
1005	Kenneth Peacock	4110 Old Redmond Rd., Redmond, WA 98052	425 555-2506	...
1006	John Viescas	15127 NE 24rh, #383, Redmond, WA 98052	425 555-2511	...
1007	Laura Callahan	901 Pine Avenue, Portland, OR 97208	503 555-2526	...
1008	Neil Patterson	233 West Valley Hwy, San Diego, CA 92199	619 555-2541	...

Составные поля

Рис. 2.2. *Таблица с составными полями*

Для того чтобы узнать, имеется ли составное поле, ответьте на совсем простой вопрос: "Можно ли взять текущее значение этого поля и разделить его на несколько более мелких частей?" Если ответом будет "Да", то это составное поле. Рис. 2.2 показывает плохо спроектированную таблицу с несколькими составными полями.

В этой таблице три составных поля: CustomerName, StreetAddress и PhoneNumber. Видно, что каждое поле можно разбить на несколько меньших полей. Например, CustomerName можно разделить на два отдельных поля: CustFirstName и CustLastName (мы использовали соглашение о присвоении имен и добавили префикс Cust к полям FirstName и LastName). Когда в таблице обнаруживается составное поле, определите, из скольких частей состоит значение, которое в нем сохраняется, а затем разделите поле на несколько более мелких полей, в зависимости от обстоятельств. На рис. 2.3 показано, как разделить многозначное поле в таблице Customers.

Customers

CustomerID	CustFirstName	CustLastName	CustAddress	CustCity	CustState	CustZipcode
1001	Suzanne	Viescas	15127 NE 24th, #383	Redmond	WA	98052
1002	Wil	Thompson	122 Spring River Drive	Duvall	WA	98019
1003	Gary	Hallmark	Route 2, Box 203B	Auburn	WA	98002
1004	Michael	Davolio	672 Lamont Ave	Houston	TX	77201
1005	Kenneth	Peacock	4110 Old Redmond Rd.	Redmond	WA	98052
1006	John	Viescas	15127 NE 24th, #383	Redmond	WA	98052
1007	Laura	Callahan	901 Pine Avenue	Portland	OR	97208
1008	Neil	Patterson	233 West Valley Hwy	San Diego	CA	92199

Рис. 2.3. *Дробление составных полей таблицы Customers*

> **Внимание!** Вместе с разбиением StreetAddress хорошо было бы разделить PhoneNumber на два отдельных поля. К сожалению, из-за отсутствия места невозможно продемонстрировать это на рис. 2.3.

Иногда затруднительно распознать составное поле. Взгляните на таблицу Instruments, представленную на рис. 2.4. На первый взгляд кажется, что в ней отсутствуют какие-либо составные поля. Но при более тщательном рассмотрении вы увидите, что InstrumentID на самом деле является составным полем. Значение, сохраненное в этом поле, представляет две отдельные части информации: категорию, к которой принадлежит инструмент, например AMP (усилитель), GUIT (гитара) и MFX (синтезатор), и его идентификационные номера. Эти два значения необходимо разделить и сохранить в своих собственных полях, чтобы гарантировать целостность данных. Представьте себе трудности корректировки этого поля, если категория MFX изменяется на MFU. Потребуется написать программу для грамматического разбора значения в этом поле и для проверки существования MFX, а затем для замены его на MFU, если он имеется в пределах синтаксически проанализированного значения.

Instruments

InstrumentID	Manufacturer	InstrumentDescription	<<другие поля>>
GUIT2201	Fender	Fender Stratocaster	...
MFX3349	Zoom	Player 2100 Multi-Effects	...
AMP1001	Marshall	JCM 2000 Tube Super Lead	...
AMP5590	Crate	VC60 Pro Tube Amp	...
SFX2227	Dunlop	Cry Baby Wah-Wah	...
AMP2766	Fender	Twin Reverb Reissue	...

Рис. 2.4. *Пример неявного составного поля*

Вы, конечно, сможете сделать эту работу, но определенно затратите больше усилий, чем необходимо, и ее совсем не придется выполнять, если база данных спроектирована надлежащим образом. Если имеются такие поля, как приведенное в примере, разбейте его на меньшие поля, чтобы это были надежные, эффективные структуры.

Дробление многозначных полей

Дробление многозначных полей является совсем несложным, но разбиение многозначных полей окажется немного более трудным и потребует некоторой работы. К счастью, вы узнаете многозначное поле, когда увидите его. Почти без исключения данные, сохраненные в поле этого типа, содержат много запятых. Запятые используются для разделения различных значений в пределах самого поля. Пример многозначного поля представлен на рис. 2.5.

В данном примере каждый летчик имеет допуск к пилотированию некоторого количества самолетов. Эти допуски сохраняются в отдельном поле с именем Certifications. Способ сохранения данных в этих полях очень ненадежен, поскольку вы обречены сталкиваться с тем же типом проблем целостности данных, которые характерны для составных полей. При более внимательном рассмотрении данных можно увидеть, что трудно выполнять поиск и сортировку по этому полю в запросе SQL. Прежде чем разделить это поле соответствующим образом, необходимо понять действительные связи между многозначным полем и таблицей, к которой оно изначально относится.

Pilots

PilotID	PilotFirstName	PilotLastName	HireDate	Certifications	<<другие поля>>
25100	John	Leverling	1994-07-11	727, 737, 757, MD80	...
25101	David	Callahan	1994-05-01	737, 747, 757	...
25102	David	Smith	1994-09-11	757. MD80. DC9	...
25103	Kathryn	Patterson	1994-07-11	727, 737, 747, 757	...
25104	Michael	Hernandez	1994-05-01	737, 757, DC10	...
25105	Kendra	Bonnicksen	1994-09-11	757, MD80. DC9	...

Рис. 2.5. *Пример многозначного поля*

Значения в многозначном поле обладают соотношением многие-ко-многим с каждой записью в его родительской таблице. Одно конкретное значение многозначного поля может быть связано с любым количеством записей родительской таблицы, а отдельную запись в родительской таблице можно соотнести с любым количеством значений в многозначном поле. На рис. 2.5, например, специальный самолет в поле Certifications может ассоциироваться с любым количеством пилотов, а отдельный пилот может быть соотнесен с любым номером самолета в поле Certifications. Эта связь многие-ко-многим разделяется так же, как и любая другая связь многие-ко-многим в рамках базы данных — с помощью связывающей таблицы.

Для создания связывающей таблицы используйте многозначное поле и *копию* поля первичного ключа из исходной таблицы как основу для новой таблицы. Присвойте новой связывающей таблице соответствующее имя и определите оба поля как составной первичный ключ (в данном случае это комбинация значений обоих полей, которые будут идентифицировать уникальным образом каждую запись в новой таблице). Теперь можно связать значение каждого из полей в связующей таблице на основе один-к-одному. На рис. 2.6 представлен пример этого процесса с использованием таблицы Pilots, показанной на рис. 2.5.

Pilots

PilotID	PilotFirstName	PilotLastName	HireDate	<<другие поля>>
25100	John	Leverling	1994-07-11	...
25101	David	Callahan	1994-05-01	...
25102	David	Smith	1994-09-11	...
25103	Kathryn	Patterson	1994-07-11	...
25104	Michael	Hernandez	1994-05-01	...
25105	Kendra	Bonnicksen	1994-09-11	...

Pilot Certifications *(таблица связей)*

PilotID	CertificationID
25100	8102
25100	8103
25100	8105
25100	8106
25101	8103
25101	8104
25101	8105

Certifications

CertificationID	TypeofAircraft	<<другие поля>>
8102	Boeing 727	...
8103	Boeing 737	...
8104	Boeing 747	...
8105	Boeing 757	...
8106	McDonnell Douglas MD80	...

Рис. 2.6. *Разделение многозначного поля с использованием связывающей таблицы*

Сопоставьте записи для Джона Леверлинга (летчика с идентификационным номером ID 25100) в старой таблице Pilots и в новой таблице Pilot_Certifications (Допуски пилотов). Основное преимущество новой связывающей таблицы в том, что теперь можно связывать *любое* количество сертификатов с одним летчиком. Задавать определенные типы запросов стало также намного легче. Например, можно определить, какие летчики имеют допуск к полетам на самолете Боинг-747, или извлечь список допусков для конкретного летчика. Можно также отсортировать данные в любом требуемом порядке без каких-либо неблагоприятных последствий.

При следовании процедурам, представленным в этом разделе, ваши поля будут в хорошем состоянии. Теперь вернемся к делам второго порядка и рассмотрим структуры таблиц.

Настройка таблиц

Таблицы служат основой для любых запросов SQL, которые вы создаете. Плохо спроектированные таблицы вызывают проблемы целостности данных и затрудняют работу с ними при создании многозначных запросов SQL. Вследствие этого нужно проверить, что структура ваших таблиц эффективна настолько, насколько это возможно, и позволяет легко извлекать необходимую информацию.

Что в имени тебе моем? (Часть вторая)

В предыдущей части мы показали, насколько важно присвоить полю соответствующее имя и почему следует серьезно обдумывать имена своих полей. То же самое относится и к таблицам. По определению таблица должна представлять собой один предмет. Если она представляет более одного предмета, ее следует разделить на меньшие таблицы. Имя таблицы должно четко идентифицировать представляемый ею предмет. Если имя таблицы допускает различное, туманное или неясное толкование, значит предмет таблицы не был тщательно продуман. Проверьте правильность имен ваших таблиц в соответствии со следующим контрольным списком:

- *Является ли имя уникальным и достаточно описательным, чтобы оно было однозначно интерпретировано во всей вашей организации?* Присвоение таблице уникального имени гарантирует, что каждая таблица в базе данных представляет другой предмет и что каждый человек в организации поймет, что в ней представлено. Определение уникального описательного имени потребует некоторых усилий с вашей стороны, но эти затраты целесообразны с точки зрения длительной перспективы.
- *Описывает ли имя предмет таблицы точно, понятно и однозначно?* Когда имя таблицы туманно и неоднозначно, то можно поспорить, что таблица представляет более одного предмета. Имя “Данные” — это хороший пример туманного имени таблицы. Трудно точно определить, что представляет эта таблица, если нет на руках ее описания. Допустим, таблица появляется в базе данных агентства развлечений.

При тщательном ознакомлении с ней обнаружится, что в ней содержатся данные о встречах с клиентами и данные о продаже билетов на постоянных эстрадных артистов. Эта таблица явно представляет два предмета. В таком случае разделите таблицу на две и присвойте каждой из них соответствующее имя, например Client_Meetings (Встречи с клиентами) и Entertainer_Schedules (График выступлений эстрадных исполнителей).

- *Содержит ли имя слова, которые обозначают физические характеристики?* Избегайте использования таких слов, как “файл”, “запись” и “таблица”, в именах таблиц, поскольку они вносят неразбериху. Таблица, в имени которой имеется слово такого типа, с очень большой вероятностью может представлять более чем один предмет. Рассмотрим имя Employee_Record (Записи о сотрудниках). На первый взгляд, в связи с таким именем не могут возникнуть какие-то проблемы. Но когда вы задумаетесь о том, что должна включать запись о сотруднике, то поймете, что проблемы весьма вероятны. Имя содержит слово, которое потенциально представляет три предмета: служащих, отделы и платежные ведомости. Учитывая это, разделите исходную таблицу (Employee_Record) на три новые таблицы, по одной для каждого из предметов.

- *Используете ли вы акронимы или сокращения в качестве имени таблицы?* Если да — измените имя немедленно! Сокращения редко могут четко описать предмет таблицы, а акронимы обычно трудно расшифровать. Пусть база данных вашей компании включает таблицу с именем SC. Как же узнать, что представляет таблица, если не знать значения, скрывающегося за этими буквами? Суть в том, что вы не сможете легко установить предмет таблицы. Более того, может оказаться, что таблица означает разное для разных отделов компании (вот это действительно ужасно!) Группа людей в отделе кадров думает, что оно обозначает Steering_Committees (Руководящие комитеты); персонал из отдела информационных систем полагает, что это System_Configurations (Конфигурации системы); сотрудники отдела безопасности считают, что оно обозначает Security_Codes (Коды безопасности). Этот пример отчетливо показывает, почему следует избегать использования сокращений и акронимов в именах таблиц.

- *Используете ли вы имя, которое явно или неявно идентифицирует более одного предмета?* Это одна из наиболее обычных ошибок, которую можно сделать, присваивая имена таблицам, и ее относительно легко распознать. Такой тип имени обычно содержит слова and (и) или or (или) и такие символы, как обратная наклонная черта (\), дефис (-) или амперсанд (&). Типичные примеры: Facility\Building (Здания\Строения) и Department or Branch (Отдел или Отделение). Когда таблице

присваивается такое имя, необходимо ясно распознавать, представляет ли она более одного предмета. Если так, разделите ее на несколько меньших таблиц и присвойте новым таблицам соответствующие имена.

Внимание! Еще раз напомним, что стандарт SQL определяет *обычный* (*регулярный*) *идентификатор* как имя, которое должно начинаться с буквы и может содержать только буквы, цифры и символ подчеркивания; *пробелы не допускаются*. Он также определяет *идентификатор с разделителями* как имя, заключенное в двойные кавычки, которое должно начинаться с буквы и может содержать буквы, цифры, символы подчеркивания, пробелы и совершенно специфический набор специальных символов. Поскольку многие версии SQL поддерживают только регулярные идентификаторы, рекомендуется использовать для имен полей именно это правило.

Проверьте еще раз имена всех таблиц и убедитесь, что имена используются во множественном числе. Множественное число необходимо потому, что таблица хранит *набор экземпляров* предмета таблицы. Например, таблица Employees (Сотрудники) хранит данные не по одному, а по многим сотрудникам. Кроме того, использование множественного числа поможет отличить имя таблицы от имени поля.

Усовершенствование структуры

Теперь сосредоточимся на структуре таблиц. Настоятельно необходимо, чтобы таблицы проектировались надлежащим образом для обеспечения эффективного хранения данных и извлечения достоверной информации. Время, затраченное на проверку построения таблицы, принесет свои дивиденды, когда потребуется создавать сложные запросы SQL для нескольких таблиц. Воспользуйтесь приведенным ниже контрольным списком и определите, надежны ли структуры ваших таблиц.

- *Убедитесь, что таблица представляет единственный предмет.*
  Да, мы уже повторили это несколько раз, но в данном случае повторение не будет лишним. Когда есть гарантия, что каждая из таблиц представляет отдельный предмет, это значительно уменьшает риск потенциальных проблем целостности данных. Предмет, представленный таблицей, может быть объектом или событием. Под “объектом” мы понимаем нечто материальное, например сотрудники, поставщики, машины, здания или отделы. “Событие” — это то, что происходит в заданный момент времени и что вы хотите зарегистрировать. Самым лучшим примером события, к которому каждый может иметь отношение, является прием у врача. Хотя его нельзя потрогать, это событие имеет характеристики, которые необходимо записать, например дата приема, время приема, кровяное давление и температура пациента.

- *Убедитесь, что в каждой таблице имеется первичный ключ.* Первичный ключ необходимо назначить каждой таблице по двум причинам. Во-первых, он однозначно идентифицирует каждую запись в таблице, а во-вторых, он используется для связывания таблиц. Если не назначить первичный ключ каждой таблице, в конце концов возникнут проблемы целостности данных и проблемы в некоторых типах запросов SQL к нескольким таблицам. Позже мы дадим несколько советов о том, как определить надлежащий первичный ключ. (Современные СУРБД, в частности Access 2000, просто не дадут определить таблицу без первичного ключа.— *Прим. пер.*)

- *Убедитесь в том, что таблица не содержит каких-либо составных или многозначных полей.* Теоретически эти вопросы следует решить во время уточнения структуры полей. Тем не менее будет нелишним проверить поля в последний раз и убедиться, что все и каждое из них полностью удалены.

- *Убедитесь в том, что в таблице отсутствуют вычисляемые поля.* Даже если вы уверены, что текущие структуры таблиц не содержат вычисляемых полей, стоит просмотреть некоторые из них во время процесса уточнения полей. В этот момент следует еще раз просмотреть структуры таблиц и удалить все вычисляемые поля, которые, возможно, были пропущены.

- *Убедитесь в том, что поля в таблицах не дублируются.* Одним из признаков плохо спроектированной таблицы является включение дублирующих полей из других таблиц. Возможно, добавить дублирующие поля в таблицу вас вынудило желание добавить "справочную" информацию или обозначить таким образом многократные появления значения определенного типа. Такие дублирующие поля вызывают различные сложности при работе с данными и при попытке извлечь информацию из таблицы. Рассмотрим, как справиться с повторяющимися полями.

Устранение дублирующих полей

Возможно, самой трудной частью усовершенствования структуры является устранение дублирующих полей. Ниже приведено несколько примеров, которые демонстрируют, как следует поступать с таблицами, содержащими дублирующие поля.

На рис. 2.7 представлен пример таблицы, содержащей повторяющиеся поля — якобы для "справки".

В данном случае поля StaffLastName и StaffFirstName появляются в таблице Classes для того, чтобы при просмотре таблицы можно было видеть имя преподавателя данного класса. Однако эти поля излишни вследствие связи один-ко-многим, которая существует между таблицами Classes и Staff (один преподаватель может

Staff

StaffID	StaffFirstName	StaffLastName	StaffStreetAddress	StaffCity	StaffState	<<другие поля>>
98014	James	Leverling	722 Moss Bay Blvd.	Kirkland	WA	...
98019	Laura	Callahan	901 Pine Avenue	Portland	OR	...
98020	Albert	Buchanan	13920 S.E. 40th Street	Bellevue	WA	...
98021	Tim	Smith	30301 166th Ave. N.E.	Seattle	WA	...
98022	Janet	Leverling	722 Moss Bay Blvd.	Kirkland	WA	...
98023	Alaina	Hallmark	Route 2. Box 203 B	Woodinville	WA	...

Classes

Эти поля не нужны

ClassID	Class	ClassroomID	StaffID	StaffLastName	StaffFirstName	<<другие поля>>
1031	Art History	1231	98014	Leverling	James	...
1030	Art History	1231	98014	Leverling	James	...
2213	Biological Principles	1532	98021	Smith	Tim	...
2005	Chemistry	1515	98019	Callahan	Laura	...
2001	Chemistry	1519	98233	Hallmark	Alaina	...
1006	Drawing	1627	98020	Buchanan	Albert	...
2907	Elementary Algebra	3445	98022	Leverling	Janet	...

Рис. 2.7. *Таблица с дублирующими полями*

вести любое количество предметов, но каждый конкретный предмет может вести только один преподаватель). Связь между этими таблицами устанавливает поле StaffID, и само наличие этой связи позволяет одновременно просматривать данные из обеих таблиц в запросе SQL. Учитывая это, можно без опаски удалить поля StaffLastName и StaffFirstName из таблицы Classes, не вызвав каких-либо отрицательных воздействий. На рис. 2.8 представлена исправленная структура таблицы Classes.

Сохранение лишних полей в таблице автоматически вызывает значительные проблемы, связанные с противоречивостью данных. Необходимо гарантировать, что значения полей StaffLastName и StaffFirstName в таблице Classes всегда соответствуют их копиям в таблице Staff. Допустим, женщина-преподаватель выходит замуж и меняет свою фамилию на фамилию мужа. Необходимо быть уверенным не только в том, что соответствующие изменения в ее записи сделаны в таблице Staff, но также гарантировать, что изменены все записи с ее именем в таблице Classes. Конечно, это можно сделать (по крайней мере технически), но придется потрудиться больше, чем необходимо. Помимо этого, одна из главных предпосылок использования реляционной БД состоит в том, что все данные встречаются во всей базе данных только один

Staff

StaffID	StaffFirstName	StaffLastName	StaffStreetAddress	StaffCity	StaffState	<<другие поля>>
98014	James	Leverling	722 Moss Bay Blvd.	Kirkland	WA	...
98019	Laura	Callahan	901 Pine Avenue	Portland	OR	...
98020	Albert	Buchanan	13920 S.E. 40th Street	Bellevue	WA	...
98021	Tim	Smith	30301- 166th Ave. N.E.	Seattle	WA	...
98022	Janet	Leverling	722 Moss Bay Blvd.	Kirkland	WA	...
98023	Alaina	Hallmark	Route 2, Box 203 B	Woodinville	WA	...

Classes

ClassID	Class	ClassroomID	StaffID	<<другие поля>>
1031	Art History	1231	98014	...
1030	Art History	1231	98014	...
2213	Biological Principles	1532	98021	...
2005	Chemistry	1515	98019	...
2001	Chemistry	1519	98233	...
1006	Drawing	1627	98020	...
2907	Elementary Algebra	3445	98022	...

Рис. 2.8. *Устранение дублирующих полей*

раз (единственным исключением из этого правила является установка связи между таблицами). Как всегда, лучше всего удалить из таблиц базы данных все дублирующие поля.

Другой пример таблицы, содержащей дубликаты полей, представлен на рис. 2.9. Он показывает, как дубликаты полей ошибочно используются для обозначения того, что значение встречается несколько раз. В этом случае три поля Committee (Комитет) используются якобы для записи названий комитетов, в которых принимает участие сотрудник.

Employees

EmployeeID	EmpFirstName	EmpLastName	Committee1	Committee2	Committee3	<<другие поля>>
7004	Peacock	Samuel	Steering			...
7005	Kennedy	John	Y2K Conformance	Safety		...
7006	Thompson	Sarah	Safety	Y2K Conformance	Steering	...
7007	Callahan	David				...
7008	Buchanan	Andrea	Y2K Conformance			...
7009	Smith	David	Steering	Safety	Y2K Conformance	...
7010	Patterson	Neil				...
7011	Viescas	Michael	Y2K Conformance	Steering	Safety	...

Рис. 2.9. *Таблица с дубликатами полей, используемых для обозначения того, что значение встречается несколько раз*

Легко понять, почему эти повторяющиеся поля создадут проблемы. Одна из них касается реального количества полей Committee в таблице. Что если несколько сотрудников будут участниками четырех комитетов? Или более? Тогда потребуется добавить еще несколько полей Committee к таблице.

Другая проблема касается извлечения информации из таблицы. Как извлечь информацию о тех сотрудниках, которые в настоящее время принимают участие в работе комитета по решению проблемы 2000 года (Y2K Conformance)? Это не невозможно, но при извлечении такой информации возникнут затруднения. Потребуется выполнить три отдельных запроса для получения правильного ответа, поскольку нельзя быть точно уверенным, в каком из трех полей Committee сохранено значение Y2K Conformance. Таким образом, будет затрачено больше времени и усилий, чем необходимо в действительности.

Третья проблема касается сортировки данных. Невозможно отсортировать данные по комитетам каким-либо практически пригодным путем, и отсутствует способ, чтобы корректно упорядочить имена комитетов в алфавитном порядке. Хотя эти проблемы, возможно, кажутся незначительными, они могут серьезно помешать при попытке получить общее представление данных в некоторой упорядоченной форме.

Если вы тщательно изучите таблицу Employees на рис. 2.9, то поймете, что между сотрудниками и комитетами, участниками которых они являются, существует связь многие-ко-многим. Отдельный сотрудник может быть участником любого количества комитетов, а отдельный комитет может состоять из любого количества сотрудников. Поэтому эти дубликаты полей можно устранить тем же способом, который использовался для разрешения любых других связей многие-ко-многим,— путем создания связывающей таблицы. В случае таблицы Employees создайте связывающую таблицу с помощью копии первичного ключа (EmployeeID) и единственного поля Committee. Присвойте новой таблице соответствующее имя, например Committee_Members (Члены комитета), определите оба поля EmployeeID и Committee как составной первичный ключ, удалите поля Committee из таблицы Employees — и все сделано. На рис. 2.10 представлена исправленная таблица Employees и новая таблица Committee_Members.

Employees

EmployeeID	EmpFirstName	EmpLastName	EmpCity	<< ... >>
7004	Peacock	Samuel	Chico	...
7005	Kennedy	John	Portland	...
7006	Thompson	Sarah	Lubbock	...
7007	Callahan	David	Salem	...
7008	Buchanan	Andrea	Medford	...
7009	Smith	David	Fremont	...
7010	Patterson	Neil	San Diego	...
7011	Viescas	Michael	Redmond	...

Committee_Members

EmployeeID	Committee
7004	Steering
7005	Y2K Conformance
7005	Safety
7006	Safety
7006	Y2K Conformance
7006	Steering
7008	Y2K Conformance
7009	Steering

Рис. 2.10. *Исправленная таблица Employees и новая таблица Committee_Members*

Хотя мы устранили дубликаты полей, которые были в исходной таблице Employees, еще не все закончено. Принимая во внимание, что сотрудники и комитеты, в которых они работают, связаны между собой как многие-ко-многим, вполне можно задать вопрос: "А где же таблица Committees?" Пока еще ее нет! По счастью, у комитетов имеются некоторые другие характеристики, которые необходимо записать, например название комнаты, где собирается комитет, и день месяца, в который проводятся собрания. Поэтому следует создать реальную таблицу Committees, которая включает такие поля, как CommitteeID, CommitteeName, MeetingRoom и MeetingDay. Закончив создание новой таблицы, замените поле Committee в таблице Committee_Members на поле CommitteeID из новой таблицы Committees. Окончательные структуры представлены на рис. 2.11.

Определив такую структуру таблиц, вы сделали большое дело, потому что теперь отдельному сотруднику можно поставить в соответствие любое количество комитетов или отдельному комитету — любое количество сотрудников. Затем можно использовать запрос SQL для одновременного просмотра информации из всех трех таблиц.

Employees

EmployeeID	EmpFirstName	EmpLastName	EmpCity	<<другие поля>>
7004	Peacock	Samuel	Chico	...
7005	Kennedy	John	Portland	...
7006	Thompson	Sarah	Lubbock	...
7007	Callahan	David	Salem	...
7008	Buchanan	Andrea	Medford	...
7009	Smith	David	Fremont	...
7010	Patterson	Neil	San Diego	...
7011	Viescas	Michael	Redmond	...

Committee_Members

EmployeeID	CommitteeID
7004	103
7005	104
7005	102
7006	102
7006	104
7006	103
7008	104
7009	103

Committees

CommitteeID	CommitteeName	MeetingRoom	MeetingDay
100	Budget	11-C	Tuesday
101	Christmas	9-F	Monday
102	Safety	12-B	Monday
103	Steering	12-D	Tuesday
104	Y2K Compliance	Main-South	Wednesday

Рис. 2.11. *Окончательные структуры таблиц Employees и Committees*

В данный момент процесс тонкой настройки структур таблиц близок к завершению. Осталось только убедиться, что каждую запись в пределах таблицы можно идентифицировать уникальным образом и что саму таблицу можно идентифицировать во всей базе данных.

Идентификация — это ключ

Первичный ключ является одним из наиболее важных ключей в таблице, потому что он уникальным образом идентифицирует каждую запись в таблице и формально определяет эту таблицу во всей базе данных. Он также устанавливает связь между парой таблиц. Невозможно переоценить важность первичного ключа — каждая таблица базы данных должна его содержать!

По определению первичный ключ является полем или группой полей, которые однозначно идентифицируют каждую запись в таблице. Первичный ключ называется простым первичным ключом (или просто первичным ключом), когда он состоит из одного поля, и составным первичным ключом, когда состоит из двух или более полей. По возможности определяйте простой первичный ключ, поскольку он более эффективен и его намного легче применять при создании связи. Используйте составной первичный ключ, только когда это уместно (например, для определения и создания связывающей таблицы).

В качестве первичного ключа можно использовать существующее поле, пока оно удовлетворяет всем критериям приведенного далее контрольного списка. Если поле, которое предполагается использовать как первичный ключ, не соответствует *всем* критериям, используйте другое поле или определите новое в качестве первичного ключа для таблицы. Уделите некоторое время сейчас и используйте следующий контрольный список для определения, является ли первичный ключ вашей базы данных надежным.

- *Идентифицирует ли данное поле однозначным образом каждую запись в таблице?* Каждая запись в таблице представляет собой экземпляр предмета таблицы. Хороший первичный ключ гарантирует, что можно однозначно выделить любую запись в этой таблице или обратиться к ней из других таблиц в базе данных. Он также помогает избегать наличия в таблице дубликатов записей.

- *Уникальны ли значения ключа?* Пока значения первичного ключа не повторяются, записи в таблице не будут дублироваться.

- *Может ли это поле в принципе содержать неизвестные значения?* Это очень важно, потому что первичный ключ не должен содержать неизвестные значения. Если у этого поля есть даже малейшая возможность содержать неизвестные значения, его следует признать непригодным.

- *Может ли значение этого поля когда-либо быть необязательным?* Если да, то использовать это поле в качестве первичного ключа нельзя. Если значение поля может быть необязательным, значит оно в некотором случае может быть неизвестным, а первичный ключ не может содержать неизвестные значения.

Pilots

PilotID	PilotFirstName	PilotLastName	HireDate	Position	PilotAreaCode	PilotPhone
25100	John	Leverling	1994-07-11	Captain	206	555-3982
25101	David	Callahan	1994-05-01	Captain	206	555-6657
25102	David	Smith	1994-09-11	FirstOfficer	915	555-1992
25103	Kathryn	Patterson	1994-07-11	Navigator	972	555-8832
25104	Michael	Hernandez	1994-05-01	Navigator	360	555-9901
25105	Kendra	Bonnicksen	1994-09-11	Captain	206	555-1106

Рис. 2.12. *Является ли PilotID надежным первичным ключом?*

- *Является ли это поле составным?* Хотя к данному моменту следовало исключить все составные поля, лишняя проверка не повредит. Если составное поле было пропущено ранее, разделите его сейчас и попытайтесь использовать как первичный ключ другое поле.
- *Может ли значение этого поля когда-либо измениться?* Значение поля первичного ключа должно оставаться статичным, т. е. никогда не следует изменять значение первичного ключа, если действительно нет убедительной причины для этого. Когда значение поля подвергается произвольным изменениям, полю трудно оставаться в соответствии с другими пунктами этого контрольного списка.

Как уже говорилось, поле должно удовлетворять всем требованиям данного контрольного списка, прежде чем его можно будет использовать как первичный ключ. На рис. 2.12 PilotID служит первичным ключом таблицы Pilots. Но вопрос в том, удовлетворяет ли PilotID всем пунктам контрольного списка? Если да, то первичный ключ надежен. Но если нет, то необходимо либо изменить его, чтобы он соответствовал всем пунктам контрольного списка, либо выбрать в качестве первичного ключа другое поле.

В действительности поле PilotID является надежным первичным ключом, потому что оно соответствует всем пунктам контрольного списка. Но что произойдет, если нет поля, которое можно выбрать в качестве первичного ключа? Рассмотрим, например, таблицу Employees на рис. 2.13. Имеется ли в этой таблице поле, которое может служить первичным ключом?

Совершенно ясно, что в этой таблице такое поле отсутствует. За исключением EmpPhone, каждое поле содержит повторяющиеся значения. EmpZip, EmpAreaCode и EmpPhone содержат неизвестные значения. Поскольку значение каждого поля в этой таблице может меняться произвольным образом, то очевидно, что отсутствует поле, которое можно использовать в качестве первичного ключа. И что же теперь делать? — Создать искусственный первичный ключ.

Это произвольное поле, которое определяется и добавляется к таблице с единственной целью использовать его в качестве первичного ключа. Преимущество добавления произвольного поля состоит в том, что оно гарантированно удовлетворяет

Employees

EmpFirstName	EmpLastName	EmpCity	EmpState	EmpZip	EmpAreaCode	EmpPhone	HireDate
Peacock	Samuel	Chico	CA	95926			1998-12-31
Kennedy	John	Portland	OR	97208	503	555-2621	1998-05-01
Thompson	Michael	Redmond	WA	98052	425	555-2626	1998-09-11
Callahan	David	Salem	OR				1998-12-27
Buchanan	Andrea	Medford	OR	97501	541	555-2641	1998-05-01
Smith	Michael	Fremont	CA	94538	510	555-2646	1998-09-11
Peacock	Neil	San Diego	CA	92199	619	555-2541	1998-05-01
Kennedy	John	Redmond	WA	98052	425	555-2511	1998-09-11

Рис. 2.13. *Имеется ли в этой таблице первичный ключ?*

Employees

EmployeeID	EmpFirstName	EmpLastName	EmpCity	EmpState	EmpZip	<<другие поля>>
98001	Peacock	Samuel	Chico	CA	95926	...
98002	Kennedy	John	Portland	OR	97208	...
98003	Thompson	Michael	Redmond	WA	98052	...
98004	Callahan	David	Salem	OR		...
98005	Buchanan	Andrea	Medford	OR	97501	...
98006	Smith	Michael	Fremont	CA	94538	...
98007	Peacock	Neil	SanDiego	CA	92199	...
98008	Kennedy	John	Redmond	WA	98052	...

Рис. 2.14. *Таблица Employees с новым искусственным первичным ключом*

всем пунктам контрольного списка. Раз это поле добавлено к таблице, определите его как первичный ключ — и все в порядке! На рис. 2.14 представлена таблица Employees с новым искусственным первичным ключом, имя которого EmployeeID.

На этот момент сделано все для упрочения и тонкой настройки структур ваших таблиц. Теперь посмотрим, как можно гарантировать, что все связи между таблицами правильные.

Установка и исправление связей

Между двумя таблицами существует связь, если записям в первой таблице некоторым образом сопоставлены записи второй таблицы. Сами связи могут быть спроектированы по одному из трех типов: один-к-одному, один-ко-многим и многие-ко-многим. Каждый тип связей устанавливается специальным способом. Кратко повторим это.

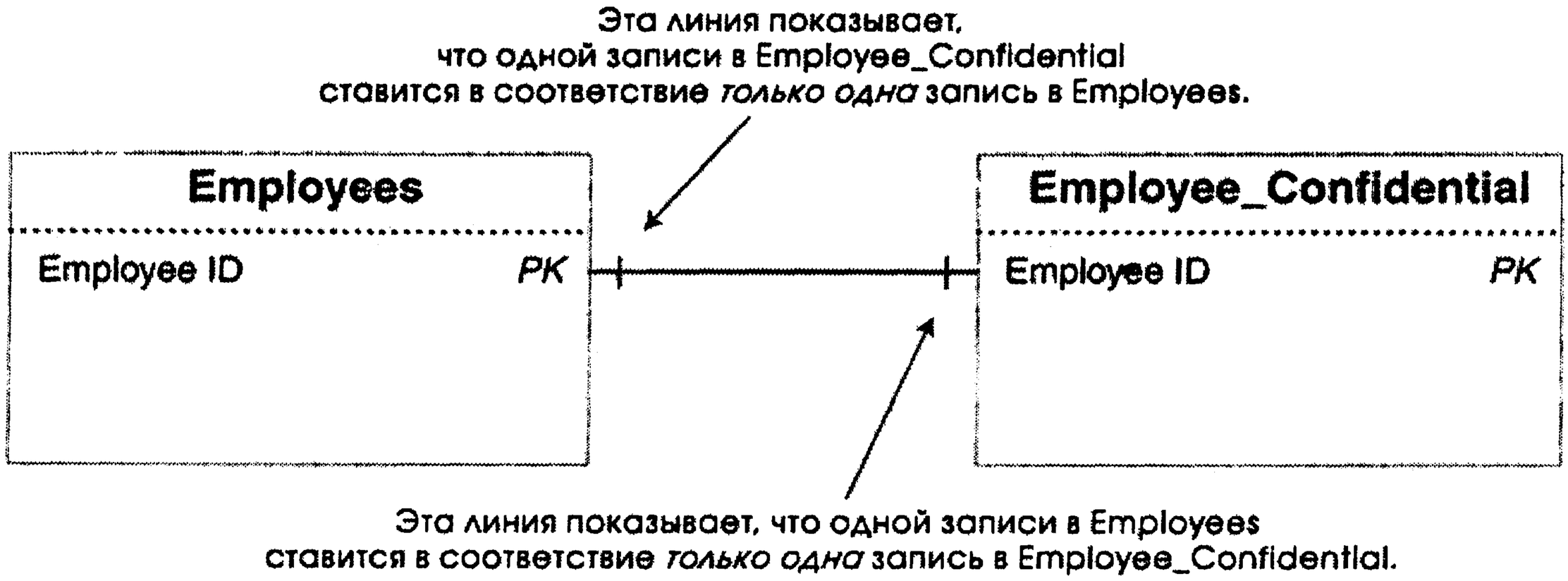


Рис. 2.15. *Изображение на диаграмме связи один-к-одному*

- Связь **один-к-одному** устанавливается, когда берется первичный ключ из "основной" таблицы и вставляется в "подчиненную" таблицу, где он становится внешним ключом. Это специальный тип связи, потому что во многих случаях внешний ключ также играет роль первичного ключа "подчиненной" таблицы. Можно изобразить эту связь, как показано на рис. 2.15.

- Связь **один-ко-многим** устанавливается, если берется первичный ключ таблицы на стороне "один" и вставляется в таблицу на стороне "многие", где он становится внешним ключом. На рис. 2.16 представлена диаграмма с этим типом связи.

> **Внимание!** Символы диаграммы, представленные в этом разделе, являются частью метода, изложенного в книге Хернандеса *Database Design for Mere Mortals*.

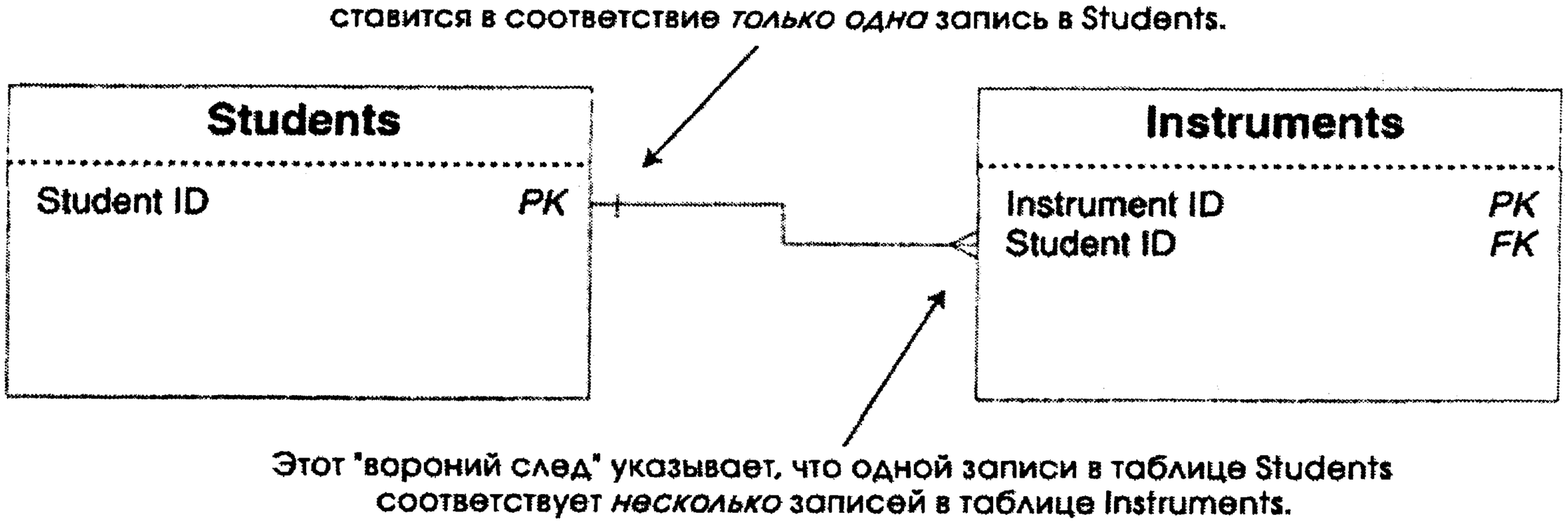


Рис. 2.16. *Представление в виде диаграммы связи один-ко-многим*

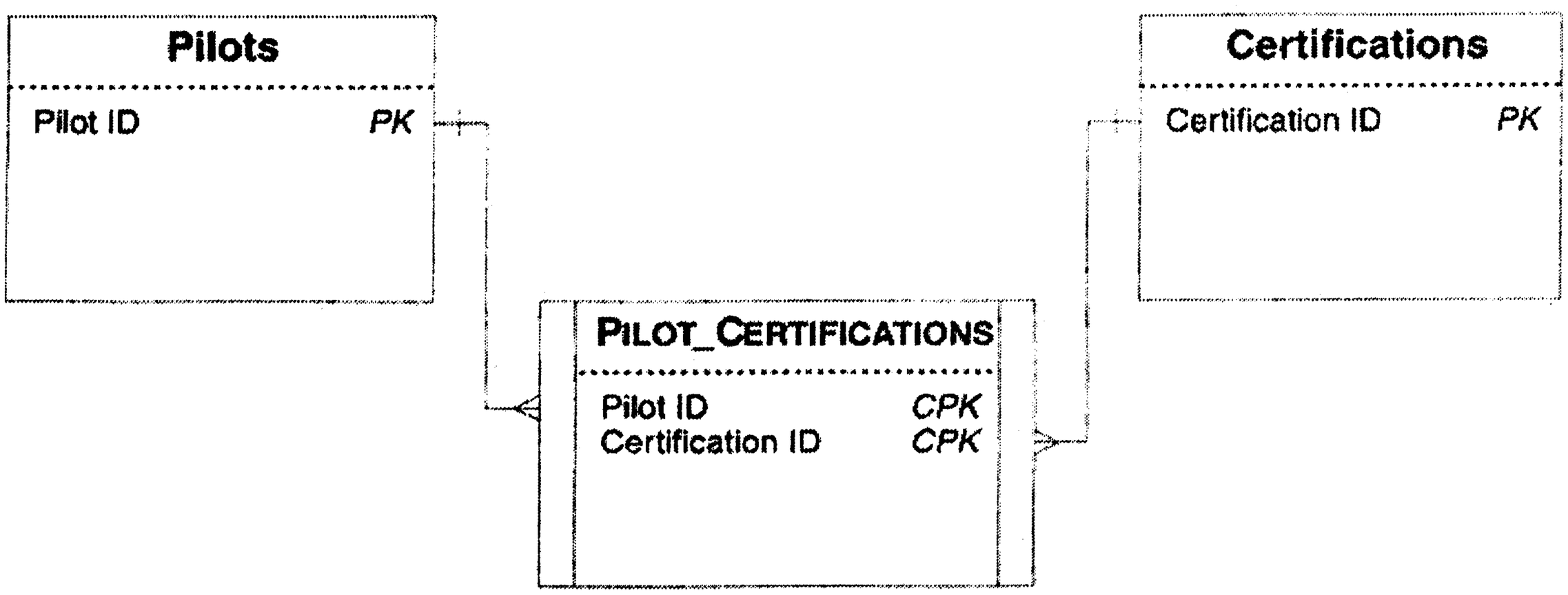


Рис. 2.17. *Представление в виде диаграммы связи многие-ко-многим*

- Связь **многие-ко-многим** устанавливается путем создания связывающей таблицы. Для этого берется копия первичного ключа каждой таблицы и используется для формирования структуры новой таблицы. Эти поля обычно служат двум целям: вместе они образуют составной первичный ключ связывающей таблицы; по отдельности каждый из них служит внешним ключом. Диаграмма этой связи представлена на рис. 2.17.

Для уверенности в том, что связи между таблицами в базе данных действительно надежные, необходимо для каждой связи определить характеристики. Они указывают, что произойдет при удалении записи, а также тип и степень участия таблицы в связи.

Прежде чем начать обсуждение характеристик связи, необходимо уточнить один момент. Приведенные ниже характеристики представлены в пределах родового и логического фрейма связи. Эти характеристики важны, поскольку они позволяют усилить реляционную целостность. Однако способ их реализации изменяется в зависимости от программного обеспечения базы данных. Необходимо изучить документацию по программному обеспечению базы данных, чтобы определить, поддерживаются ли эти характеристики, и если да, то как их можно реализовать.

Определение правила удаления

Правило удаления определяет, что произойдет, когда пользователь выдаст запрос на удаление записи в "исходной" таблице со связью один-к-одному или в таблице на стороне "один" в связи один-ко-многим. Определение этого правила может защитить от появления "зависших" записей. (Под "зависшими" записями понимаются записи в "подчиненной" таблице связи один-к-одному, у которой отсутствуют связанные с ней записи в "первичной" таблице, или записи в таблице на стороне "многие" связи один-ко-многим, у которых отсутствуют связанные с ними записи в таблице на стороне "один".)

Можно установить два типа **правил удаления** для связи: *ограничивающее* и *каскадное*.

- **Ограничивающее правило удаления** не позволяет удалять указанную в запросе запись, когда в "подчиненной" таблице связи один-к-одному или в таблице на стороне "многие" связи один-ко-многим имеются зависимые записи. Необходимо удалить все зависимые записи, *прежде* чем удалять запись, указанную в запросе. Этот тип правила удаления используется как само собой разумеющееся.
- Когда действует **каскадное правило удаления**, можно удалять запись, указанную в запросе, и при этом будут также удалены все зависимые записи в "подчиненной" таблице связи один-к-одному или в таблице на стороне "многие" связи один-ко-многим. Пользуйтесь этим правилом очень осторожно, поскольку иначе можно ликвидировать таблицу, удалив записи, которые в действительности вы хотели сохранить!

Независимо от выбранного правила удаления всегда тщательно проверяйте свои связи, чтобы определить, какой тип связи подходит. Один очень простой вопрос поможет решить, какое правило использовать. Вначале выберите пару таблиц, а затем спросите себя: "Если удаляется запись в [имя "первичной" таблицы или на стороне "один"], должны ли также удаляться зависимые записи в [имя "подчиненной" таблицы или на стороне "многие"]?"

Этот вопрос поставлен в общем смысле, чтобы можно было понять предпосылки, стоящие за ним. Чтобы применить этот вопрос в действительном смысле, замените фразы, заключенные в квадратные скобки, на имена таблиц. Вопрос будет выглядеть, возможно, так: "Если удаляется запись в таблице Committees, должны ли также удаляться зависимые записи в таблице Committee_Members?"

Если ответ "нет", то используйте ограничивающее правило удаления; в противном случае используйте каскадное правило. В конечном счете ответ на этот вопрос в большой мере зависит от того, как используются данные, сохраненные в базе данных. Именно поэтому следует тщательно изучить связи и убедиться, что выбрано верное правило. На рис. 2.18 показано, как изобразить на диаграмме правило удаления для этой связи. Для ограничивающего правила удаления используется обозначение (R), а для каскадного — (C).

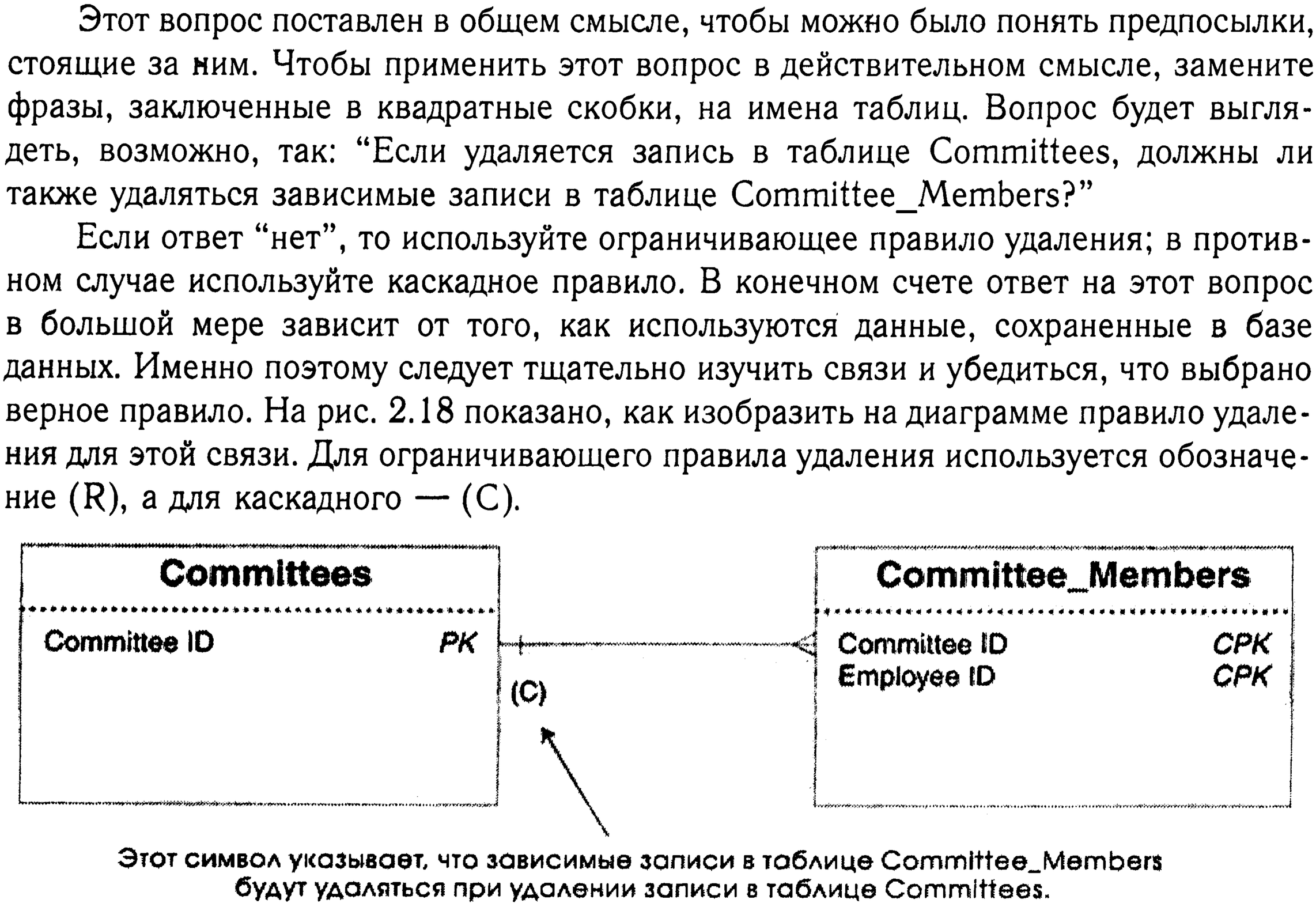


Рис. 2.18. *Представление на диаграмме правил удаления для таблиц Committees и Committee_Members*

Установка типа участия

При определении связи между парой таблиц каждая таблица участвует в ней конкретным образом. *Тип участия*, присвоенный указанной таблице, определяет, должна ли запись присутствовать в этой таблице до того, как можно внести запись в другую таблицу. Имеется два типа участия:

- **Обязательный** — В этой таблице должна существовать по меньшей мере одна запись, прежде чем можно будет вносить записи в другую таблицу.
- **Необязательный** — Отсутствует требование о наличии любых записей в этой таблице до внесения каких-либо записей в другую таблицу.

Тип участия, выбираемый для пары таблиц, зависит главным образом от логической схемы деятельности вашей организации. Предположим, вы работаете в большой компании, состоящей из нескольких отделов. В созданной вами БД имеются таблицы Employees (Сотрудники), Departments (Отделы) и Department_Employees

Employees

EmployeeID	EmpFirstName	EmpLastName	EmpCity	<<другие поля>>
7004	Peacock	Samuel	Chico	...
7005	Kennedy	John	Portland	...
7006	Thompson	Sarah	Lubbock	...
7007	Callahan	David	Salem	...
7008	Buchanan	Andrea	Medford	...
7009	Smith	David	Fremont	...
7010	Patterson	Neil	San Diego	...
7011	Viescas	Michael	Redmond	...

Department_Employees

EmployeeID	DepartmentID	Position
7004	1000	Head
7005	1000	Floater
7005	1001	Floater
7007	1001	Staff
7008	1001	Head
7009	1003	Floater
7010	1002	Head
7011	1004	Head

Departments

DepartmentID	DepartmentName	Floor
1000	Accounting	5
1001	Administration	5
1002	HumanResources	7
1003	InformationServices	6
1004	Legal	7

Рис. 2.19. *Таблицы Employees, Departments и Department_Employees*

(Сотрудники отдела). Вся информация, касающаяся сотрудников, находится в таблице Employees, а вся информация об отделах — в таблице Departments. Таблица Department_Employees является связывающей таблицей, которая позволяет ставить в соответствие указанному сотруднику любое количество отделов. Эти таблицы показаны на рис. 2.19.

На последнем совещании было решено сформировать новый отдел исследований и разработок. Теперь вы хотите удостовериться, что добавили новый отдел в таблицу Departments и сможете назначать персонал этому отделу в таблице Department_Employees. Именно здесь начинают действовать характеристики типа участия. Установите для таблицы Departments тип участия обязательный, а для Department_Employees — необязательный. Определяя такие настройки, можно гарантировать, что отдел уже будет существовать в таблице Departments, прежде чем вы сможете назначить каких-либо сотрудников для этого отдела в таблице Department_Employees.

Как и в случае с правилом удаления, тщательно исследуйте каждую связь для определения соответствующей установки типа участия для каждой таблицы в связи. На рис. 2.20 тип участия представлен в виде диаграммы.

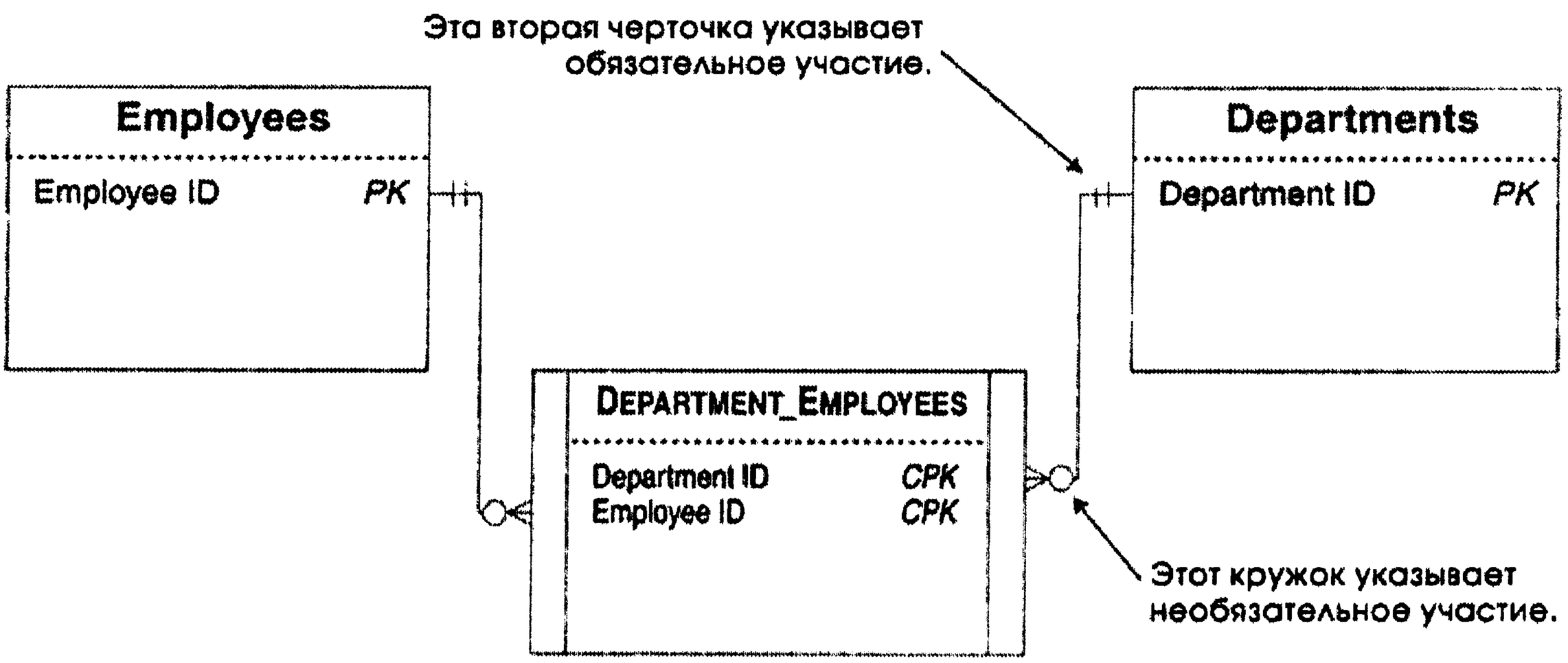


Рис. 2.20. *Представление в виде диаграммы типа участия для таблиц Departments и Department_Employees*

Установка степени участия

Теперь необходимо вычислить, *до какой степени* каждая таблица будет участвовать в связи. Это выполняется путем определения минимального и максимального количества записей в одной таблице, которые могут быть сопоставлены одной записи в другой таблице. Этот процесс называется идентификацией *степени участия* для таблицы. Степени участия представляется двумя числами, разделенными запятыми и заключенными в скобки. Первое число указывает минимально возможное количество зависимых записей, а второе — максимальное количество зависимых записей.

Например, степень участия "(1,12)" указывает, что минимальное количество записей, которое может быть поставлено в соответствие, равно единице, а максимальное — двенадцати.

Выбранная степень участия для различных таблиц базы данных во многом зависит от того, как в организации просматриваются и используются данные. Предположим, что вы — регистрирующий агент агентства талантов и что в вашей базе данных две таблицы: Agents (Агенты) и Entertainers (Эстрадные артисты). Между этими таблицами имеется связь один-ко-многим. Одной записи в таблице Agents можно поставить в соответствие много записей из таблицы Entertainers, но одной записи в таблице Entertainers можно сопоставить только одну запись в таблице Agents. В данном случае гарантируется, что эстрадный артист приписывается только одному агенту. (Мы намеренно избегаем возможности противопоставления одного агента другому.)

Как оказалось, босс хотел гарантировать, что все его агенты хорошо встряхнутся, получая хорошие комиссионные, и хотел бы подавить соперничество между агентами до абсолютного минимума. По этой причине он установил новую политику, состоящую в том, что отдельный агент может представлять максимум шесть эстрадных артистов. Для реализации этой новой политики он установил степень участия для обеих таблиц следующим образом:

Agents	(1,1) — Эстрадный артист может быть связан с одним и только с одним агентом.
Entertainers	(0,6) — Раз агент не должен вообще ставиться в соответствие эстрадному артисту, то в конкретный момент времени данному агенту не может быть сопоставлено более шести эстрадных артистов.

На рис. 2.21 показано, как изобразить на диаграмме степень участия для этих таблиц.

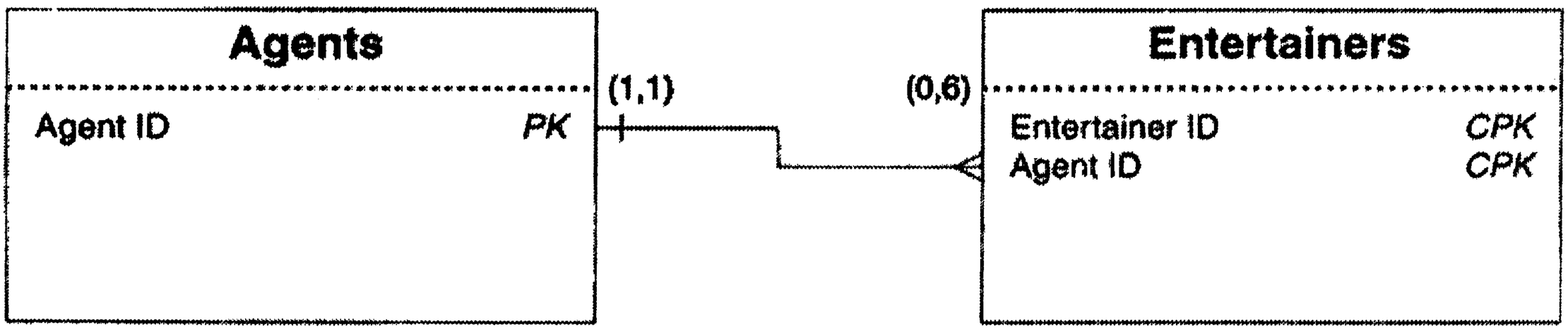


Рис. 2.21. *Представление в виде диаграммы степени участия для таблиц Agents и Entertainers*

Как и для других характеристик связи, тщательно изучите каждую связь, чтобы можно было идентифицировать соответствующую степень участия для каждой таблицы. Предположим, что все характеристики связи для таблиц Agents и Entertainers

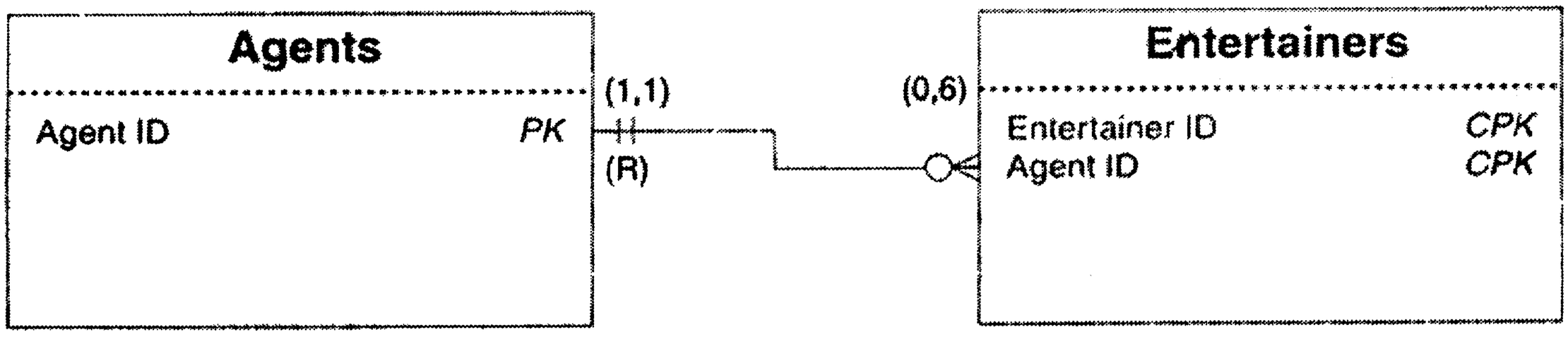


Рис. 2.22. *Диаграмма со всеми характеристиками связи для таблиц Agents и Entertainers*

установлены. На рис. 2.22 показано, как добавить степень участия к диаграмме связей и как она будет выглядеть после этого. Теперь ее можно использовать для идентификации типа связи, правила удаления для связи, типа и степени участия для каждой таблицы.

И это все?

Используя методы, рассмотренные в данной главе, мы приняли базовые меры для обеспечения уровня целостности данных в БД. На следующем шаге необходимо изучить способ просмотра и использования данных в вашей организации, чтобы определить деловые правила и применить их к БД. Но чтобы действительно получить максимальную пользу от базы данных, следует вернуться к началу и бегло просмотреть процесс проектирования базы данных, используя хорошую методику проектирования. К сожалению, эти вопросы выходят за рамки данной книги. Однако хорошую методику проектирования можно изучить по таким книгам, как *Database Desiogn for Mere Mortals* Майкла Дж. Хернандеса или *Handbook of Relational Database Design* (Barbara Von Halle, Candace C. Fleming). Запомните следующее: чем лучше проработана структура базы данных, тем легче извлекать информацию из данных и создавать программы приложений.

Итоги

Эту главу мы начали с краткого обсуждения, почему следует обеспечить хороший уровень структуры БД. Плохо спроектированные таблицы могут вызвать многочисленные проблемы, одна из которых касается целостности данных.

Затем мы обсудили тонкую настройку полей каждой таблицы. Очень важно присвоить полям хорошие имена: это гарантирует, что каждое имя имеет смысл и реально помогает найти скрытые проблемы в самой структуре полей. Теперь вы можете выполнить тонкую настройку своих структур полей и гарантировать, что они удовлетворяют некоторым простым правилам. Правила предписывают, чтобы каждое поле представляло отдельную характеристику предмета таблицы, содержало только одно значение и никогда не хранило вычисления. Мы также обсудили проблемы, касающиеся составных и многозначных полей, и способы их разрешения.

Мы также рассмотрели тонкую настройку таблиц. Имена таблиц — это в такой же степени важный вопрос, как и имена полей, во многих случаях по тем же самым причинам. Теперь вы знаете, как присвоить своим таблицам имена, поддающиеся интерпретации, и гарантировать, что каждая таблица представляет только отдельный предмет. Мы привели набор правил, которые следует использовать для обеспечения надежности структуры каждой таблицы. Хотя некоторые из этих правил повторяют правила, используемые для тонкой настройки структур полей, их применение для тонкой настройки структур таблиц добавляет дополнительный уровень гарантий, обеспечивающий, насколько возможно, абсолютную надежность структур таблиц.

Кроме того, мы обсудили первичные ключи и важность определения первичного ключа для каждой таблицы базы данных. Первичный ключ должен удовлетворять определенному набору характеристик, и поле, которое будет служить первичным ключом таблицы, необходимо выбирать с особой тщательностью. Можно создать искусственный первичный ключ, если в таблице отсутствует поле, которое удовлетворяет полному набору характеристик для первичного ключа.

В конце главы мы обсудили вопрос оптимизации связей и представили три их типа на диаграмме. Теперь вы знаете, как определить и обозначить на диаграмме правило удаления для связи. Это правило помогает защититься от "зависших" записей. Последние два вопроса касались типа и степени участия каждой таблицы в связи. Участие таблицы может быть обязательным или необязательным, и можно установить конкретный диапазон для количества зависимых записей между каждой из таблиц.

3

Краткая история SQL

"Существует только одна религия,
хотя во множестве вариантов".

— Джордж Бернард Шоу
Приятные и неприятные пьесы

Вопросы, рассматриваемые в данной главе:

- Истоки SQL
- Ранние реализации
- "... а затем был Стандарт"
- Эволюция стандарта ANSI/ISO
- Что готовит будущее
- Зачем учить SQL
- Итоги

Изложение истории всегда включает неопределенные и неоднозначные оценки различных событий, политические интриги и человеческие слабости. История SQL не отличается в этом смысле от истории любых других предметов. В той или иной форме SQL использовался почти с самого момента появления реляционной модели, и существует несколько подробных отчетов о его длительном (и в нескольких вариантах) существовании. В данной главе мы более пристально рассмотрим происхождение, развитие и будущее этого языка баз данных. Перед нами две цели. Первая — дать представление о том, как SQL развивался в язык, используемый сегодня большинством систем реляционных баз данных, а вторая — объяснить, почему важно знать, как использовать SQL.

Истоки SQL

Д-р Э. Ф. Кодд представил реляционную модель базы данных в 1970 г. Вскоре после этого поворотного момента такие организации, как университеты и исследовательские лаборатории, начали прилагать усилия к разработке языка, который мог бы использоваться как фундамент для СУБД, поддерживающих реляционную модель. Исходная работа привела в середине 70-х годов к разработке нескольких языков, а в результате последующих усилий появились SQL и базы данных, основанные на

SQL, используемые сегодня. Но где истоки SQL? Как он развивался? Какое у него будущее? Для ответа на эти вопросы мы должны начать наш рассказ из исследовательской лаборатории IBM в Сан-Хосе, штат Калифорния.

В начале 1970-х годов IBM начала разрабатывать большой исследовательский проект System/R. Этот проект должен был доказать жизнеспособность реляционной модели и дать некоторый опыт проектирования и реализации реляционной базы данных. Первоначальные попытки исследователей в 1974-75 гг. достигли успеха: удалось создать самый базовый прототип реляционной базы данных.

Кроме того, разработчики также трудились над определением языка баз данных. Можно утверждать, что работа, выполненная в этой лаборатории, является наиболее коммерчески значащей из всех первоначальных попыток определения такого языка. В 1974 г. д-р Дональд Чемберлин и его коллеги разработали структурированный английский язык запросов (Structured English Query Language, SEQUEL, произносится "сиквел"). Этот язык позволял осуществлять запросы к реляционной базе данных, используя четко определенные предложения, близкие к английскому языку. Д-р Чемберлин и его сотрудники вначале реализовали этот новый язык в прототипе базы данных с наименованием SEQUEL-XRM.

Первоначальные отклики и успех SEQUEL-XRM поддержали намерения д-ра Чемберлина и его сотрудников продолжать исследования. В 1976-77 гг. они полностью пересмотрели SEQUEL и назвали новую версию SEQUEL/2. Однако впоследствии, по юридическим причинам, наименование SEQUEL было изменено на SQL (Structured Query Language) — оказывается, сокращение SEQUEL уже кем-то использовалось. До сих пор многие все еще произносят название языка как "сиквел", хотя широко принятым "официальным" произношением является "эскюэль". В SQL появилось несколько новых возможностей, таких как поддержка многотабличных запросов и одновременный доступ нескольких пользователей к совместно используемым данным.

Вскоре после появления SQL IBM начала новый и еще более амбициозный проект, нацеленный на создание прототипа БД, который доказал бы коммерческую пригодность реляционной модели. Новый прототип был назван "System R" и основывался на большом подмножестве SQL. После завершения большей части исходной работы по разработке IBM инсталлировала System R на нескольких собственных системах и вычислительных комплексах избранных клиентов для тестирования и оценки. В System R и SQL было внесено множество изменений, основанных на опыте и откликах пользователей. IBM закрыла проект в 1979 г., сделав вывод, что реляционная модель является действительно ценной технологией для коммерческих БД.

Внимание! Одним из наиболее важных успехов, приписываемых этому проекту, является разработка SQL. Но "предком" SQL в действительности был язык, созданный в исследовательских целях, который назывался SQUARE (Specifying Queries As Relational Expressions — Определение запросов как реляционных выражений). Этот язык был разработан в 1975 г. (т. е. еще до запуска проекта System R) и был спроектирован для реализации реляционной алгебры в предложениях, близких к английскому языку.

Ранние реализации

Работа, выполненная в исследовательской лаборатории IBM в 70-е годы, вызвала большой интерес различных технических журналов, и достоинства новой реляционной модели живо обсуждались на семинарах по технологии баз данных. Ближе к концу десятилетия стало ясно, что IBM сильно заинтересована и активно участвует в разработке продуктов, основанных на технологии реляционных баз данных и SQL. Это, конечно, заставило многие компании задуматься о том, как скоро IBM сможет выпустить свой первый продукт. Некоторые поставщики рассудили, что следует как можно скорее начать работу над своими собственными продуктами, а не дожидаться, пока IBM захватит весь рынок.

В 1977 г. группой инженеров из городка Мемо-Парк, штат Калифорния, была образована компания Relational Software Inc. с целью построения нового продукта реляционной базы данных, основанного на SQL. Они назвали свой продукт Oracle. Relational Software Inc. выпустила этот продукт в 1979 г., опередив первый продукт IBM на рынке на два года и представив первую коммерчески доступную систему управления реляционными базами данных (СУРБД). Одно из преимуществ СУРБД Oracle заключалось в том, что она работала на мини-компьютерах VAX компании Digital, а не на более дорогих мэйнфреймах IBM. Впоследствии Relational Software Inc. была переименована в Oracle Corporation и на сегодняшний день она является одним из ведущих поставщиков СУРБД.

Тем временем Майкл Стоунбрэкер, Эжен Вонг и несколько других преподавателей лаборатории вычислительных систем Калифорнийского университета в Беркли также занимались исследованиями технологии реляционных баз данных. Подобно группе IBM, они разработали прототип реляционной базы данных и дали своему продукту имя INGRES. В состав INGRES был включен язык базы данных, названный языком запросов (Query Language, QUEL), который в сравнении с SQL был намного более структурированным, но в меньшей степени использовал операторы, близкие к разговорному английскому языку. В конце концов, когда стало ясно, что SQL превратился в стандартный язык баз данных, INGRES была преобразована в СУРБД на базе SQL. Несколько профессоров расстались с Беркли в 1980 г. и учредили Relational Technology Inc., а в 1981 г. они объявили о выходе первой коммерческой версии INGRES. Компания Relational Technology прошла через несколько преобразований и в настоящее время входит в Computer Associates International, Inc. Сегодня INGRES все еще является одним из лидирующих продуктов баз данных в этой области.

Теперь возвратимся к самому началу истории IBM. В 1981 г. IBM анонсировала свою собственную СУРБД с именем SQL/Data System (SQL/DS), а в 1982 г. начала ее поставки. В 1983 г. она представила новую версию SQL/DS для операционной системы VM/CMS (одной из нескольких ОС, поставляемых IBM для своих больших вычислительных систем) и анонсировала новый продукт СУРБД с именем Database 2 (DB2), который можно было использовать на мэйнфреймах IBM под управлением MVS — основной операционной системы IBM на то время. Впервые поставленная

в 1985 г., DB2 стала главной СУРБД от IBM, а ее технология была внедрена во всю серию продуктов IBM. Кстати, с IBM с тех пор ничего не произошло — она все еще IBM.

За прошедшие более чем 25 лет мы увидели как то, что начиналось, как исследование по проекту System R, превратилось в силу, которая оказывает влияние почти на все уровни сегодняшнего бизнеса и развилось в отрасль с оборотом в несколько миллиардов долларов.

"... а затем был Стандарт"

В суматохе, окружавшей разработку баз данных, задумывался ли хоть кто-нибудь о стандартизации? Хотя эта идея буквально носилась в воздухе среди сообщества, связанного с базами данных, отсутствовало какое бы то ни было единодушие или согласие как в отношении того, кто должен установить стандарт, так и в отношении того, на каком диалекте он должен основываться. Поэтому каждый поставщик продолжал разрабатывать и улучшать свой собственный продукт базы данных, надеясь, что он и, следовательно, его диалект SQL станут отраслевым стандартом.

Отзывы и требования клиентов принуждали многих поставщиков включать определенные элементы в свой диалект SQL, и со временем появился неофициальный стандарт. Это были миниатюрные спецификации в сравнении с современными стандартами, поскольку они содержали только те элементы, которые были сходными в различных диалектах SQL. Однако эти спецификации обеспечивали пользователей БД исходным набором критериев, позволяющим оценивать различные программы баз данных на рынке, а также предоставляли пользователям небольшой набор основных сведений, которые они могли использовать при переходе из одной программы базы данных к другой.

В 1982 г. Американский национальный институт стандартов (American National Standards Institute, ANSI) отреагировал на растущую потребность в официальном стандарте языка для реляционных баз данных, поручив своему техническому комитету по базам данных (комитету X3H2) в составе организации X3 разработать предложения для такого стандарта. X3 является одной из многих организаций под контролем ANSI. В свою очередь, X3H2 Database (X3H2) является просто одним из многих технических комитетов, который отчитывается перед X3. X3H2 состоит из экспертов в области баз данных и представителей почти от всех основных разработчиков БД на основе SQL. В начальный период комитет просматривал и обсуждал преимущества и недостатки различных предлагаемых языков, а также начал работу над стандартом, основанном на QUEL — языке баз данных для INGRES. Но давление со стороны рынка и значительный рост активности IBM в отношении SQL склонили комитет взять за основу своего предложения SQL. Стандарт, предложенный комитетом X3H2, в значительной степени был основан на диалекте SQL для IBM DB2.

Комитет продолжил работу над несколькими версиями своего стандарта в течение еще двух лет и в некоторой степени улучшил его. Однако результатом этих

улучшений стало одно печальное обстоятельство: новый стандарт перестал быть совместимым с существующими основными диалектами SQL. Вскоре X3H2 осознал, что изменения, внесенные в SQL, не настолько улучшили его, чтобы в достаточной степени служить оправданием несовместимости, поэтому комитет возвратился к исходной версии стандарта.

В 1986 г. ANSI принял стандарт X3H2, получивший название “ANSI X3.135-1986 — Язык баз данных SQL” и стал широко известен под именем “SQL/86". Хотя X3H2 внес некоторые незначительные изменения в свой стандарт до того, как он был принят ANSI, SQL/86 просто определял минимальный набор требований, являющихся "наименьшим общим знаменателем", которому должны соответствовать поставщики баз данных. По существу, он присвоил “официальный” статус элементам, которые были сходными в различных диалектах SQL и уже были реализованы многими поставщиками баз данных. Новый стандарт наконец обеспечил специальный базис, на основе которого могли в дальнейшем разрабатываться язык и его конечные продукты.

Международная организация по стандартизации (International Organization for Standardization, ISO) в 1987 г. утвердила свой собственный документ (который в точности соответствовал ANSI SQL/86) в качестве международного стандарта и опубликовала его как документ “ISO 9075-1987 — Язык баз данных SQL”. (На оба стандарта принято ссылаться просто как на SQL/86.) Международное сообщество разработчиков баз данных смогло теперь опираться на те же стандарты, что и компании США. Но несмотря на то, что SQL приобрел статус официального стандарта, язык был далек от состояния завершения.

Развитие стандарта ANSI/ISO

Вскоре SQL/86 был подвергнут критике со стороны правительства, периодических журналов и таких экспертов, как С. Дж. Дэйт. Причиной этой критики являлись: избыточность синтаксиса SQL (было несколько способов определения того же самого запроса), отсутствие поддержки для определенных реляционных операторов и отсутствие реляционной целостности. Хотя X3H2 знал об этих проблемах еще до опубликования SQL/86, он считал, что лучше выпустить стандарт раньше, хотя и в недоработанном виде, чем вообще не иметь стандарта.

Как ISO, так и ANSI отреагировали на критику, касающуюся реляционной целостности, приняв исправленные версии стандартов. ISO опубликовала “ISO 975:1989 — Язык SQL для баз данных с расширением целостности" в 1989 г., тогда как ANSI в тот же год, но позднее, принял свой стандарт “X3.135-1989 — Язык SQL для баз данных с расширением целостности”, который также часто указывается как SQL/89. Тем не менее работа комитета ANSI, длящаяся годы, не закончена даже сейчас: X3H2 все еще пытается решить важный вопрос, поставленный правительством.

Некоторые правительственные пользователи выражали неудовольствие тем, что спецификации, поясняющие, как встроить SQL в обычный язык программирования, не были явными компонентами стандарта (спецификации были включены, но они

отсылались в приложение). Они были обеспокоены тем, что поставщики могли не поддерживать переносимые реализации вложенного SQL, поскольку в рамках стандартов для них отсутствовали специальные требования для обеспечения этого. X3H2 отреагировал на это разработкой второго стандарта, который требовал соответствия спецификациям с учетом обеспечения вложенности, и опубликовал его как "ANSI X3.168-1989 — Язык для баз данных с вложенным SQL". Интересно отметить, что ISO решила не публиковать соответствующий стандарт вследствие отсутствия указанной проблемы в международном сообществе. Это означает, что у ISO не имелось спецификаций для встраивания SQL в язык программирования, и эта ситуация изменилась только после опубликования ISO ее стандарта SQL/92.

SQL/86 и SQL/89 были далеко не завершенными стандартами — в них отсутствовали самые основные возможности, необходимые для коммерческих СУБД. Например, стандарт не определял способ внесения изменений в структуру базы данных (и в саму систему базы данных) после того, как она была определена. Никто не может изменить или удалить какие-либо структурные компоненты (например, таблицы или столбцы) или сделать любые изменения в защите базы данных. Например, можно создать таблицу (CREATE), но стандарт не включает определения команды DROP (Удалить) для удаления таблицы или команды ALTER (Изменить) для ее изменения. Также можно предоставить (GRANT) защищенный доступ к таблице, но стандарт не включает определения команды REVOKE (Отмена), чтобы разрешить удаление полномочий доступа. По иронии судьбы эти возможности предусматривались всеми коммерческими базами данных, основанными на SQL. Однако они не были включены ни в один стандарт, поскольку каждый поставщик реализовывал их по-разному. Другие возможности были широко реализованы во многих базах данных, основанных на SQL, но были пропущены в стандартах. И опять это был вопрос различий в реализации.

К моменту принятия стандарта SQL/89 как ANSI, так и ISO уже работали над основными версиями для SQL, которые должны было сделать его полным и устойчивым языком. Новая версия должна была называться SQL/92 и включать в себя возможности, которые уже были широко реализованы самыми основными поставщиками баз данных. Но одной из главных целей как ANSI, так и ISO было избежать принятия еще одного стандарта, который просто объединял бы общие моменты различных реализаций. В результате было решено включить возможности, которые еще не получили широкого распространения, и добавить новые, которые, по существу, выходили за рамки возможностей, реализованных в то время.

ANSI и ISO опубликовали свои новые стандарты SQL: "X3.135-1992 — Язык баз данных SQL" и "ISO/IEC 9075:1992 — Язык баз данных SQL" в октябре 1992 г. Документ SQL/92 значительно больше по объему, чем SQL/89, и намного шире по содержанию. Например, в нем предусматриваются средства изменения структуры базы данных после ее определения; поддержка дополнительных операций для манипулирования символьными строками, значениями даты и времени; определяются возможности дополнительной защиты. SQL/92 обозначает большой шаг вперед в сравнении со всеми его предшественниками.

По иронии судьбы в настоящее время еще нет СУБД, полностью поддерживающих и реализующих все из SQL/92. Это в большой степени является следствием сложности многих возможностей нового стандарта. Но было бы неблагоразумно думать, что производители смогут ввести эти возможности за одну или две версии своих продуктов. Помимо того, что реализация многих сложных возможностей займет некоторое время и потребует усилий, разработчики СУБД, движимые рынком, вынуждены решать другие насущные задачи в отношении своих продуктов, как, например, повышение производительности, увеличение надежности и обеспечение расширенной системной интеграции. Это не столько оправдание, сколько объяснение того, как это происходит в отрасли, связанной с базами данных.

К счастью, комитеты по стандартам в некоторой степени предугадали эту ситуацию. Для того чтобы облегчить плавный и постепенный переход к новому стандарту, ANSI и ISO определили три уровня SQL/92.

Начальный SQL	Подобно SQL/89, этот уровень включает возможности для облегчения перехода от SQL/89 к SQL/92, а также возможности, которые корректируют ошибки в стандарте SQL/89. Идея состояла в том, что этот уровень будет легче всего реализовать, поскольку большинство его возможностей уже были широко представлены в существующих продуктах.
Переходный SQL	Этот уровень включает в себя большинство возможностей нового стандарта. Такое решение обоих комитетов было основано на нескольких факторах. Общая цель состояла в расширении стандарта таким образом, чтобы SQL лучше поддерживал концепции реляционной модели, и в переопределении синтаксиса, который был неоднозначным или нечетким. Возможностям, затребованным пользователями СУБД на основе SQL, было уделено большое внимание, с тем чтобы они удовлетворяли этим целям и относительно просто реализовывались большинством пользователей. Этот уровень должен был гарантировать, что продукты, в разумных пределах, будут иметь как можно более надежную реализацию. Это все еще тот уровень, который большинство поставщиков баз данных пытаются реализовать в своих продуктах.

Полный SQL — Это полная реализация стандарта SQL/92. Она обязательно включает наиболее сложные возможности, которые были пропущены на первых двух уровнях. Хотя эти возможности считаются важными для удовлетворения потребностей клиентов или для последующей "чистки" языка, их трудно немедленно реализовать большинству поставщиков. К сожалению, соответствие "полному SQL" пока еще не является требованием, поэтому необходимо некоторое время, прежде чем мы сможем ожидать появления СУБД, полностью реализующих стандарт.

Многие разработчики СУБД продолжают работать над реализацией возможностей SQL/92, и при этом они также разрабатывают и реализуют свои собственные возможности. Их дополнения к стандарту SQL называются расширениями. Например, разработчик может предоставить более шести типов данных, определенных в SQL/92. Хотя эти расширения предоставляют больше функциональных возможностей в рамках данного продукта и позволяют продукту выделяться на рынке, они являются помехами, поскольку заставляют диалект SQL каждого поставщика все больше отклоняться от исходного стандарта. Это, в свою очередь, мешает разработчикам баз данных создавать переносимые приложения, которые можно выполнять из любой базы данных. Возможно, эта ситуация изменится после выпуска следующей версии стандарта.

Другие стандарты SQL

Стандарт ANSI/ISO SQL/92 наиболее широко принят сегодня. Конечно, существуют и другие стандарты, которые включают SQL в той или иной форме. Ниже приведены самые важные из них.

X/Open — Группа европейских поставщиков (которая вся в целом называется X/OPEN) разработала ряд стандартов, которые могли бы помочь установить среду переносимых приложений, основанную на UNIX. Возможность "переноса" приложения с одной компьютерной системы на другую без изменений является важным вопросом на европейском рынке. Хотя они и приняли SQL как часть набора стандартов, их версии отклоняются от стандарта ANSI/ISO в нескольких аспектах.

SAA — Компания IBM разработала свой собственный диалект SQL, который она внедрила в свои спецификации Архитектуры системных приложений (SAA). Интеграция диалекта SQL

от IBM в полный ряд продуктов IBM для баз данных была одной из целей спецификаций SAA. Хотя эта цель и не была никогда достигнута, SQL все еще играет важную роль в унификации продуктов IBM для баз данных.

FIPS — Национальный институт по стандартам и технологиям (National Institute of Standards and Technology, NIST) сделал SQL федеральным стандартом обработки информации (Federal Information Processing Standard, FIPS) начиная с 1987 г. Опубликованный в оригинале как "FIPS PUB 127", он определяет уровень на котором СУРБД должна соответствовать стандарту ANSI/ISO. С этих пор все системы управления реляционными базами данных, используемые правительством США, должны соответствовать текущим публикациям FIPS.

ODBC — В 1989 г. группа поставщиков баз данных образовала SQL Access Group для решения проблем взаимодействия. Хотя их первые усилия были в некоторой степени безуспешными, они занялись проблемой включения способа привязывания базы данных SQL к языку интерфейса пользователя. Результатом этих усилий стали спецификации интерфейса уровня вызовов (CLI), опубликованные в 1992 г. В том же году Microsoft опубликовала свои спецификации ODBC, основанные на стандарте CLI. С этого момента ODBC стал *de facto* средством доступа и совместного использования данных в БД, поддерживающих SQL.

Эти стандарты непрерывно развиваются по мере того, как ANSI/ISO принимают новые версии SQL, и иногда также развивались независимо.

Коммерческие реализации

Вначале SQL появился в среде для мэйнфреймов. Такие продукты, как DB/2, INGRES и Oracle, были популярны с 1979 г. и узаконили использование SQL как предпочтительного метода работы с реляционными базами данных. В течение 80-х годов реляционные базы данных и SQL распространились на среду персональных компьютеров, а такие продукты, как R:BASE, dBase IV и Super Base, перенесли мощь SQL на клавиатуру пользователя. В начале 90-х началась эпоха клиент-серверной вычислительной модели и были разработаны такие СУРБД, как Microsoft SQL Server и Informix-SE, которые предоставляли службы БД пользователям в многопользовательских средах различного типа. Сейчас, на заре нового тысячелетия, прилагаются совместные усилия по обеспечению доступности информации из баз данных через Интернет. Компании различных профилей ухватились за идею электронной

коммерции, и многие быстро продвигаются в направлении установления своего присутствия в Web. В результате разработчикам баз данных требуются более мощные базы данных клиент/сервер и более новые версии продуктов СУРБД для мэйнфрейма, которые они могли бы использовать для разработки и сохранения баз данных, необходимых для их Web-сайтов.

Что готовит будущее

Незадолго до опубликования SQL/92 комитет ANSI X3H2 уже приступил к основательному пересмотру стандарта. X3H2 отошел от своего традиционного стандарта присвоения имен и предложил новые спецификации SQL3. Одной из главных целей комитета в отношении новой версии была поддержка комбинации реляционной модели с объектной моделью. Эта комбинация иногда называется *расширенной реляционной моделью*, а иногда *объектно-реляционной моделью*. Комитет также работает над добавлением возможностей оперативной аналитической обработки (OLAP). Однако общее внимание комитета должно быть направлено на значительное расширение стандарта SQL/92 и дополнительную поддержку возможностей, существующих в большинстве объектно-реляционных сред баз данных.

Работа, выполненная до сих пор в отношении SQL3, по своему объему и качеству достойна всяческого уважения. Преследовались не только упомянутые ранее цели, но также стандарт был разделен на несколько частей. Некоторые из них были разделены или объединены в том или ином месте, и иногда требуется хорошая карточка с пометками для отслеживания того, что имеется или отсутствует в новом стандарте.

В таблице 3.1 приведено имя и описание каждой из частей SQL3, а также состояние каждой части на момент написания данной книги.

Таблица 3.1

Структура SQL3 на настоящий момент

Имя	Состояние	Описание
Часть 1: SQL/Каркас	Завершено в 1999	Описывает каждую часть стандарта и информацию, общую для всех частей.
Часть 2: SQL/Базис	Завершено в 1999	Определяет синтаксис и семантику частей определения и манипулирования данными в языке SQL.
SQL/OLAP (Оперативная аналитическая обработка)	В процессе	Описывает функции и операции, используемые для аналитической обработки (рассматривается как поправка к SQL/Базис).
Часть 3: SQL/CLI (Интерфейс уровня вызовов)	Завершено в 1999	Разработанная группой SQL Access, эта часть соответствует спецификациям ODBC от Microsoft.

Таблица 3.1 (*продолжение*)

Структура SQL3 на настоящий момент

Имя	Состояние	Описание
Часть 4: SQL/PSM (Постоянно хранимые модули)	Завершено в 1999	Определяет процедурный язык SQL-операторов, которые полезны в функциях и процедурах, определенных пользователем. (Поддержка хранимых процедур, хранимые функции, оператор вызова и вызов процедур были в итоге перенесены в SQL/Базис.)
Часть 5: SQL/Связывания	Завершено в 1999	Определяет, как SQL встраивается в языки программирования, не являющиеся объектно-ориентированными. В следующей версии SQL эта часть будет объединена с SQL/Базис.
Часть 6: Транзакции (спецификация XA)	Отсутствует	Специализация SQL в соответствии со спецификацией X-Open XA. Эта часть была изъята.
Часть 7: SQL/Данные времени	Задерживается	Определяет поддержку сохранения и извлечения данных, связанных со временем. Имеются различные мнения в отношении этих требований и подробностей временных данных, поэтому в последние несколько лет работа приостановилась.
Часть 8: SQL/Объекты — Расширенные объекты	Отсутствует	Определяет как абстрактные типы данных, определенные приложением, обрабатываются СУРБД. Эта часть была снова объединена с SQL/Базис, поэтому больше не существует.
Часть 9: SQL/MED Управление данными	В процессе	Определяет дополнительный синтаксис и определения для SQL/Базис, которые предоставляют SQL доступ к источникам не-SQL данных (т. е. к файлам).
Часть 10: SQL/OLB Привязки к объектному языку	Завершено в 1998 (стандарт ANSI)	Определяет синтаксис и семантику встроенного SQL в языке программирования Java. Это соответствует другому стандарту ANSI — SQLJ, часть 0.
Часть 11: SQL/Схемы	Отсутствует	Информационные схемы и схемы определения. В настоящее время это часть SQL/Базис, но будет выделено в отдельную часть в следующей версии SQL.

Таблица 3.1 (*продолжение*)

Структура SQL3 на настоящий момент

Имя	Состояние	Описание
Процедуры SQL, использующие язык программирования Java™	Завершено в 1999 (стандарт только ANSI, основанный на SQL/92)	Определяет, как программный код на Java может использоваться в базе данных SQL.
Часть 12: SQL/Тиражирование	Работа началась в 2000 г.	Определяет поддержку и возможности тиражирования базы данных SQL.

В 1997 г. организация X3 в составе ANSI была переименована в Национальный комитет по стандартам информационных технологий (NCITS). А технический комитет, ответственный за стандарт SQL3, теперь называется ANSI NCITS-H2. Учитывая огромное количество систем реляционных баз данных, существующих в мире сегодня, NCITS-H2 продолжит работу над SQL. Фактически работа над следующей версией уже начата.

Зачем изучать SQL

Изучение SQL дает навыки, необходимые для извлечения информации из любой реляционной базы данных. Оно также помогает понять механизмы, лежащие в основе графического интерфейса запросов, которые можно найти во многих продуктах СУРБД. Понимание SQL поможет проектировать сложные запросы и обеспечит знания, необходимые для диагностики запросов при возникновении проблем.

Поскольку SQL применяется в самых разных СУРБД, вы можете использовать свои навыки для разнообразных платформ. Например, если вы изучали SQL в Microsoft Access 2000, то можете воспользоваться своими знаниями, когда ваша компания примет решение перейти на Sybase SQL Server. Вам не потребуется изучать повторно SQL — просто нужно будет изучить разницу между диалектами Microsoft Access 2000 и Sybase SQL Server. То же самое относится и к переходу с Sybase SQL Server на IBM DB/2.

SQL — это всерьез и надолго. Многие поставщики инвестировали большие суммы денежных средств, времени и усилий на исследования по внедрению SQL в свои продукты СУРБД. Огромное количество предприятий и организаций построили большую часть своей инфраструктуры информационных технологий на этих продуктах. Как вы, вероятно, догадались, SQL продолжит свое развитие, чтобы удовлетворить изменяющийся спрос и требования рынка.

Итоги

Мы начали эту главу с обсуждения происхождения SQL. SQL является языком реляционных баз данных, который был создан вскоре после введения реляционной модели. На ранних этапах своего развития SQL был тесно связан с развитием самой реляционной модели.

Первые реляционные базы данных были реализованы на мэйнфреймах. IBM и Oracle стали главными игроками на рынке СУРБД.

Затем мы обсудили происхождение стандарта SQL. До того как ANSI установил официальный стандарт, существовал неофициальный стандарт, и комитет ANSI X3H2 провел огромную подготовительную работу над спецификациями.

Хотя новый стандарт включал множество возможностей, составлявших "наименьший общий знаменатель", он предоставил основу, на которой язык смог развиваться в дальнейшем. ISO опубликовала свой собственный стандарт, который точно соответствует спецификациям ANSI.

Различные люди и организации критиковали первоначальный стандарт. ANSI/ISO ответили на эту критику, приняв несколько редакций стандарта. Переходя от одной версии к другой, был достигнут стандарт SQL/92. Стандарт определяет различные уровни соответствия, которые позволяют поставщикам реализовать стандартные возможности в их продуктах как можно более гладко. Существуют и другие стандарты, включающие SQL в той или иной форме.

В конце мы кратко обсудили будущее SQL. Сейчас развивается SQL3, и это будет намного более сложный стандарт, чем SQL/92. Итак, SQL продолжает развиваться, и имеются убедительные причины для изучения этого языка.

Часть II

Основы SQL

4

Создание простых запросов

"Думай как мудрец, но говори языком простых людей"

— Уильям Батлер Йетс

Вопросы, рассматриваемые в данной главе:

- Знакомство с SELECT
- Оператор SELECT
- Краткое отступление: данные в сравнении с информацией
- Перевод запроса на SQL
- Исключение дублирующих строк
- Сортировка информации
- Сохранение работы
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

Теперь самое время перейти к сути дела и изучить сам язык. Нас будет интересовать та часть языка, которая связана с манипулированием данными. Поэтому мы полностью сосредоточим свое внимание на действительно рабочей лошадке SQL — операторе SELECT.

Знакомство с SQL

В отличие от всех других ключевых слов, SELECT действительно представляет собой сердце SQL. Он является краеугольным камнем самых мощных и сложных операторов языка и средством извлечения информации из таблиц базы данных. SELECT используется совместно с другими ключевыми словами и условиями для поиска и представления информации безграничным числом способов. На любой вопрос типа: кто, что, где, когда, что если и сколько — можно ответить, используя SELECT.

До тех пор, пока база данных спроектирована надлежащим образом и содержит соответствующие данные, можно получить необходимые ответы для принятия правильного решения.

Операцию SELECT в SQL можно разделить на три меньшие операции, которые мы будем называть оператор SELECT, выражение SELECT и запрос SELECT. Такой способ позволит легче понять и оценить сложность оператора. Для каждой из этих операций существует свой собственный набор ключевых слов и условий, обеспечивающий гибкость в создании окончательного оператора SQL, соответствующего запросу, который вы хотите сформулировать для базы данных. Можно даже объединять операции различным образом, чтобы ответить на очень сложные вопросы.

В данной главе начинается обсуждение оператора SELECT и запроса SELECT (подробнее см. в главе 6).

> **Внимание!** Обычно в книгах и статьях по реляционным базам данных можно увидеть термины “таблица”, “запись” и “поле”, которые используются как взаимозаменяемые с терминами “таблица”, “строка” и “столбец”. Однако стандарт SQL определяет именно термины “таблица”, “строка” и “столбец” для указания этих конкретных элементов структуры базы данных. Сохраним соответствие стандарту SQL и будем использовать эти термины в остальной части книги.

Оператор SELECT

Оператор SELECT образует основу каждого вопроса, который вы задаете базе данных. Когда создается и выполняется оператор SELECT, то вы “обращаетесь с запросом к базе данных” (надеемся, что все читатели единодушны в этом вопросе). Фактически многие программы СУРБД позволяют сохранить оператор SELECT как *запрос*, *представление* или *хранимую процедуру*. Когда кто-то собирается обратиться с запросом к базе данных, вы должны понимать, что предполагается выполнить некий оператор SQL. В зависимости от программы СУРБД операторы SELECT могут выполняться по разному: непосредственно из окна командной строки, из таблицы интерактивного запроса с использованием примера (Query By Example, QBE) или из блока программного кода. Независимо от того, каким образом вы решили определить и выполнить его, синтаксис оператора SELECT всегда один и тот же.

Основные условия оператора SELECT

Оператор SELECT состоит из нескольких отдельных ключевых слов, называемых *условиями*. Он определяется с помощью различных конфигураций этих условий для извлечения требуемой информации. Некоторые из этих условий являются обязательными, другие — нет. К тому же каждое условие имеет одно или несколько ключевых слов, которые представляют обязательное или необязательное значение.

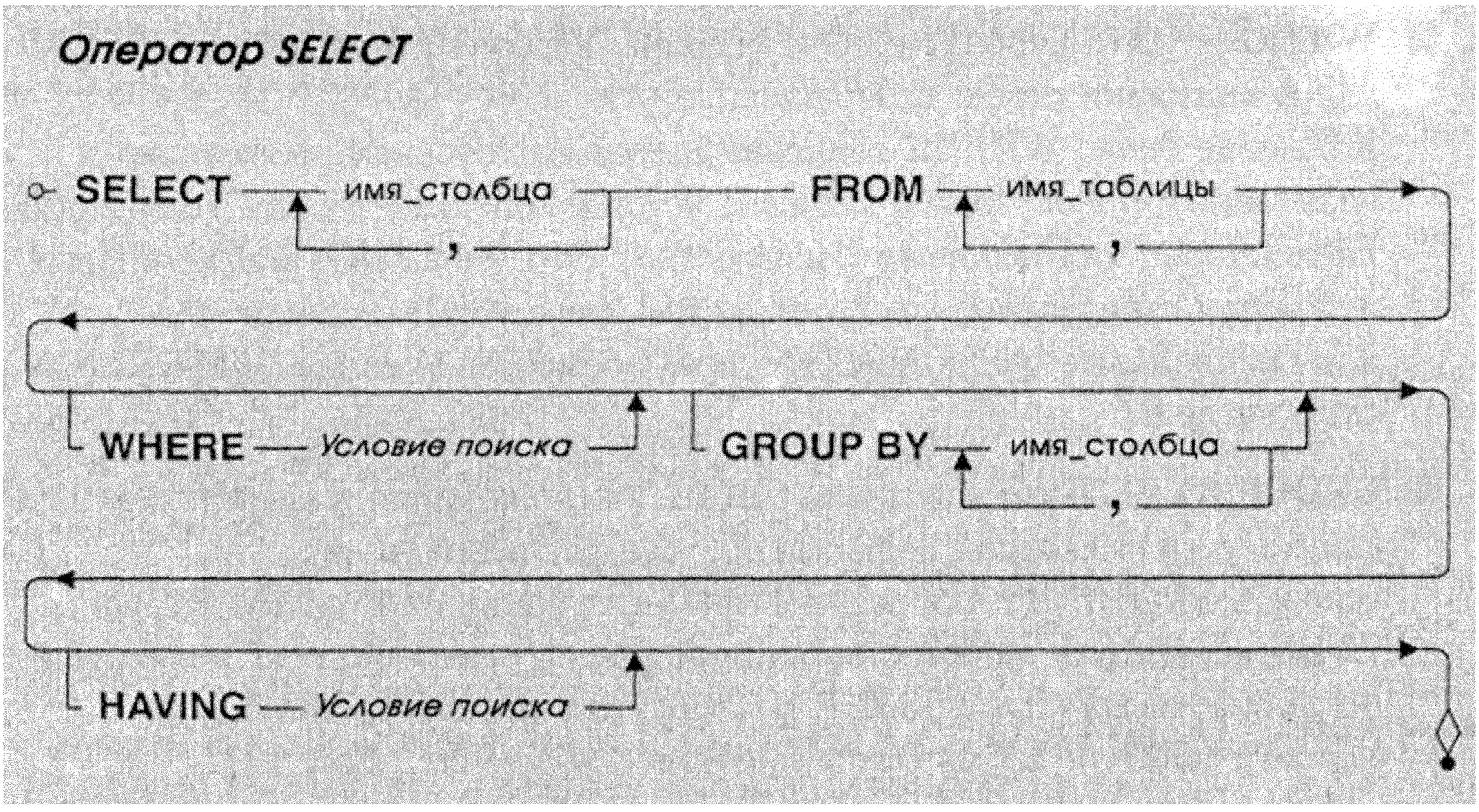


Рис. 4.1. *Диаграмма оператора SELECT*

Эти значения используются условием, чтобы помочь извлечь информацию, запрашиваемую оператором SQL в целом. На рис. 4.1 представлена диаграмма оператора SELECT и его условий.

> **Внимание!** Синтаксическая диаграмма на рис. 4.1 отражает упрощенный оператор SELECT. Мы продолжим корректировку и модификацию этой диаграммы по мере знакомства и работы с новыми ключевыми словами и условиями.

Ниже приведено краткое изложение условий оператора SELECT:

- **SELECT** — Это основное условие оператора SELECT, и его наличие абсолютно обязательно. Оно используется для определения столбцов, которые вы хотите получить в наборе результата для своего запроса. Сами столбцы извлекаются из таблицы или представления, которые определены в условии FROM. Их также можно извлечь из нескольких таблиц одновременно (см. часть III). В условии FROM можно использовать такие обобщенные функции, как “Quantity × Price” (Количество × Цена).
- **FROM** — Это второе наиболее важное условие в операторе SELECT, и оно также является обязательным. Оно используется для определения таблиц, из которых должны извлекаться столбцы, перечисленные в условии SELECT. Это условие можно использовать и для более сложных способов.

- **WHERE** — Это необязательное условие, которое используется для фильтрации строк, возвращенных условием FROM. Ключевое слово WHERE сопровождается выражением, формально называемым *предикатом*, значение которого оценивается как True (Истина), False (Ложь) или Unknown (Неизвестно). Это выражение можно проверить, используя стандартные операторы сравнения, булевы операторы или специальные операторы (обо всех элементах условия WHERE см. в главе 6).

- **GROUP BY** — Когда в условии SELECT используются агрегатные функции для получения сводной информации, следует использовать условие GROUP BY для разделения информации на отдельные группы. СУБД использует любой столбец или список столбцов, расположенный после ключевых слов GROUP BY, как группирующие столбцы. Условие GROUP BY является необязательным (см. главу 13).

- **HAVING** — Условие HAVING специально связано с условием GROUP BY и используется для фильтрации сгруппированной информации. Оно подобно условию WHERE, в котором за ключевым словом HAVING следует выражение, оцениваемое как True, False или Unknown. Это выражение можно проверить, используя стандартные операторы сравнения, булевы операторы или специальные операторы. Условие HAVING также необязательно (подробнее см. в главе 14).

Вначале мы должны усвоить самые базовые конструкции оператора SELECT, поэтому сосредоточимся на условиях SELECT и FROM. Другие условия будем добавлять по одному, переходя к следующим главам и формируя более сложный оператор SELECT.

Краткое отступление: Данные в сравнении с информацией

Прежде чем обратиться с первым запросом к базе данных, необходимо четко уяснить один момент. Между данными и информацией имеется глубокое различие. По существу, данные — это то, что хранится в базе данных, а информация — это то, что извлекается из нее. Это различие важно понимать, потому что оно помогает видеть надлежащее соотношение вещей. Вспомните, что база данных проектируется для того, чтобы обеспечить осмысленную информацию для сотрудников организации. Однако информацию можно предоставить, только если соответствующие данные существуют в базе данных и если структура самой базы данных поддерживает эту информацию. Рассмотрим эти термины более конкретно.

Значения, сохраняемые в базе данных, являются данными. Данные являются статичными в том смысле, что они остаются в том же состоянии до тех пор, пока вы

не измените их вручную или с использованием автоматизированного процесса. На рис. 4.2 представлены примеры некоторых данных.

```
Katherine Ehrlich 89931 Active 79915
```

Рис. 4.2. *Пример данных*

При поверхностном взгляде эти данные не имеют смысла. Например, совсем непросто определить, что означает 89931. Это почтовый индекс? Номер какой-то части? Даже если известно, что оно представляет собой идентификационный номер клиента, можно ли его связать с Катериной Эрлих? Возможность узнать это отсутствует полностью до того момента, как данные будут обработаны. Как только данные обработаны и стали понятными и полезными для работы, они становятся информацией. Информация является динамической в том смысле, что она постоянно изменяется относительно данных, сохраненных в базе данных, а также вследствие ее способности к бесконечному числу способов обработки и представления. Можно показать информацию как результат оператора SELECT, отобразить ее на некотором бланке на экране компьютера или вывести на бумагу как отчет. Но данные необходимо обработать таким образом, который позволяет превратить их в осмысленную информацию.

На рис. 4.3 представлены данные из предыдущего примера, преобразованные в информацию на экране клиента. Здесь показано, как можно манипулировать данными для того, чтобы придать им смысл.

При работе с оператором SELECT мы использовали его условия для манипулирования *данными*, но сам оператор возвратил *информацию*. Она будет выглядеть так:

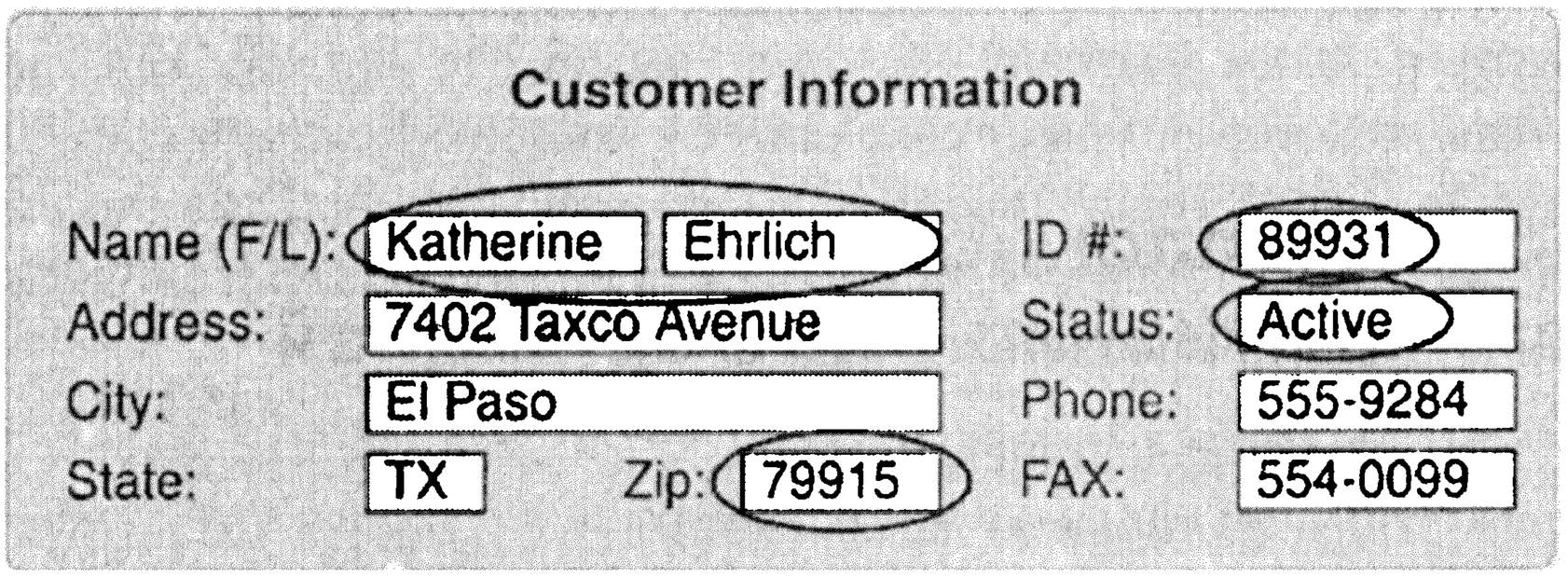


Рис. 4.3. *Пример данных, преобразованных в информацию*

Нам необходимо решить последний вопрос. Когда выполняется оператор SELECT, он обычно извлекает одну или несколько строк информации. Точное количество зависит от того, как построен оператор. Все эти строки вместе называются *набором результатов*, и этот термин используется во всей оставшейся части книги. В этом наименовании заключен точный смысл, поскольку при использовании реляционной базы данных вы всегда работаете с наборами данных. (Вспомните, что реляционная

модель основана, в частности, на теории множеств.) Можно легко просмотреть информацию в наборе результатов, и во многих случаях его данные можно модифицировать. Но все это зависит от построения оператора SELECT, к изучению которого мы и приступим.

Перевод запроса на SQL

Запрос информации из БД обычно имеет вид вопроса или утверждения, которое подразумевает вопрос. Например, можно сформулировать такие утверждения:

"В каких городах живут ваши клиенты?"

"Показать текущий список наших служащих и номера их телефонов".

"Предметы какого типа мы предлагаем в настоящее время?"

"Дать имена людей, состоящих в нашем штате, и даты их приема на работу".

Когда известно, что вы хотите спросить, можно преобразовать запрос в более формальное утверждение. Преобразование выполняется с использованием следующей формы:

Select <item> from the <source>
(Выбрать <элемент> из <источник>)

Начните с рассмотрения своего запроса и замените все слова или фразы типа "список", "показать", "какой", "который" и "кого/что" на слово Select (Выбрать). Затем найдите в запросе все имена существительные и определите, представляет ли данное существительное элемент, который вы хотите видеть, или оно является именем таблицы, в которой может сохраняться элемент. Если это элемент, вставьте его вместо компонента <item> преобразуемого утверждения. Если это имя таблицы, вставьте его вместо компонента <source>. Если преобразовать наш первый вопрос, то утверждение будет примерно следующим:

Select city from the customers table
(Выбрать город из таблицы заказчиков)

После того как утверждение определено, необходимо превратить его в законченный оператор SELECT, используя синтаксис, показанный на рис. 4.4. Однако первый шаг состоит в приведении в порядок преобразуемого утверждения. Это достигается путем вычеркивания всех слов, которые не являются существительными, представляющими собой имя столбца или таблицы, и не являются ключевыми словами SQL. Взгляните, какой вид принимает преобразуемое утверждение в процессе окончательного преобразования:

Select city from ~~the~~ customers ~~table~~

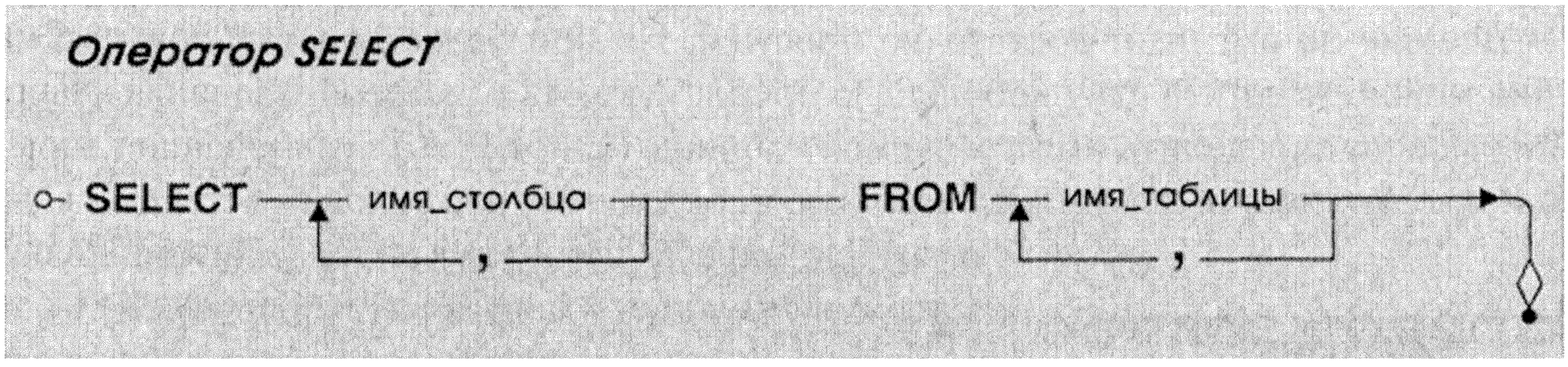


Рис. 4.4. *Синтаксис простого оператора SELECT*

Удалите вычеркнутые слова, и вы получите завершенный оператор SELECT:

```
SELECT City FROM Customers
```

Эту методику из трех этапов можно использовать к любому запросу, с которым обращаются к базе данных. Фактически эта методика применяется в остальной части книги и рекомендуется для использования при построении таких операторов. Однако в конечном счете эти этапы можно объединить в одну неразрывную операцию, и вы сможете это сделать, когда освоитесь с написанием операторов SELECT.

Начиная изучать использование SQL, вы будете работать в основном со столбцами и таблицами. Синтаксическая диаграмма на рис. 4.4 отражает этот факт, используя имя_столбца в условии SELECT и имя_таблицы в условии FROM. В следующей главе мы покажем, как использовать другие элементы в этих условиях для создания более сложных операторов SELECT.

Запрос, использованный в предыдущем примере, является относительно простым. Нам легко далось как его переопределение в переводимое утверждение, так и идентификация имен столбцов, которые были представлены в операторе. Но что если запрос не является простым и легким для преобразования и если сложно идентифицировать столбцы, необходимые для условия SELECT. Самый легкий способ — это уточнить свой запрос и сделать его более конкретным. Например, запрос типа *"Представить информацию о наших клиентах"* можно видоизменить и конкретизировать: *"Привести в списке имя, город и телефонный номер каждого нашего клиента"*. Если уточнение запроса не решает проблемы, есть еще два варианта действий. При первом варианте необходимо установить, содержит ли таблица, определенная в условии FROM оператора SELECT, все имена столбцов, которые могут помочь прояснить запрос и облегчить определение преобразуемого утверждения. Вторым вариантом является более тщательная проверка запроса на предмет того, *подразумевает ли* слово или фраза, которые в нем содержатся, какие-либо имена столбцов. Можно ли воспользоваться любым или обоими вариантами — зависит от самого запроса. Просто помните, что имеются доступные методы, когда окажется затруднительным определить преобразуемое утверждение. Рассмотрим на примере каждый метод и его применение в типичном сценарии.

Чтобы проиллюстрировать первый метод, предположим, что вы пытаетесь преобразовать следующий запрос:

"I need the names and address of all our employees".
("Мне нужны имена и адреса всех наших сотрудников".)

Внешне он выглядит достаточно простым. Но при более пристальном рассмотрении можно заметить одну небольшую проблему. Хотя для преобразования утверждения можно определить имя требуемой таблицы (Employees), в запросе нет ничего, что могло бы помочь определить столбцы, необходимые для условия SELECT. Слова "имена" и "адреса" имеются в запросе, но они являются терминами общего характера. Эту проблему можно разрешить, повторно просмотрев таблицу, идентифицированную в запросе, и определив, содержит ли она какие-либо столбцы, которыми можно было бы заменить эти термины. Если да — используйте имена столбцов в преобразуемом утверждении. (Можно использовать обобщенные версий имен столбцов, если это поможет более ясно представить утверждение. Однако в синтаксисе SQL требуется использовать реальные имена столбцов.) В данном случае поищите в таблице Employees имена столбцов, которые могут использоваться вместо слов "имена" и "адреса". Используйте таблицу Employees, показанную на рис. 4.5, и определите, можно ли использовать какой-либо из ее столбцов.

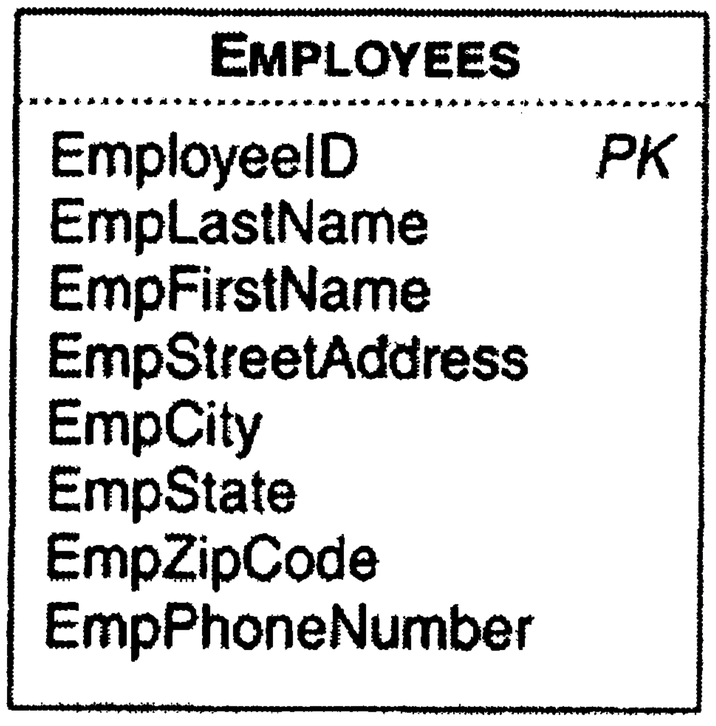


Рис. 4.5. *Структура таблицы Employees*

Несомненно, из этой таблицы будут использоваться шесть столбцов. EmpFirstName и EmpLastName оба заменяют слово "имена" в запросе, тогда как EmpStreetAddress, EmpCity, EmpState и EmpZipCode заменяют "адреса". Теперь примените весь процесс преобразования к запросу, который для удобства приводится повторно. (Будем использовать общие формы имен столбцов для преобразуемого утверждения и реальные имена столбцов в синтаксисе SQL.)

"I need the names and addresses of all our employees".
("Мне нужны имена и адреса всех наших сотрудников".)

Преобразование: Select first name, last name, street address, city, state, and zip code from the employees table
(Выбрать имя, фамилию, улицу, город, штат и почтовый код из таблицы Employees)

Уточнение: Select first name, last name, street address, city, state, ~~and~~ zip code from ~~the~~ employees ~~table~~
(Выбрать имя, фамилию, улицу, город, штат, почтовый код из Employees)

SQL
```
SELECT EmpFirstName, EmpLastName, EmpStreet,
    EmpCity, EmpState, EmpZipCode
FROM Employees.
```

Внимание! В этом примере явно показано, как использовать составные столбцы в условии SELECT (подробнее см. ниже).

Следующий пример иллюстрирует второй метод, который включает поиск подразумеваемых столбцов в запросе. Предположим, вы пытаетесь преобразовать следующий запрос:

"What kind of classes do we currently offer?"
("Какие виды учебных курсов мы проводим в настоящее время?")

На первый взгляд, определение преобразуемого утверждения из этого запроса может показаться затруднительным. В запросе отсутствуют имена столбцов, а при отсутствии хотя бы одного элемента для выбора невозможно создать полное преобразуемое утверждение. Что же теперь делать? Следует более внимательно рассмотреть каждое слова в запросе и попытаться определить, имеется ли какое-либо имя, которое *подразумевает* имя столбца из таблицы Classes (Учебные курсы). Можно ли найти такое слово?

В данном случае слово "виды" может подразумевать имя столбца в таблице Classes. Почему? Потому что тип предметов также может рассматриваться как категория класса. Если в таблице Classes имеется столбец такой категории, то у вас имеется имя столбца, которое необходимо для преобразования утверждения, и, как предполагается, оператор SELECT. Предположим, что в таблице Classes имеется столбец категории, и снова проведем процесс преобразования в три этапа.

"Какие виды учебных курсов мы проводим в настоящее время?"

Преобразование:	Select category from the classes table (Выбрать категории из таблицы Classes)
Уточнение:	Select category from ~~the~~ classes ~~table~~ (Выбрать категории из "Classes")
SQL	`SELECT Category` `FROM Classes`

Как показано в примере, этот метод включает использование синонимов для замены определенных слов или фраз запроса. Если установлено слово или фраза, под которыми может подразумеваться имя столбца, попытайтесь заменить его на синоним, который может действительно идентифицировать столбец, существующий в базе данных. Если первый пришедший на ум синоним не подходит, попробуйте другой и продолжайте искать до тех пор, пока не найдется синоним, который идентифицирует имя столбца, либо пока вы не решите, что ни исходное слово, ни какой-либо его синоним не представляют имя столбца.

Внимание! Все имена столбцов и таблиц, используемые в примерах по синтаксису SQL, взяты из учебных баз данных в приложении B (если не указано иное). Это соглашение применяется ко всем примерам в остальной части книги.

Расширение области рассмотрения

В операторе SELECT несколько столбцов можно извлечь так же просто, как и один столбец. Перечислите имена столбцов, которые вы хотите использовать в условии SELECT, и отделите каждое имя в этом списке запятой. В синтаксической диаграмме, показанной на рис. 4.6, опция использования более одного столбца указывается линией, расположенной под имя_столбца. Запятая в середине линии разделяет имена столбцов, которые вы хотите использовать в условии SELECT.

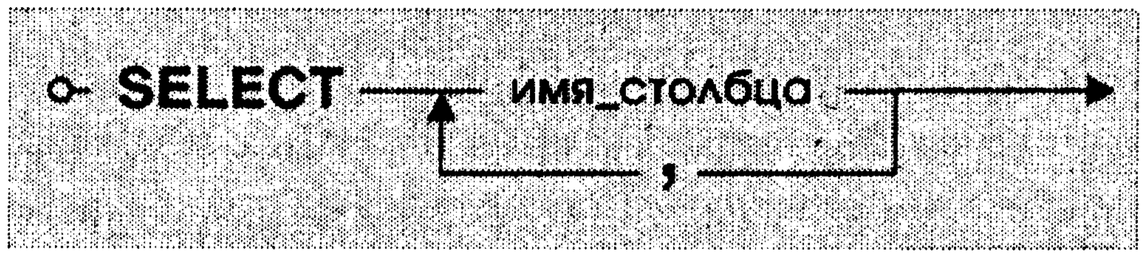


Рис. 4.6. *Отображение нескольких столбцов в условии SELECT*

Опция использования нескольких столбцов в операторе SELECT предоставляет средства ответов на вопросы, подобные следующему:

"Show me a current list of our employees and their phone number".
("Показать текущий список наших сотрудников и номера их телефонов".)

Преобразование: Select the last name, first name and phone number of all our employees from the employees table
(Выбрать фамилию, имя и номер телефона всех наших служащих из таблицы служащих)

Уточнение: Select ~~the~~ last name, first name ~~and~~ phone number ~~of all our employees~~ from ~~the~~ employees table
(Выбрать имя, фамилия, номер телефона из "Служащие")

SQL
```
SELECT EmpLastName, EmpFirstName, EmpPhoneNumber
FROM Employees
```

"What are the names and prices of the products we carry, and under what cathegory is each item listed?"
("Как называются и сколько стоят перевозимые нами продукты, и с какой категорией указывается каждая его позиция в перечне?")

Преобразование: Select the name, price, and category of every product from the product table
(Выбрать наименование, цену и категорию каждого продукта из таблицы продуктов)

Уточнение: Select ~~the~~ name, price, ~~and~~ category ~~of every product~~ from ~~the~~ products ~~table~~
(Выбрать наименование, цену, категорию из "Продукты")

SQL
```
SELECT ProductName, Retail Price, Category
FROM Products
```

Вы добьетесь преимуществ, рассматривая широкий спектр информации при работе с несколькими столбцами в операторе SELECT. Кстати, последовательность столбцов в условии SELECT не важна — они могут быть перечислены в любом порядке. Это обеспечивает гибкость при просмотре одной и той же информации множеством способов.

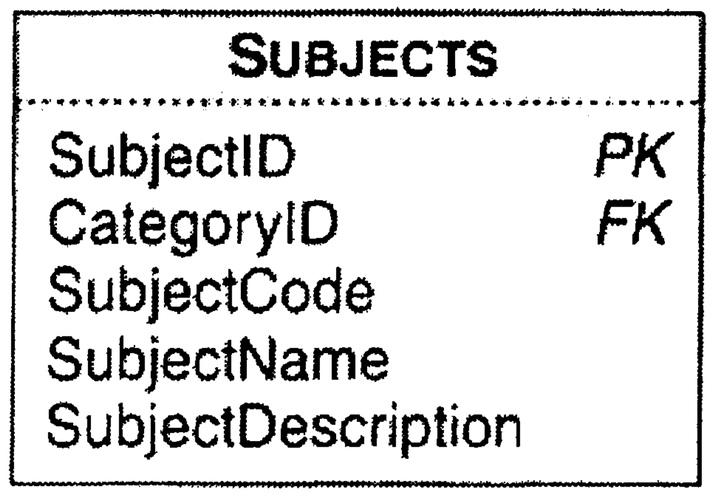


Рис. 4.7. *Структура таблицы Subjects*

Предположим, вы работаете с таблицей, представленной на рис. 4.7, и хотите обратиться к базе данных со следующим запросом:

> *"Show me a list of subjects, the categoty each belong to, and the code we use in our catalog. But I'd like to see the name first, followed by the category and then the code".*
> *("Показать список тем, категорию, к которой каждая из них принадлежит, и код, используемый в нашем каталоге. Но мне хотелось бы видеть вначале наименование, потом категорию, а затем код".)*

Вы все еще можете преобразовать этот запрос в соответствующий оператор SELECT, даже если хотите видеть столбцы в определенном порядке. Просто перечислите имена столбцов в указанном порядке, когда определяете преобразуемое утверждение. Далее показан процесс преобразования этого запроса в оператор SELECT:

Преобразование: Select the subject name, category ID, and subject code from the subjects table
(Выбрать название темы, идентификатор категории и код темы из таблицы "Темы")

Уточнение: Select ~~the~~ subject name, category ID, ~~and~~ subject code from ~~the~~ subjects ~~table~~
(Выбрать название темы, идентификатор категории, код темы из "Темы")

Количество столбцов, которые можно определить в условии SELECT, не ограничено. Фактически можно перечислить все столбцы из таблицы источника. В следующем примере приведен оператор SELECT, используемый для определения всех столбцов из таблицы Subjects на рис. 4.7:

```
SQL                SELECT SubjectID, CategoryID, SubjectCode,
                       SubjectName, SubjectDescription
                   FROM Subjects
```

Когда определяются все столбцы из таблицы источника, а в ней имеется множество столбцов, придется изрядно поупражнять пальцы на клавиатуре! К счастью,

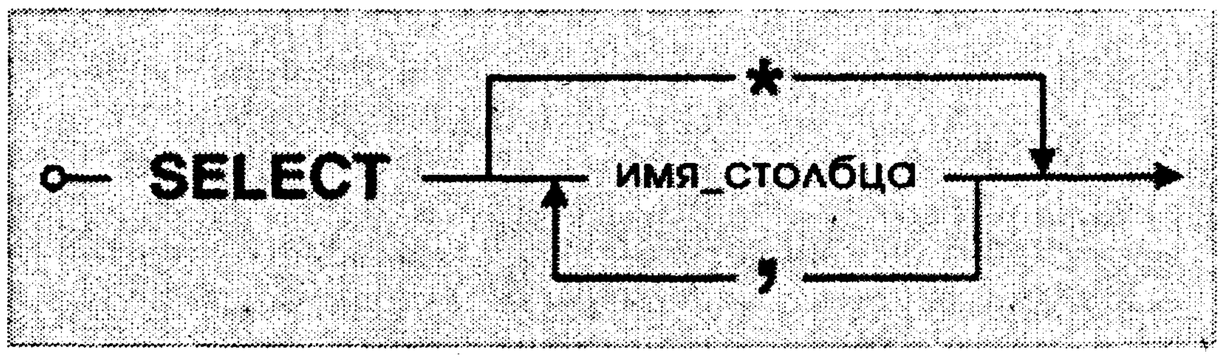


Рис. 4.8. *Синтаксическая диаграмма, показывающая использование звездочки для сокращения*

стандарт SQL разрешает применить звездочку как сокращенное обозначение для существенного укорачивания оператора. Синтаксическая диаграмма на рис. 4.8 показывает использование звездочки как альтернативы списку столбцов в условии SELECT.

Поместите звездочку сразу после условия SELECT, когда нужно определить все столбцы из исходной таблицы в условии FROM. Например, так выглядит предшествующий оператор SELECT при использовании сокращения:

```
SQL                    SELECT *
                       FROM Subjects
```

При использовании этого оператора печатать нужно намного меньше! Однако при использовании данного способа возникает один вопрос. Звездочка представляет все столбцы, существующие в *текущий момент* в исходной таблице, а добавление или удаление столбцов оказывает влияние на то, что вы видите в наборе результата оператора SELECT. (Как ни странно, стандарт SQL утверждает, что добавление или удаление столбцов *не должно оказывать* влияния на набор результатов.) Этот вопрос важен только тогда, когда необходимо постоянно видеть те же столбцы в наборе результатов. Когда в условии SELECT используется звездочка, СУБД не предупредит об удалении столбцов из исходной таблицы, но выдаст предупреждение, если не сможет найти *явно* указанный столбец. Хотя это не вызывает реальных проблем, это будет важно при углублении в мир программирования на SQL. Наше эмпирическое правило состоит в следующем: используйте звездочку только тогда, когда необходимо создать "сляпанный наспех" запрос, чтобы увидеть всю информацию в указанной таблице. В противном случае определите все столбцы, необходимые для запроса. В конечном итоге запрос возвратит точно ту информацию, которая требуется, и будет в большей степени самодокументированным.

Рассмотренные до сих пор примеры основывались на простых запросах, для которых требуются столбцы только из одной таблицы. Работа с более сложными запросами, для которых требуются столбцы из нескольких таблиц, изучается в части III.

Исключение дубликатов строк

При работе с операторами SELECT вы неизбежно столкнетесь в наборах результатов с повторяющимися строками. Поэтому используйте ключевое слово DISTINCT в своем операторе SELECT, и набор результатов не будет содержать повторяющихся строк. На рис. 4.9 показана синтаксическая диаграмма для ключевого слова DISTINCT.

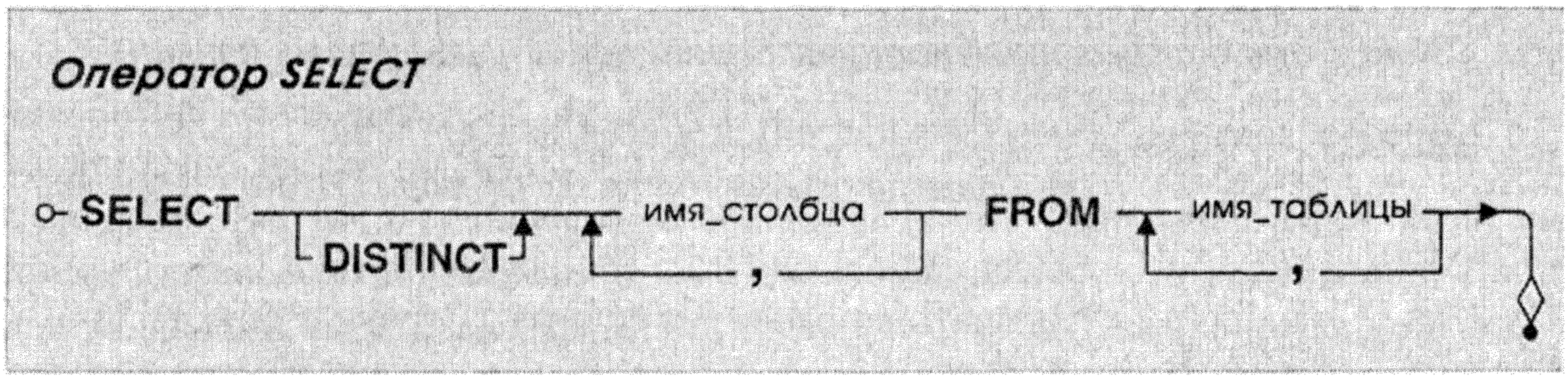


Рис. 4.9. *Синтаксис ключевого слова DISTINCT*

Как показано на диаграмме, DISTINCT является необязательным ключевым словом, которое предшествует списку столбцов, указанных в условии SELECT. Ключевое слово DISTINCT требует от СУБД оценить значения всех столбцов каждой строки как один элемент, строка за строкой, и исключить все обнаруженные повторения. Оставшиеся уникальные строки возвращаются в наборе результатов. В приведенном ниже примере представлено, какое различие может создать ключевое слово DISTINCT при соответствующих обстоятельствах.

Предположим, что вы обращаетесь к базе данных со следующим запросом:

> *"Which cities are represented by our bowling league membership?"*
> *("Какие города представляют участники нашей лиги игры в боулинг?")*

Вопрос кажется достаточно простым, поэтому перейдем к процессу преобразования.

Преобразование: Select city from the bowlers table
(Выбрать города из таблицы игроков в боулинг)

Уточнение: Select city from ~~the~~ bowlers ~~table~~
(Выбрать город из "Игроки в боулинг")

SQL
```
SELECT City
FROM Bowlers
```

Здесь проблема состоит в том, что набор результатов для этого оператора SELECT показывает *каждое появление* каждого названия города, обнаруженного в таблице Bowlers (Игроки в боулинг). Например, если имеется 20 человек из Беллевью, 7 человек из Кента и 14 человек из Сиэтла, то в наборе результатов будет 20 раз упомянут Беллевью, 7 раз — Кент и 14 — Сиэтл. Очевидно, что эта избыточная информация не нужна. Хотелось бы видеть по *одному* названию каждого города, которые были обнаружены в таблице Bowlers. Эта проблема решается посредством использования ключевого слова DISTINCT в операторе SELECT для исключения избыточной информации.

Еще раз выполним перевод с использованием ключевого слова DISTINCT. Обратите внимание, что мы теперь включаем слово distinct ("по одному") как на этапе преобразования, так и на этапе уточнения.

"Какие города представляют участники нашей лиги игры в боулинг?"

Преобразование: Select distinct city from the bowlers table
(Выбрать по одному города из таблицы "Игроки в боулинг")

Уточнение: Select distinct city from ~~the~~ bowlers ~~table~~
(Выбрать по одному города из "Игроки в боулинг")

SQL
```
SELECT DISTINCT City
FROM Bowlers
```

Набор результатов для этого оператора SELECT отображает именно то, что вы ищете,— единственное упоминание каждого уникального города в таблице.

Ключевое слово DISTINCT также можно использовать для составных столбцов. Изменим предыдущий пример, запросив из таблицы Bowlers как штат, так и город. Наш новый оператор SELECT будет выглядеть так:

```
SELECT DISTINCT State, City FROM Bowlers
```

Этот оператор SELECT возвращает набор результатов, который содержит уникальные записи и показывает определенные отличия между городами с одинаковым названием. Например, показывается различие между "Portland, ME" и "Portland, OR" (Портленд, штат Мэн и Портленд, штат Орегон), "Hollywood, CA" и "Hollywood, FL" (Голливуд, штат Калифорния, и Голливуд, штат Флорида).

Ключевое слово DISTINCT — это очень полезный инструмент в определенных обстоятельствах. Используйте его, когда вы действительно хотите видеть уникальные строки в наборе результатов.

Сортировка информации

Операцию SELECT можно разделить на три более мелкие операции: оператор SELECT, выражение SELECT и запрос SELECT. Можно объединять эти операции различными способами, чтобы ответить на сложные запросы. Также необходимо объединять эти операции для сортировки строк в наборе результатов.

По определению строки набора результатов, возвращенные оператором SELECT, не упорядочены. Последовательность, в которой они появляются, основывается на

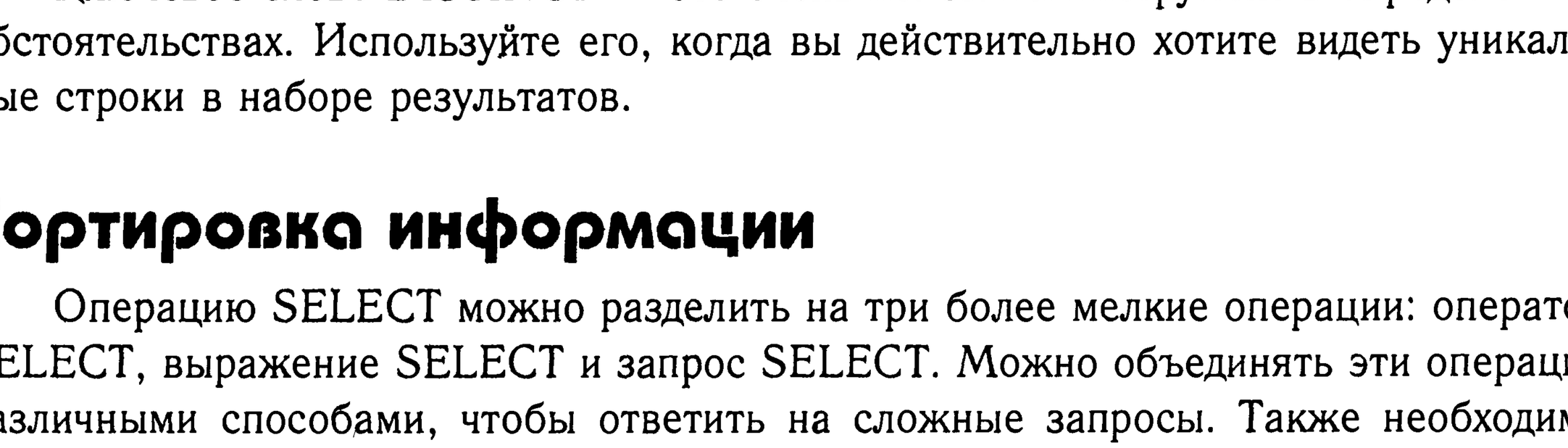


Рис. 4.10. *Синтаксическая диаграмма для запроса SELECT*

их физическом расположении в таблице. Единственный способ сортировки набора результатов заключается во встраивании оператора SELECT в запрос SELECT, как показано на рис. 4.10. Запрос SELECT определяется как оператор SELECT с условием ORDER BY. Именно условие ORDER BY запроса SELECT позволяет определить последовательность строк в окончательном наборе результатов. Из последующих глав вы узнаете, что для ответа на очень сложные вопросы можно вложить оператор SELECT в другой оператор SELECT или в выражение SELECT. Однако запрос SELECT невозможно вложить в какой-либо другой уровень.

Внимание! Во всей данной книге используются термины, которые можно найти в стандарте SQL от ANSI или которые являются общеупотребительными в большинстве систем базы данных. Однако стандарт ANSI SQL определяет условие ORDER BY только как часть *курсора* — объекта, который определяется внутри прикладной программы (полное обсуждение курсоров выходит за рамки данной книги). Поскольку многие реализации SQL позволяют в конце оператора SELECT включать условие ORDER BY, нами придуман термин *"запрос SELECT"* для описания такого типа оператора.
Это также позволяет обсудить концепцию сортировки окончательного вывода запроса для оперативного отображения или для использования в отчете.

Условие ORDER BY позволяет упорядочить набор результатов указанного оператора SELECT по одному или нескольким столбцам, а также содержит опцию упорядочивания по возрастанию или убыванию для каждого столбца. В условии ORDER BY можно использовать только те столбцы, которые в настоящий момент перечислены в условии SELECT. Хотя это требование указано в стандарте SQL, некоторые реализации его полностью игнорируют (однако во всех примерах, используемых в данной книге, это требование нами соблюдается). Когда в условии ORDER BY используются два или более столбцов, каждый столбец отделяется запятой. Как только сортировка завершается, запрос SELECT возвращает окончательный набор.

Внимание! Условие ORDER BY *не оказывает* влияния на физический порядок строк в таблице. Если требуется изменить порядок строк, обратитесь к документации по программному обеспечению для базы данных.

Важные дела в первую очередь: Схемы сортировки

Способ сортировки (или упорядочения) информации, определяемый условием ORDER BY, зависит от схемы сортировки, используемой в конкретной СУБД. Эта схема определяет порядок предшествования для каждого символа, перечисленного в текущем наборе символов языка, определенном операционной системой. Например, он устанавливает, должны ли буквы нижнего регистра располагаться при сортировке перед символами верхнего регистра или регистр не имеет значения. Проверьте

документацию по программному обеспечению базы данных и проконсультируйтесь с администратором БД, чтобы определить схему сортировки по умолчанию для своей базы данных.

А теперь вернемся к порядку

При наличии условия ORDER BY можно придать информации, извлекаемой из таблицы, больший смысл. Это относится как к простым запросам, так и к сложным. Можно перефразировать свои запросы так, чтобы они указывали требования по сортировке. Например, вопрос *"Категории каких курсов лекций предлагаются в настоящий момент?"* можно переделать в *"Перечислить категории предлагаемых нами курсов лекций и представить их в алфавитном порядке"*.

Прежде чем начать работу с запросом SELECT, необходимо скорректировать способ определения преобразуемого утверждения. Это включает добавление новой части в конец преобразуемого утверждения для учета новых требований по сортировке, определенных в запросе.

> Select <item> from the <source> **and order by <column(s)>**
> (Выбрать <элемент> из <источник> **и упорядочить по <столбец(ы)>**)

Теперь, когда запрос будет включать фразы типа "отсортировать результаты по названию города", "показать их в порядке по годам" или "привести список по именам и фамилиям", изучайте запрос более тщательно, чтобы определить, какой столбец или столбцы необходимо использовать в целях сортировки. Это простое упражнение, поскольку большинство людей использует такие типы фраз, а столбцы, которые нужны для сортировки, обычно самоочевидны. Как только соответствующий столбец или столбцы установлены, используйте их в качестве замены для <столбец(ы)> в преобразуемом утверждении. Рассмотрим простой запрос, чтобы увидеть, как это работает:

> *"List the categories of classes we offer and show them in alphabetical order".*
> *("Составить список категорий предлагаемых курсов лекций и показать их в алфавитном порядке".)*

Преобразование: Select category from the classes table and order by category
(Выбрать категории из таблицы "Курсы лекций" и упорядочить по категории)

Уточнение: Select category from ~~the~~ classes ~~table and~~ order by category
(Выбрать категорию из "Курсы лекций", упорядочить по категории)

SQL

```
SELECT Category
FROM Classes
ORDER BY Category
```

В данном примере можно предположить, что категория будет использоваться для сортировки, поскольку это единственный столбец, указанный в запросе. Также можно предположить, что сортировка должна производиться в порядке возрастания, потому что в запросе не указано противоположное. Это безопасное предположение. В соответствие со стандартом SQL, если не определен порядок сортировки, то автоматически предполагается сортировка в порядке возрастания. Однако, если вы хотите все указать явно, вставьте ASC (от Ascending — по возрастанию) после Category в условии ORDER BY.

В следующем запросе столбец, необходимый для сортировки, определяется более явно:

> *"Show me a list of vendor names in zip code order".*
> *("Показать список имен поставщиков, расположив их в порядке почтовых индексов".)*

Преобразование: Select vendor name and zip code from the vendors table and order by zip code
(Выбрать имена поставщиков и почтовые коды из таблицы поставщиков и упорядочить их по почтовому индексу)

Уточнение: Select vendor name ~~and~~ zip code from ~~the~~ vendors ~~table and~~ order by zip code
(Выбрать имя поставщика, почтовый индекс из "Поставщики", упорядочить по почтовому индексу)

SQL
```
SELECT VendName, VendZipCode
FROM Vendors
ORDER BY VendZipCode
```

Если нужно отобразить набор результатов в обратном порядке, вставьте ключевое слово DESC (от Descending — по убыванию) после соответствующего столбца в условии ORDER BY. Посмотрите, как изменяется оператор SELECT из предыдущего примера, когда требуется представить информацию, отсортированную по почтовому индексу в порядке убывания:

SQL
```
SELECT VendName, VendZipCode
FROM Vendors
ORDER BY VendZipCode DESC
```

В следующем примере представлен более сложный запрос, который требует сортировки нескольких столбцов. Единственным отличием между этим примером и предыдущим является то, что в нем используется больше столбцов в условии ORDER BY. Обратите внимание, что столбцы разделены запятыми в соответствии с синтаксической диаграммой, показанной ранее на рис. 4.10.

"Display the names of our employees, including their phone number and ID number, and list them by last names and first name".
("Отобразить имена служащих, включая номера их телефонов и идентификационные номера, из таблицы "Служащие", упорядочив их по фамилии и имени".)

Преобразование: Select last name, first name, phone number, and employee ID from the employees table and order by last name and first name
(Выбрать фамилию, имя, номер телефона и идентификационный номер служащего из таблицы "Служащие" и упорядочить их по фамилии и имени)

Уточнение: Select last name, first name, phone number, ~~and~~ employee ID from ~~the~~ employees table ~~and~~ order by last name ~~and~~ first name
(Выбрать фамилию, имя, номер телефона и идентификационный номер из "Служащие" и упорядочить по фамилии, имени)

SQL
```
SELECT EmpLastName, EmpFirstName,
    EmpPhoneNumber, EmployeeID
FROM Employees
ORDER BY EmpLastName, EmpFirstName
```

Со столбцами в условии ORDER BY можно проделать интересный опыт, определив разный порядок сортировки для каждого столбца. В предыдущем примере можно указать убывающий порядок для фамилии и возрастающий для имени. Так будет выглядеть оператор SELECT после внесения соответствующих изменений:

SQL
```
SELECT EmpLastName, EmpFirstName, EmpPhoneNumber,
    EmployeeID
FROM Employees
ORDER BY EmpLastName DESC, EmpFirstName ASC
```

Хотя использование ключевого слова ASC необязательно, при его включении оператор приобретает большую ясность.

Предыдущий пример вызывает интересный вопрос, имеет ли какое-либо значение расположение столбцов в условии ORDER BY. Ответ гласит *"Да!"*. Последовательность важна потому, что СУБД анализирует столбцы в условии ORDER BY слева направо. Важность последовательности возрастает прямо пропорционально количеству используемых столбцов. Всегда располагайте столбцы в условии ORDER BY в надлежащей последовательности, чтобы сортировка результата выполнялась в соответствующем порядке.

Сохранение работы

Сохраняйте свои операторы SELECT — каждая главная программа ПО для базы данных предоставляет способ для их сохранения. Это исключает необходимость их повторного написания при обращении с тем же запросом к базе данных. Когда вы сохраняете свой оператор SELCT, присваивайте ему осмысленное имя, которое поможет запомнить, какой вид информации он предоставляет. И, если программное обеспечение базы данных это позволяет, составьте краткое описание назначения оператора. Ценность описания станет совершенно понятной, когда вы после значительного интервала попытаетесь вспомнить, зачем этот оператор был создан.

В некоторых программах сохраненный оператор SELECT определяется как запрос, а в других — как представление. Независимо от его назначения каждая программа базы данных предоставляет средства его выполнения и работает с его набором результатов.

> **Внимание!** Далее в обсуждении будет использоваться слово “запрос” для обозначения сохраненного оператора SELECT и слова “выполнение” и “вызов” для обозначения метода, используемого для работы с ними.

Для выполнения запроса используются два общих метода. Первый — это интерактивный вызов (с помощью панели инструментов, таблицы запросов и т. п.), а второй — вызов из программного кода. Первый метод используется достаточно широко. О втором методе не стоит беспокоиться до тех пор, пока вы не начнете работать с языком программирования, используемым в программном обеспечении базы данных. Наша задача — научить вас создавать и использовать операторы SQL, а ваша задача — научиться создавать, сохранять и выполнять их в программе для вашей СУБД.

Примеры операторов

Теперь обратим внимание на некоторые примеры применения оператора SELECT и запроса SELECT в различных сценариях. В этих примерах представлены учебные базы данных, показано использование оператора SELECT, запроса SELECT и двух дополнительных методов, используемых для определения столбцов для преобразуемого утверждения. Также включены примеры наборов результатов, возвращаемых этими операциями и расположенных после синтаксической линии для SQL. Имя, которое появляется сразу же над набором результатов, предназначено для двух целей: оно используется для идентификации самого набора результатов, а также для присвоения оператору SQL.

Присвоение имени каждому оператору SQL связано с тем, что мы их сохраняем! Фактически мы поименовали и сохранили все операторы SQL, которые описываются в данной книге. Все они сохранены в соответствующем образце базы данных (см. примеры), и эти базы данных можно загрузить с сайта издательства “Лори” (www.lory-press.ru). Это даст возможность посмотреть указанные операторы в действии, прежде чем пытаться написать их самостоятельно.

Внимание! Еще раз напомним, что все имена столбцов и таблиц, используемые в этих примерах, взяты из структур учебных баз данных, показанных в приложении B.

База данных заказов на покупку

"Show me the names of all our vendors".
("Показать имена всех наших поставщиков".)

Преобразование: Select the vendor name
from the vendors table
(Выбрать имя поставщика
из таблицы "Поставщики")

Уточнение: Select ~~the~~ vendor name
from ~~the~~ vendors ~~table~~
(Выбрать имя поставщика
из "Поставщики")

SQL
```
SELECT VendName
FROM Vendors
```

Vendor_Names
(10 строк)

VendName
Shinoman, Incorporated
Viscount
Nikoma of America
ProFormance
Kona, Incorporated
Big Sky Mountain Bikes
Dog Ear
Sun Sports Suppliers
Lone Star Bike Supply
Armadillo Brand

"Which states do our customers come from?"
("В каких штатах находятся наши клиенты?")

Преобразование: Select distinct state
from the customers table
(Выбрать по одному штаты
из таблицы "Клиенты")

Уточнение: Select distinct state
from ~~the~~ customers ~~table~~
(Выбрать по одному штаты
из "Клиенты")

SQL
```
SELECT DISTINCT CustState
FROM Customers
```

Customer_States
(4 строки)

CustState
CA
OR
TX
WA

"What are the names and prices of all the products we carry?"
("Как называются и сколько стоят товары, которыми мы торгуем?")

Преобразование: Select product name, retail price from the products table
(Выбрать наименование товара, розничную цену
из таблицы "Продукты")

Уточнение: Select product name, retail price from ~~the~~ products ~~table~~
(Выбрать наименование товара, розничную цену
из “Продукты”)

SQL
```
SELECT ProductName, Retail Price
FROM Products
```

Product_Price_List (40 строк)

ProductName	RetailPrice
Trek 9000 Mountain Bike	$1,200.00
Eagle FS-3 Mountain Bike	$1,800.00
Dog Ear Cyclecomputer	$75.00
Victoria Pro All Weather Tires	$54.95
Dog Ear Helmet Mount Mirrors	$7.45
Viscount Mountain Bike	$635.00
Viscount C-500 Wireless Bike Computer	$49.00
Kryptonite Advanced 2000 U-Lock	$50.00
Nikoma Lok-Tight U-Lock	$33.00
Viscount Microshell Helmet	$36.00
<< остальные строки >>	

База данных эстрадных мероприятий

“List all entertainers, the cities they’re based in. and sort it by city and name in ascending order”.
(“Привести список артистов, города в которых они живут, и отсортировать их по городу и имени в порядке возрастания”.)

Преобразование: Select city and stage name from the entertainers table and order by city and stage name
(Выбрать город и псевдоним артиста из таблицы “Эстрадные артисты” и упорядочить по городу и псевдониму артиста)

Уточнение: Select city ~~and~~ stage name from ~~the~~ entertainers ~~table and~~ order by city ~~and~~ stage name
(Выбрать город, псевдоним артиста из “Эстрадные артисты”, упорядочить по городу, псевдониму артиста)

SQL

```
SELECT EntCity, EntStageName
FROM Entertainers
ORDER BY EntCity ASC, EntStageName ASC
```

Entertainer_Locations (13 строк)

EntCity	EntStageName
Auburn	Caroline Coie Cuartet
Auburn	Topazz
Bellevue	Albert Buchanan
Bellevue	Jazz Persuasion
Bellevue	Susan McLain
Redmond	Carol Peacock Trio
Redmond	JV & the Deep Six
Seattle	Coldwater Cattle Company
Seattle	Country Feeling
Seattle	Julia Schnebly
<<остальные строки>>	

"Give me a unique list of engagement dates; I'm not concerned with how many engagements there are per date".
("Предоставить список дат ангажементов без повторений. Неважно, сколько ангажементов приходится на день".)

Преобразование: Select distinct start date from the engagements table (Выбрать по одной даты начала из таблицы "Ангажементы")

Уточнение: Select distinct start date from ~~the~~ engagements ~~table~~ (Выбрать по одной даты начала из "Ангажементы")

SQL

```
SELECT DISTINCT StartDate
FROM Engagements
```

Engagement_Dates (66 строк)

StartDate
1999-07-01
1999-07-10
1999-07-11
1999-07-15
1999-07-17
1999-07-18
1999-07-24
1999-07-29
1999-07-30
1999-07-31
<<остальные строки>>

База данных расписания занятий

"Can we view complete class information?"
(Можно ли просмотреть полную информацию об уроках)

Преобразование: Select all columns from
the classes table
(Выбрать все столбцы
из таблицы "Уроки")

Уточнение: Select all columns from
the classes ~~table~~
(Выбрать все столбцы
из "Уроки")

SQL
```
SELECT *
FROM Classes
```

Class_Information (76 строк)

ClassID	Subject	Classroom	Credits	StartTime	Duration	<<other columns>>
1000	Introduction to Art	1231	5	10:00	50	...
1002	Design	1619	4	15:30	110	...
1004	Drawing	1627	4	08:00	50	...
1006	Drawing	1627	4	09:00	110	...
1012	Painting	1627	4	13:00	170	...
1020	Computer Art	3404	4	13:00	110	...
1030	Art History	1231	5	11:00	50	...
1031	Art History	1231	5	14:00	50	...
1156	Composition—Fundamentals	3443	5	08:00	50	...
1162	Composition—Fundamentals	3443	5	09:00	80	...
<<остальные строки>>						

"Give me a list of the buildings on campus and the number of floors for each building. Sort the list by building in ascending order".
("Предоставить список зданий на территории университетского городка с указанием количества этажей в каждом здании. Упорядочить список по зданиям в возрастающем порядке".)

Преобразование: Select building name and number of floors from the buildings table
(Выбрать название здания и количество этажей в нем из таблицы “Здания”)

Уточнение: Select building name ~~and~~ number of floors from ~~the~~ buildings ~~table~~
(Выбрать название здания и количество этажей в нем из “Здания”)

SQL

```
SELECT BuildingName, NumberOfFloors
FROM Buildings
ORDER BY BuildingName ASC
```

Building_List (6 строк)

BuildingNam	NumberOfFloors
Arts and Sciences	3
College Center	3
Instructional Building	3
Library	2
PE and Wellness	1
Technology Building	2

База данных лиги игры в боулинг

“Where are we holding our tourneys?”
(“Где проводятся наши турниры?”)

Преобразование: Select distinct tourney location from the tournaments table
(Выбрать по одному места проведения турниров из таблицы “Турниры”)

Уточнение: Select distinct tourney location from ~~the~~ tournaments ~~table~~
(Выбрать по одному места проведения турниров из “Турниры”)

SQL

```
SELECT DISTINCT
    TourneyLocation
FROM Tournaments
```

Tourney_Locations (7 строк)

TourneyLocation
Acapulco Lanes
Bolero Lanes
Imperial Lanes
Red Rooster Lanes
Sports World Lanes
Thunderbird Lanes
Totem Lanes

"Give me a list of all tourney dates and locations. I need the dates in descending order and the locations in alphabetical order".
("Предоставить список всех дат и мест проведения турниров. Мне нужны даты в убывающем порядке, а места проведения в алфавитном порядке".)

Преобразование: Select tourney date and location from the tournaments table and order by tourney date in descending order and location in ascending order
(Выбрать даты турниров и места проведения турниров из таблицы "Турниры" и упорядочить их по датам в убывающем порядке и по местам проведения в возрастающем порядке)

Уточнение: Select tourney date ~~and~~ location from ~~the~~ tournaments ~~table~~ and order by tourney date ~~in~~ descending ~~order~~ ~~and~~ location ~~in~~ ascending ~~order~~
(Выбрать даты турниров, места проведения из "Турниры", упорядочить по датам в убывающем порядке, по местам в возрастающем порядке)

SQL
```
SELECT TourneyDate, TourneyLocation
FROM Tournaments
ORDER BY TourneyDate DESC, TourneyLocation ASC
```

Tourney_Dates (14 строк)

TourneyDate	TourneyLocation
1999-09-04	Acapulco Lanes
1999-08-28	Totem Lanes
1999-08-21	Sports World Lanes
1999-08-14	Imperial Lanes
1999-08-07	Bolero Lanes
1999-07-31	Thunderbird Lanes
1999-07-24	Red Rooster Lanes
1999-07-17	Acapulco Lanes
1999-07-10	Totem Lanes
1999-07-03	Sports World Lanes
<<остальные строки>>	

База данных рецептов

"What types of recipes do we have, and what are the names of the recipes we have for each type? Can you sort the information by type and recipe name?"
("Какие у нас имеются виды рецептов и какие названия рецептов имеются для каждого из видов? Можно ли привести информацию по видам и названиям рецептов?")

Преобразование: Select recipe class ID and recipe title from the recipes table and order by recipe class ID and recipe title
(Выбрать идентификатор класса рецептов и заголовок рецепта из таблицы "Рецепты" и упорядочить по идентификатору классов рецептов и заголовкам рецептов)

Уточнение: Select recipe class ID ~~and~~ recipe title from ~~the~~ recipes ~~table and~~ order by recipe class ID ~~and~~ recipe title
(Выбрать идентификатор класса рецептов, заголовок рецепта из "Рецепты", упорядочить по идентификатору классов рецептов, заголовкам рецептов)

SQL

```
SELECT RecipeClassID, RecipeTitle
FROM Recipes
ORDER BY RecipeClassID ASC, RecipeTitle ASC
```

Recipe_Classes_And_Titles (15 строк)

RecipeClassID	RecipeTitle
1	Fettuccini Alfredo
1	Huachinango Veracruzana (Red Snapper, Veracruz style)
1	Irish Stew
1	Pollo Picoso
1	Roast Beef
1	Salmon Filets in Parchment Paper
1	Tourtière (French-Canadian Pork Pie)
2	Asparagus
2	Garlic Green Beans
3	Yorkshire Pudding
<< остальные строки >>	

"Show me a list of unique recipe class IDs in the recipes table".
("Показать список неповторяющихся идентификаторов классов рецептов из таблицы "Рецепты".)

Преобразование: Select distinct recipe class ID from the recipes table (Выбрать неповторяющиеся идентификаторы классов рецептов из таблицы "Рецепты")

Уточнение: Select distinct recipe class ID from ~~the~~ recipes ~~table~~ (Выбрать неповторяющиеся идентификаторы классов рецептов из "Рецепты")

SQL
```
SELECT DISTINCT RecipClassID
FROM Recipes
```

Recipe_Class_Ids (6 строк)

RecipeClassID
1
2
3
4
5
6

Итоги

В данной главе мы познакомились с операцией SELECT. Это одна из четырех операций манипулирования данными в SQL. Операцию SELECT можно разделить на три более мелкие компонента: оператор SELECT, выражение SELECT и запрос SELECT.

При обсуждении оператора SELECT были представлены условия, являющиеся его компонентами. Условия SELECT и FROM являются основными, необходимыми для извлечения информации из базы данных. Остальные условия — WHERE, GROUP BY и HAVING — используются для обработки и фильтрации информации, возвращенной условием SELECT, с учетом этих условий.

Мы показали различия данных и информации. Значения, сохраненные в базе данных, являются данными, а информацией являются данные, обработанные таким образом, чтобы у них появился смысл для просматривающего их человека. Строки информации, возвращенные оператором SELECT, называются набором результатов.

Вопрос об извлечении информации мы начали с представления основной формы оператора SELECT. Вы узнали, как построить соответствующий оператор SELECT, используя методику из трех этапов, которая включает анализ запроса и преобразование его в соответствующий синтаксис SQL. В операторе SELECT можно использовать два или более столбцов, чтобы увеличить объем информации, извлекаемой из базы данных. Затем мы кратко рассмотрели ключевое слово DISTINCT, и вы узнали, что существуют средства, позволяющие исключить дубликаты строк из набора результатов.

Запрос SELECT можно объединить с оператором SELECT для сортировки набора результатов, полученного для этого оператора. Это необходимо, поскольку запрос SELECT является единственной операцией SELECT, которая содержит условие ORDER BY. Условие ORDER BY используется для сортировки информации одного или нескольких столбцов, и для каждого столбца можно указать его собственный, возрастающий или убывающий, порядок сортировки. Необходимо сохранять свои операторы SELECT для будущего их использования в качестве запроса или представления.

Мы привели несколько примеров, использующих различные таблицы из учебных баз данных. В примерах иллюстрируется, как различные концепции и методики, представленные в данной главе, используются в типичных сценариях и приложениях. В следующей главе мы более детально рассмотрим условие SELECT и покажем, как извлечь что-либо еще, помимо информации, из списка столбцов.

Задачи для самостоятельного решения

Ниже приведены формулировки запросов и имена решений этих запросов в учебных базах данных. Попрактикуйтесь немного и составьте SQL для каждого запроса, а затем сверьте свой ответ с запросом, который сохранен нами в этих базах данных. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов,— важно, чтобы набор результатов был тем же.

База данных заказов на закупку

1. *"Show me all the information on our employees"*.
   ("Показать всю информацию по нашим сотрудникам".)
   Решение можно найти в Employee_Information (8 строк).

2. *"Show me a list of cities, in alphabetical order, where our vendors are located, and include the names of the vendors we work with in each city"*.
   ("Показать в алфавитном порядке список городов, в которых размещены наши поставщики, и включить в него имена всех поставщиков, с которыми мы сотрудничаем в каждом городе".)
   Решение можно найти в Vendor_Locations (10 строк).

База данных эстрадных мероприятий

1. *"Give me the names and phone numbers of all our agents, and list them in last name/first name order"*.
   ("Найти телефонные номера всех наших агентов и представить их в упорядоченном виде по именам/фамилиям".)
   Решение можно найти в Agent_Phone_list (8 строк).

2. *"Give me the information on all our engagements".*
("Предоставить информацию по ангажементам".)
Решение можно найти в Engagement_Information (131 строка).

3. *"List all engagements and their associated start dates. Sort the records by date in descending order and by engagement in ascending order".*
("Составить список всех ангажементов и дат их начала.
Отсортировать записи по дате в убывающем порядке,
а по ангажементам в возрастающем порядке".)
Решение можно найти в Scheduled_Engagements (131 строка).

База данных расписания занятий

1. *"Show me a complete list of all the subjects we offer".*
("Показать список всех предлагаемых нами предметов".)
Решение можно найти в Subject_List (56 строк).

2. *"What kinds of titles are associated with our faculty?".*
("Какие заголовки связаны с нашим факультетом?")
Решение можно найти в Faculty_Titles (3 строки).

3. *"List the names and phone number of all our staff, and sort them by last name and first name".*
("Привести список имен и телефонных номеров для всего персонала
и отсортировать его по фамилии и имени".)
Решение можно найти в Staff_Phone_List (27 строк).

База данных лиги игры в боулинг

1. *"List all of the teams in alphabetical order".*
("Привести список команд в алфавитном порядке".)
Решение можно найти в Team_List (8 строк).

2. *"Show me all the bowling score information for each of our members".*
("Показать всю информацию о счетах по боулингу для каждого
из наших участников".)
Решение можно найти в Bowling_Score_Information (1344 строки).

3. *"Show me a list of bowlers with their current average and handicap, and sort it in alphabetical order".*
("Показать список игроков с их текущим средним счетом и гандикапом
и отсортировать его в алфавитном порядке".)
Решение можно найти в Bowler_Statistics (32 строки).

База данных рецептов

1. *"Show me a list of all the ingredients we currently keep track of"*.
("Привести список всех ингредиентов, за которыми мы следим".)
Решение можно найти в Complete_Ingredients_List (79 строк).

2. *"Show me all the main recipe information, and sort it by the name of the recipe in alphabetical order"*.
("Показать информацию о всех основных рецептах и о наименовании рецептов в алфавитном порядке".)
Решение можно найти в Main_Recipe_Information (15 строк).

5

Как получить нечто большее, чем просто столбцы

"Факты — упрямая вещь".

— Тобиас Смоллетт
Жиль Блас из Сантильяны

Вопросы, рассматриваемые в данной главе:

- Условие SELECT, дубль два
- За пределами азов
- Что такое "выражение"
- Что вы пытаетесь выразить
- Типы выражений
- Использование выражений в условии SELECT
- Значение Null
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

Из главы 4 вы узнали, как использовать условие SELECT для извлечения информации из одного или нескольких столбцов таблицы. Эта методика полезна, если к базе данных обращаются с простыми запросами относительно некоторых основных фактов. Однако для работы со сложными запросами потребуется расширить свой "словарный запас" SQL. В данной главе мы обсудим, как *тип* данных, сохраненных в столбце, может оказывать важное влияние на запросы и как настроить область информации, извлекаемой из базы данных, используя *выражения* для манипулирования данными. Начнем с пересмотра условия SELECT.

Условие SELECT, дубль два

Информация из столбцов в таблице извлекается путем перечисления в списке соответствующих имен столбцов в условии SELECT оператора SELECT. Например, для извлечения имени, фамилии и телефонного номера каждого служащего из таблицы Employees можно использовать следующий оператор SELECT:

```
SQL                    SELECT FirstName, LastName, PhoneNumber
                       FROM Employees
```

Это самый общий метод извлечения информации из таблицы. При определении имени столбца в условии SELECT используется то, что в стандарте SQL называется *ссылкой на столбец*. На рис. 5.1 представлена синтаксическая диаграмма для этого термина.

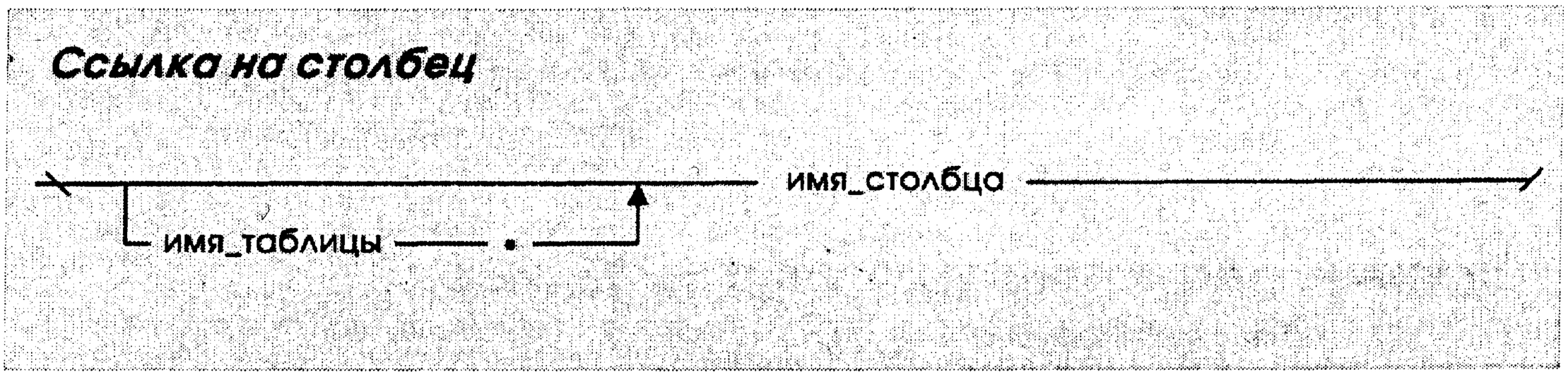


Рис. 5.1. *Синтаксическая диаграмма для ссылки на столбец*

Хотя в условии SELECT можно использовать просто имя столбца, также можно явно указать и имя таблицы, к которой он относится (родительской таблицы). Ниже показано, как переписать предыдущий оператор SELECT для включения уточненных имен столбцов:

```
SQL                    SELECT Employees.FirstName, Employees.LastName,
                           Employees.PhoneNumber
                       FROM Employees
```

Если оператор SELECT обращается к одной таблице, нет необходимости заботиться об уточнении каждого имени столбца, но когда он обращается к нескольким таблицам, часто возникают неоднозначности (подробнее см. в главе 8).

Определение явных значений

Стандарт SQL предоставляет возможность улучшения информации, возвращаемой оператором SELECT, и разрешает использование таких значений, как символьные строки, числа, даты, время или подходящая комбинация этих элементов в любом допустимом выражении, используемом с оператором SELECT. Стандарт SQL определяет категории этих типов значений как *значения литералов* и устанавливает способ их определения.

Строковые литералы

Строковый литерал представляет собой последовательность отдельных символов, заключенных в апострофы (*одинарные* кавычки). Вы, вероятно, привыкли использовать двойные кавычки для заключения строк символов, но здесь представлена концепция, определенная в стандарте SQL. На рис. 5.2 показана синтаксическая диаграмма строкового литерала.

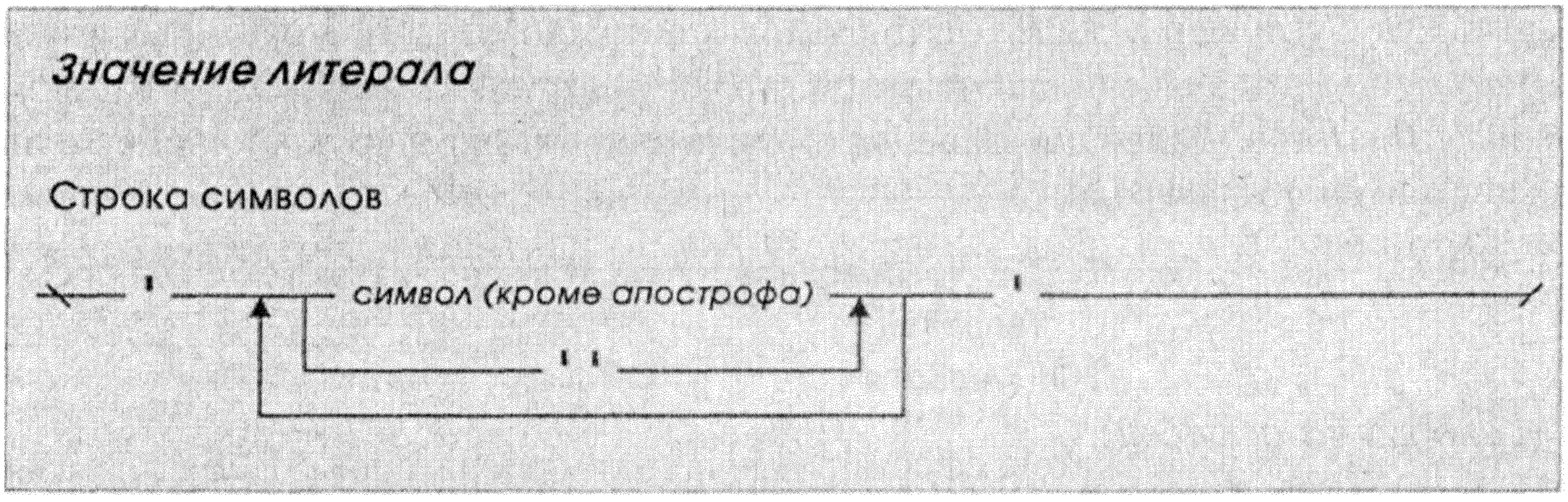


Рис. 5.2. *Диаграмма строкового литерала*

Далее приведено несколько примеров литералов типа строки символов, которые можно определить:

```
'Это пример строкового литерала.'
'Если нужно поставить апостроф внутри строки, это делается вот так:
'' - и готово!'
'B-28'
'Seattle'
```

Вы, наверное, обратили внимание на двойные кавычки в середине второй строки. На самом деле это не двойные кавычки, а два последовательных символа одиночных кавычек без пробела между ними. Согласно стандарту SQL одиночная кавычка (апостроф) внутри строки символов представляется двумя последовательными одиночными кавычками, чтобы СУБД могла отличить символ одиночной кавычки, который определяет начало или конец литерала типа символьной строки, и кавычку, которую нужно включить в литерал. Следующие две строки показывают, как это работает:

SQL	`'The Vendor''s name is: '`
Отображается как	**The Vendor's name is:**

Строковые литералы можно использовать для более осмысленного представления информации, возвращаемой оператором SELECT. Хотя информацию из набора результатов обычно легко понять, весьма вероятно, что эту информацию можно сделать более понятной. Например, при выполнении следующего оператора SELECT

информация набора результатов отображает только адрес Web-узла поставщика и имя поставщика:

```
SQL                    SELECT VendWebPage, VendName
                       FROM Vendors
```

В некоторых случаях можно повысить понятность информации, определив строку символов, в которой содержится дополнительный текст с пояснением, а затем добавить его к условию SELECT. Этот метод стоит использовать в разумных пределах, потому что это значение символьной строки будет появляться в каждой строке набора результатов. Далее показано, как можно изменить предыдущий пример с помощью строкового литерала:

```
SQL                    SELECT VendWebPage, 'is the web site for',
                           VendName
                       FROM Vendors
```

Строка набора результатов, сформированная этим оператором SELECT, выглядит следующим образом:

www.datatexcg.com	is the web site for	DataTex Consulting Group

Это в некоторой степени проясняет информацию, отображенную набором результатов, устанавливая действительное назначение Web-адреса. Хотя это и простой пример, он показывает, что можно делать со строковыми литералами.

> **Внимание!** Эта методика особенно полезна при работе со старыми базами данных, которые содержат непонятные имена столбцов. Однако, если следовать рекомендациям главы 2, вам не потребуется использовать эту методику слишком часто со своими собственными базами данных.

Цифровые литералы

Цифровой литерал является еще одним типом литерала, который можно использовать в операторе SELECT. Он состоит из цифр со знаком и может включать десятичные разряды, символ, обозначающий порядок, и число, указывающее порядок. На рис. 5.3 представлена синтаксическая диаграмма для цифрового литерала.

Вот примеры цифровых литералов:

```
427
-11.253
.554
0.3E-3
```

Наибольшую пользу дает применение цифровых литералов в выражениях (см. ниже).

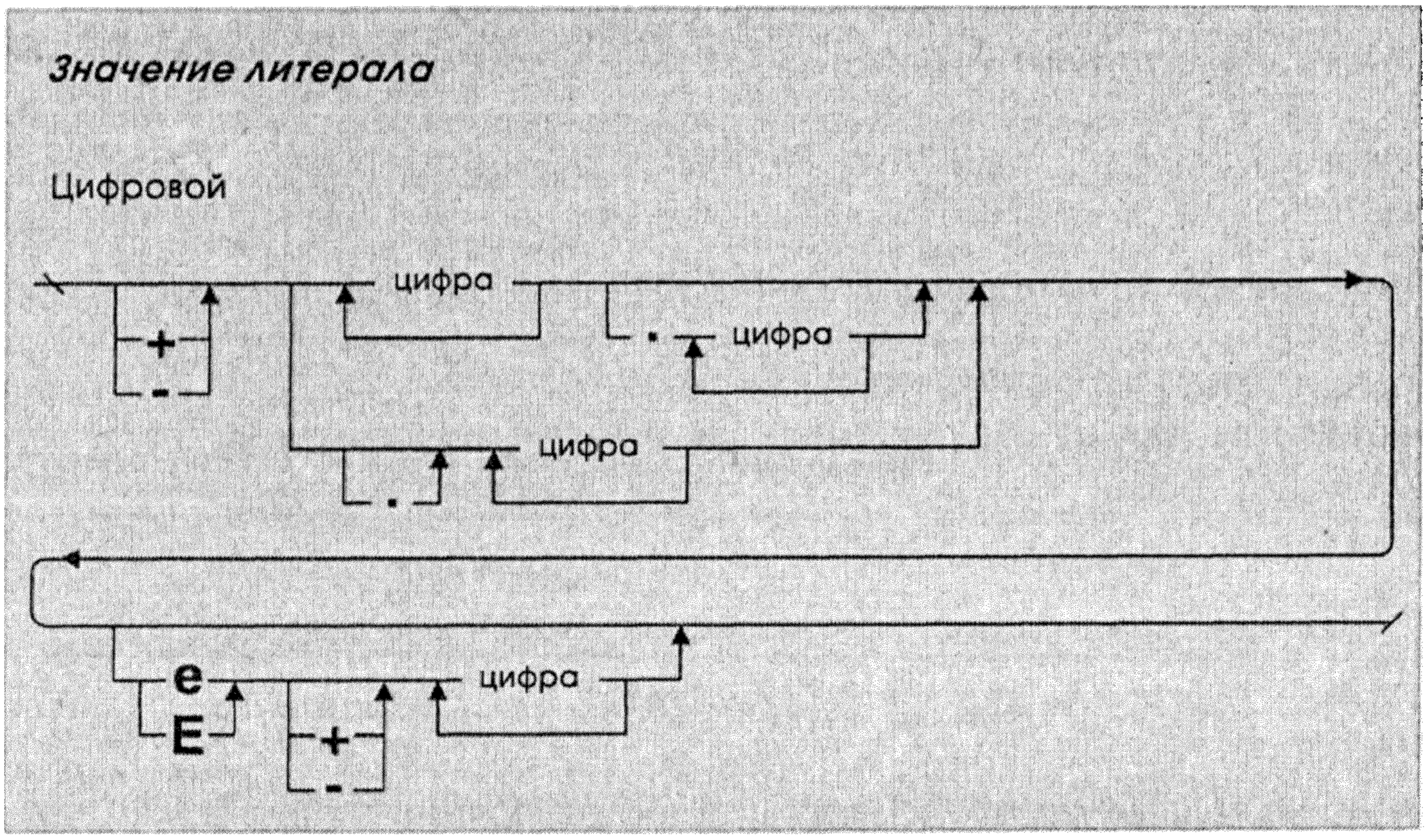


Рис. 5.3. *Диаграмма значения цифрового литерала*

Литералы даты и времени

Указать конкретные даты и время для использования в операторе SELECT можно с помощью *литералов даты* или *времени*. В стандарте SQL они называются *литералами даты/времени*. Их определение не вызывает проблем (см. рис. 5.4). Однако при использовании литералов даты/времени следует помнить о некоторых моментах.

Дата	Формат литерала даты: год-месяц-день (этот формат используется во всей книге). Однако многие базы данных SQL допускают более привычный формат — месяц-день-год (США) или формат день/месяц/год (европейский формат).
Время	Часовой формат основывается на 24-часовом представлении времени. Например, американское 07:00 p.m. представляется как 19:00.

Ниже приведены некоторые примеры литерала дата/время.

```
DATE '1999-05-16'
DATE '2016-11-22'
TIME '21:00'
TIME '03:30:25'
```

Оба литерала состоят из ключевого слова и строки символов, указывающих нужное значение. Хотя ключевые слова DATE и TIME определяются стандартом SQL как обязательные компоненты литералов даты и времени, большинство СУБД редко поддерживают эти ключевые слова в конкретном контексте и требуют только

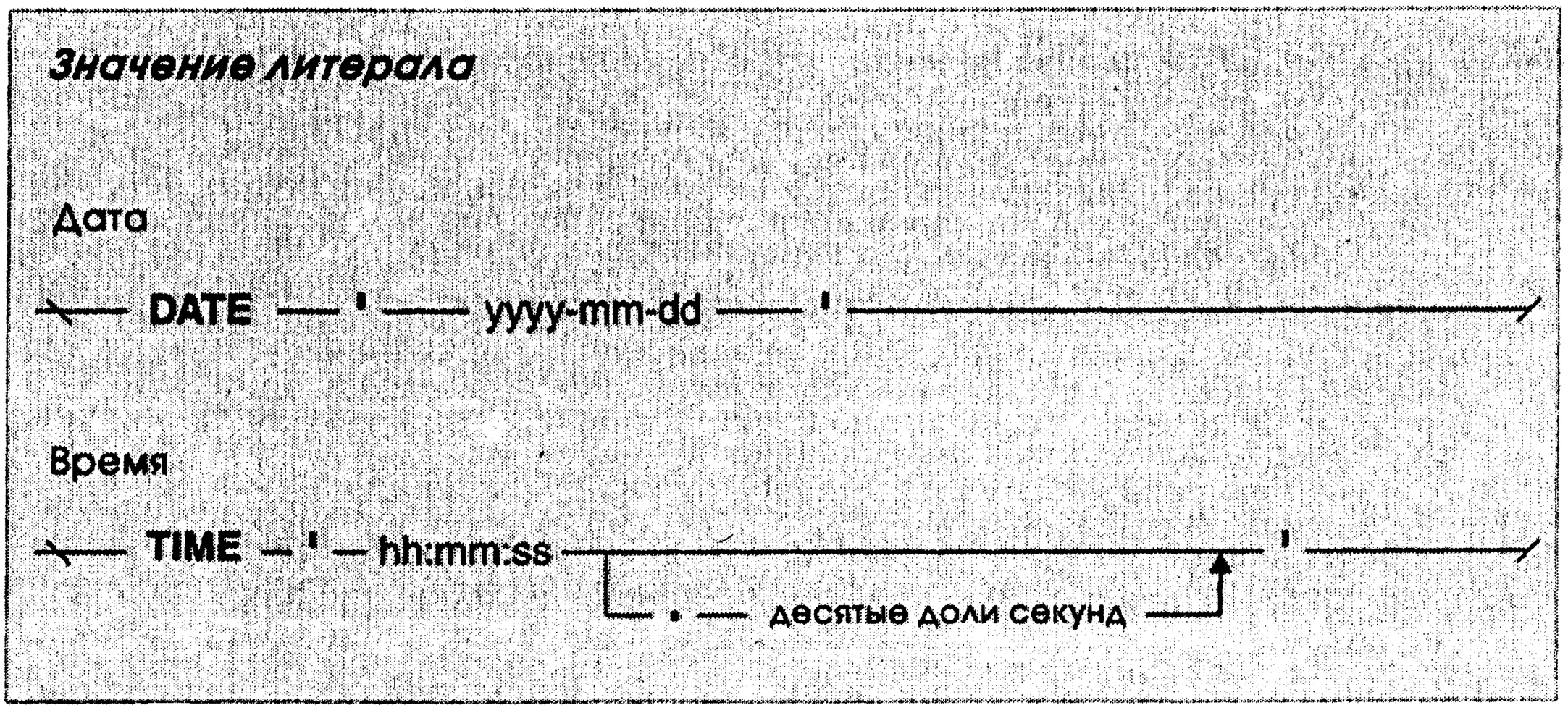


Рис. 5.4. *Диаграмма литерала дата/время*

часть, составляющую символьную строку литерала. Поэтому мы будем использовать вместо них одиночные кавычки для выделения литерала даты или времени (в примерах в остальной части книги). Использование даты и времени в выражениях обсуждается позже в данной главе.

За пределами азов

Представленные нами методы позволят получить ответы на простые запросы, с которыми обращаются к базе данных. Теперь посмотрим, как поступать с более сложными запросами.

Стандарт SQL предоставляет инструменты, необходимые для построения сложных запросов. Решение о том, какие инструменты использовать, зависит от характера самих запросов. Ниже приводится несколько примеров:

- Для нахождения имен преподавателей, которые ведут определенный курс лекций, введите *условие поиска* в условие WHERE оператора SELECT (см. главу 6).
- Для просмотра информации о входящих в штат врачах и их пациентах, для сбора информации из соответствующих таблиц воспользуйтесь одним или несколькими условиями JOIN (см. главу 8).
- Для фильтрации групп данных введите условие поиска в условие HAVING оператора SELECT (см. главу 14).

По мере освоения книги мы изучим, как использовать эти методы. Они упомянуты здесь, поскольку в них есть кое-что общее: каждая методика требует использования *выражения* для выполнения запроса надлежащим образом. Поскольку выражение является критическим компонентом всех операторов SQL, используемых для получения ответа на сложные запросы, очень важно хорошо понимать выражения, прежде чем переходить к изучению других методов.

Что такое "выражение"

Выражение является некоторым видом операции, которая включает числа, символьные строки или значения даты и времени. В нем можно использовать значения, извлеченные из конкретных столбцов таблицы, значения литералов или их комбинацию. Как только операция, определенная выражением, завершена, оно возвращает значение оператору SQL для дальнейшей обработки. Выражения можно использовать для расширения или сужения объема информации, извлекаемой из базы данных. Выражения особенно полезны при формулировании вопросов “что если?”. Далее приведены примеры типов запросов:

Какова общая сумма для элементов каждой строки?

Предоставить список сотрудников, указав вначале фамилию.

Показать время начала и окончания для каждого занятия.

Показать разницу очков между Handicap Score и Raw Score для каждого игрока в боулинг.

Каков размер почасовой ставки для каждого ангажемента?

Что будет, если мы поднимем цены на наши изделия на 5%?

Тип данных, используемых в выражении, оказывает влияние на значение, возвращаемое выражением, поэтому вначале рассмотрим некоторые типы данных, предлагаемые стандартом SQL.

Что вы пытаетесь выразить

Каждому столбцу в базе данных присвоен *тип данных*, определяющий вид значений, которые столбец может хранить. Тип данных также определяет операции, которые можно выполнять со значениями столбцов. Если известны типы данных столбцов, используемых в выражении, то можно гарантировать, что выражение является корректным и возвратит надлежащее значение.

Типы данных SQL

Стандарт SQL определяет семь основных типов данных в рамках трех общих категорий: символьных, цифровых и дата/время. В свою очередь, для каждого типа имеется одна или несколько вариаций с уникальными наименованиями, которые также известны как типы данных. Приведем краткий обзор каждого из этих типов.

CHARACTER (символьный)	Этот тип данных служит для хранения символьных строк фиксированной или переменной длины, состоящих из одного или нескольких символов. Допустимые для него символы обычно основаны на наборах символов ASCII или EBCDIC. Данные типа “символьный

	фиксированной длины" обозначаются кодовым словом CHARACTER или CHAR, а данные типа "символьный переменной длины" обозначаются кодовым словом CHARACTER VARYING, CHAR VARYING или VARCHAR. Размер данных типа "символьный фиксированной длины" определяется пользователем, а максимальный размер данных типа "символьный переменной длины" определяется системой базы данных. (Это правило также применяется к типу данных National Character — национальные символы.)
National Character (национальные символы)	Тип данных National Character является таким же, что и тип данных Character, за исключением того, что он использует символы из наборов символов для иностранных языков, определенных ISO. Для обозначения National Character фиксированной длины используются имена NATIONAL CHARACTER, NATIONAL CHAR и NCHAR, тогда как NATIONAL CHARACTER VARYING, NATIONAL CHAR VARYING и NCHAR VARYING являются именами, используемыми для обозначения типа National Character переменной длины.
Bit (битовый)	В данных этого типа можно сохранять строки последовательностей двоичных чисел, например оцифрованные изображения и звуковые волны. СУБД определяет размер данных типа, как для целых чисел. Этот тип данных может указываться, как BIT или BIT VARYING.
Exact Numeric (точный цифровой)	Этот тип данных сохраняет целые числа и числа с десятичными разрядами. Точность (число значащих цифр) и масштаб (число цифр справа от десятичной точки) для Exact Numeric может определяться пользователем и может быть только равной или меньше максимальных пределов, допустимых системой базы данных. NUMERIC, DECIMAL, DEC, INTEGER, INT и SMALLINT — это все имена, используемые для ссылки на этот тип данных. Однако стандарт SQL — как и большинство СУБД — определяет INTEGER как имеющий больший диапазон значений, чем SMALLINT. Проверьте в своей документации по системе базы данных допустимые пределы.

Approximate Numeric (приблизительный цифровой)	Данные с этим типом сохраняют числа в виде десятичной дробной части и значения порядка. Имена, используемые для указания этого типа данных, включают FLOAT, REAL и DOUBLE PRECISION. Approximate Numeric не содержат точность и масштаб, как таковые, но стандарт SQL допускает точность, определенную пользователем, только для данных типа REAL. Любой масштаб, связанный с этими типами данных, всегда определяется системой базы данных. Стандарт SQL, как и большинство СУБД, определяет диапазон значений для данных типа DOUBLE PRECISION большим, чем диапазон для данных типа REAL. Также уточните в своей документации и эти диапазоны.
Datetime (дата/время)	В данных этого типа сохраняются даты, время и их комбинации. Стандарт SQL определяет формат данных как год-месяц-день, а значения времени определяются на основе 24-часового представления. Хотя большинство СУБД позволяет использовать более общие форматы даты: месяц/день/год или день/месяц/год и значения времени на основе представления a.m./p.m., в этой книге используются форматы для даты и времени, определенные стандартом SQL. Для указания такого типа данных используются три имени: DATE, TIME и TIMESTAMP. Для сохранения комбинации даты и времени можно использовать тип данных TIMESTAMP.
Interval (интервал)	С этим типом данных сохраняется промежуток времени между двумя значениями дата-время, выраженными как год, месяц; год/месяц; день, время; или день/время. Большинство СУБД пока еще не поддерживает этот тип данных.

Многие системы БД предусматривают дополнительные типы данных (помимо определенных стандартом SQL), которые называются *расширенными типами данных*. К ним относятся: MONEY/CURRENCY, BOOLEAN (для значений True/False), SERIAL/ROWID (для уникальных идентификаторов строк) и BYTE/BLOB (для неструктурированных двоичных данных).

Поскольку мы рассматриваем исключительно ту часть SQL, которая связана с *манипулированием данными*, необходимо позаботиться о соответствующих диапазонах значений для каждого типа данных, которые поддерживает ваша СУБД. Это поможет гарантировать, что определенные вами выражения будут выполнены надлежащим образом, поэтому вы должны быть хорошо знакомы с типами данных, предусмотренными вашей СУБД.

Теперь перейдем к обсуждению процесса построения простых выражений.

Типы выражений

Обычно при работе с операторами SQL используются следующие три типа выражений:

Сцепление (конкатенация)	Объединение двух или более элементов в одну символьную строку.
Математические	Сложение, вычитание, умножение и деление.
Арифметические для даты/времени	Применение сложения или вычитания к датам и времени

Сцепление

Стандарт SQL определяет две вертикальные полосы как обозначение оператора конкатенации, или сцепления. Можно объединить два элемента, поместив между ними оператор сцепления. Результатом будет одна строка символов, т. е. соединение обоих элементов. На рис. 5.5 представлена синтаксическая диаграмма для выражения-объединения.

Далее приведено общее представление операции сцепления:

```
Выражение          Первый_Элемент || Второй_Элемент
```

Результат **Первый_ЭлементВторой_Элемент**

Начнем с самого простого примера в мире: с объединения двух строковых литералов, а именно — имени и фамилии:

```
Выражение          'Mike' || 'Hernandez'
```

Результат **MikeHernandez**

Здесь необходимо обсудить два момента. Во-первых, как имя, так и фамилию необходимо заключить в одиночные кавычки, потому что они являются строковыми литералами. Во-вторых, имя и фамилия в результате операции слились. Хотя операция выполнила правильное объединение, это не совсем то, что ожидалось. Решение состоит в добавлении между ними пробела.

```
Выражение          'Mike' || ' ' || 'Hernandez'
```

Результат **Mike Hernandez**

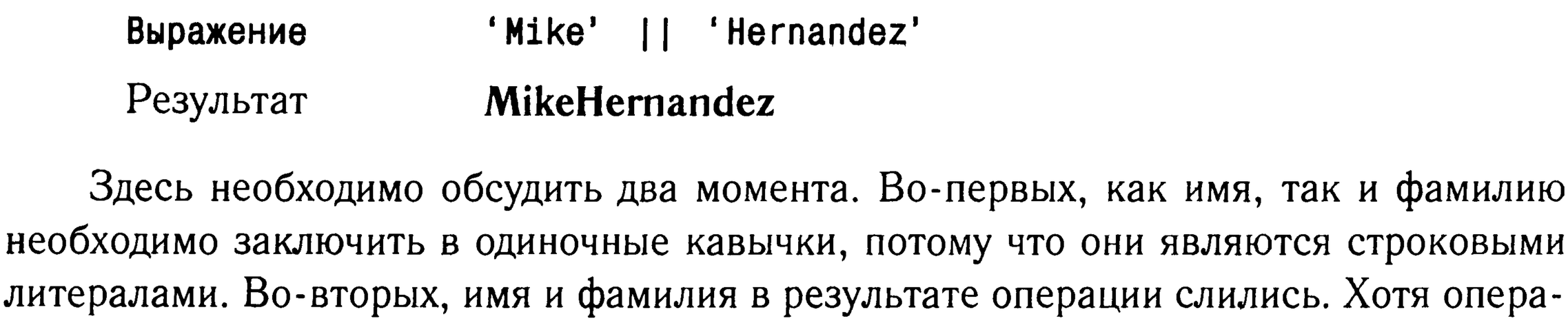


Рис. 5.5. *Синтаксическая диаграмма для выражения-объединения*

Этот пример показывает, что можно объединять дополнительные символьные значения, используя несколько операторов сцепления. Ограничения на количество символьных значений, которые можно объединять, отсутствуют, но существует предел на максимальную длину символьной строки, которую возвращает операция сцепления. В общем случае длина символьной строки, возвращенной операцией сцепления, не может быть больше, чем максимальная длина, допустимая для данных типа Character переменной длины. В вашей СУБД этот вопрос может решиться немного иначе, поэтому уточните дополнительные детали в своей документации.

Объединение двух или более символьных строк само по себе имеет смысл, но указанным способом также можно объединять значения двух и более столбцов. Предположим, имеется два столбца с именами CompanyName и City. Можно создать выражение, которое объединяет значения этих столбцов, используя их имена в выражении. В приведенном ниже примере выполняется объединение значений из этих столбцов в символьную строку:

Выражение	`CompanyName || ' расположена в ' || City`
Результат	**DataTex Consulting Group расположена в Seattle**

В данном случае CompanyName или City не требуется заключать в одиночные кавычки, потому что это ссылки на столбцы. Ссылку на столбец можно использовать в выражении любого типа (как видно из примеров в остальной части книги).

Можно также объединять даты или числа с символьными строками, но для этого необходимо использовать функцию *CAST* (см. рис. 5.6).

Функция CAST преобразует значение литерала или значение столбца в конкретный тип данных. Это помогает гарантировать, что типы данных значений в выражении являются "совместимыми". Все значения, которые используются в выражении, должны быть совместимы для того, чтобы операция, определенная в выражении, работала надлежащим образом. В противном случае СУБД обязательно выдаст сообщение об ошибке.

Внимание! Каждая СУБД располагает функцией или набором функций, которые можно применять для преобразования типа данных. Хотя стандарт SQL явно определяет функцию CAST, она может и не использоваться в конкретной СУБД. Уточните детали в отношении функций преобразования, предусмотренных в вашей системе, в документации по системе базы данных.

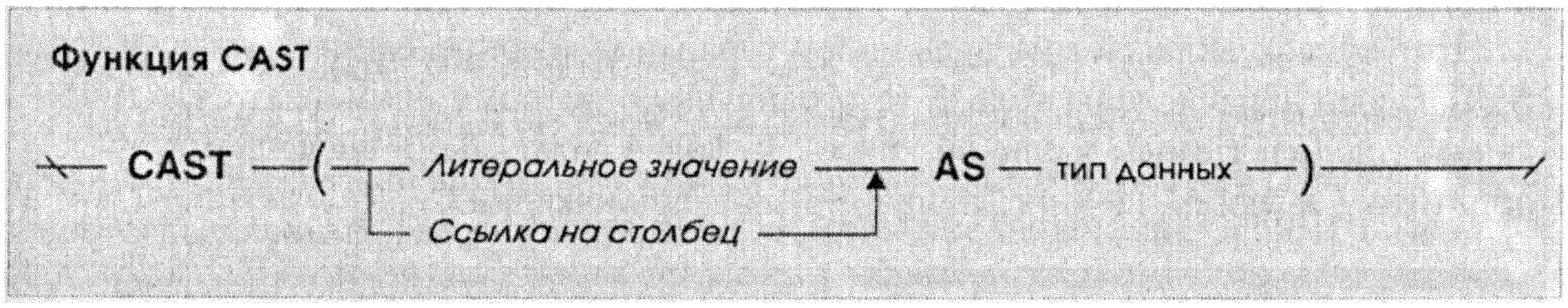


Рис. 5.6. *Синтаксическая диаграмма для функции CAST*

Преобразование значения литерала из одного типа данных в другой является относительно интуитивной и простой задачей. Однако при преобразовании значения столбца из его исходного типа данных в другой тип необходимо учитывать следующие ограничения:

- Значение символьного столбца переменной длины может усекаться (VARCHAR), если оно преобразуется в символьный столбец фиксированной длины (CHARACTER). СУБД должна выдать предупреждение, что может произойти усечение.
- Символьный столбец можно преобразовать в любой другой тип данных, но значение столбца должно представлять собой допустимое значение литерала получаемого типа данных. Обратите внимание на то, что СУБД игнорирует все ведущие и/или концевые пробелы при преобразовании значения символьного столбца в числовое значение или значение дата/время.
- При преобразовании значения цифрового столбца в другой тип данных Numeric, СУБД выдаст ошибку, если значение не соответствует типу данных результата. Например, вероятно, будет получена ошибка, если попытаться преобразовать значение типа REAL, большее чем 32 767, в SMALLINT. К тому же при преобразовании числа с дробной частью в INTEGER или SMALLINT цифры справа от десятичной точки будут отброшены или округлены, в зависимости от ситуации. Результат усечения или округления определяется системой базы данных.
- При преобразовании значения цифрового столбца в тип данных Character будет получен один из трех возможных результатов:
  1. Преобразование выполнится успешно.
  2. Результат будет дополнен пробелами, если его длина окажется меньше, чем длина, определенная для символьного столбца.
  3. СУБД выдаст ошибку, если символьное представление цифрового значения длиннее, чем длина, определенная для символьного столбца.

Внимание! Хотя Стандарт SQL определяет эти ограничения, ваша СУБД может дать некоторую свободу действий при преобразовании значения из одного типа данных в другой. За подробностями обратитесь к документации по вашей системе базы данных.

Этот список включает не весь набор ограничений, определенных стандартом SQL. Здесь перечислены только те ограничения, которые применяются к типам данных, используемым в данной книге. За более детальными сведениями обратитесь к любой из книг, приведенных в приложении С.

Теперь рассмотрим, как рименять функцию CAST в выражении со сцеплением.

Для объединения строкового литерала или значения символьного столбца с литералом типа "дата" или значением столбца даты воспользуйтесь функцией CAST для преобразования значения даты в символьную строку. Рассмотрим пример использования CAST для преобразования значения столбца даты с именем DateEntered (Введенная дата):

```
Выражение       EntStageName || ' заключил контракт с агентством '
                || CAST(DateEntered as CHARACTER)
```

Результат **Modern Dance заключил контракт с агентством 1999-05-16**

Функция CAST также может использоваться для объединения цифрового литерала или значения цифрового столбца с символьным типом данных. В следующем примере CAST используется для преобразования значения цифрового столбца с именем RetailPrice (Розничная цена):

```
Выражение       ProductName || ' продается по '
                || CAST(RetailPrice AS CHARACTER)
```

Результат **Trek 9000 Mountain Bike продается по $1200.00**

В выражении со сцеплением могут одновременно использоваться символьные строки, значения типа дата/время и цифровые значения. В следующем примере показано, как можно использовать все три типа данных в рамках одного выражения:

```
Выражение       'Заказ номер' || Cast(OrderNumber AS CHARACTER) ||
                ' был размещен ' || CAST(OrderDate AS CHARACTER)
```

Результат **Заказ номер 1 был размещен 1999-07-04**

> **Внимание!** Стандарт SQL определяет многообразие функций, которые можно использовать для извлечения информации из столбца или для вычисления значения по некоторому диапазону строк. Часть этих функций подробно рассматривается в главе 12.

Теперь рассмотрим различные типы выражений, которые можно создавать, используя цифровые данные.

Математические выражения

Стандарт SQL определяет сложение, вычитание, умножение и деление как операции, которые можно выполнять над цифровыми данными. Понятно, что это достаточно ограниченный набор операций. К счастью, большинство СУРБД предлагает намного более широкое разнообразие операций, включая модуль, квадратный корень,

экспоненту и абсолютный показатель степени. Они также обеспечивают широкий набор научных, тригонометрических, статистических, а также математических функций. Для наших целей мы ограничимся только операциями, определенными стандартом SQL.

Порядок, в котором выполняются эти четыре математические операции (называемый *порядком предшествования*), важен при составлении математических выражений. Стандарт SQL присваивает равный приоритет для умножения и деления и определяет, что они должны выполняться прежде любого сложения или вычитания. Однако в большинстве СУБД математические выражения по умолчанию вычисляются слева направо. Это может привести к некоторым интересным результатам в зависимости от того, как было построено выражение. Поэтому мы настоятельно рекомендуем активно изучить использование круглых скобок в сложных математических выражениях, чтобы гарантировать их надлежащее вычисление.

Если вы помните, как записывались математические выражения в школе, то вам уже известно, как записать их в SQL. По существу, для создания выражения используется цифровое значение, возможно, со знаком, математический оператор и другое цифровое значение, не обязательно со знаком. На рис. 5.7 представлена диаграмма этого процесса.

Приведем несколько примеров математических выражений, использующих значения литерала, ссылки на столбец и их комбинацию:

```
25 + 35
-12 * 22
Retail Price * QuantityOnHand
Total Score / GamesBowled
Retail Price - 2.50
Total Score / 12
```

Для того чтобы гарантировать надлежащее вычисление сложных математических выражений, необходимо использовать круглые скобки. Далее приведено несколько простых примеров использования круглых скобок:

```
Выражение          (11 - 4) + (12 * 3)
Результат          43
```

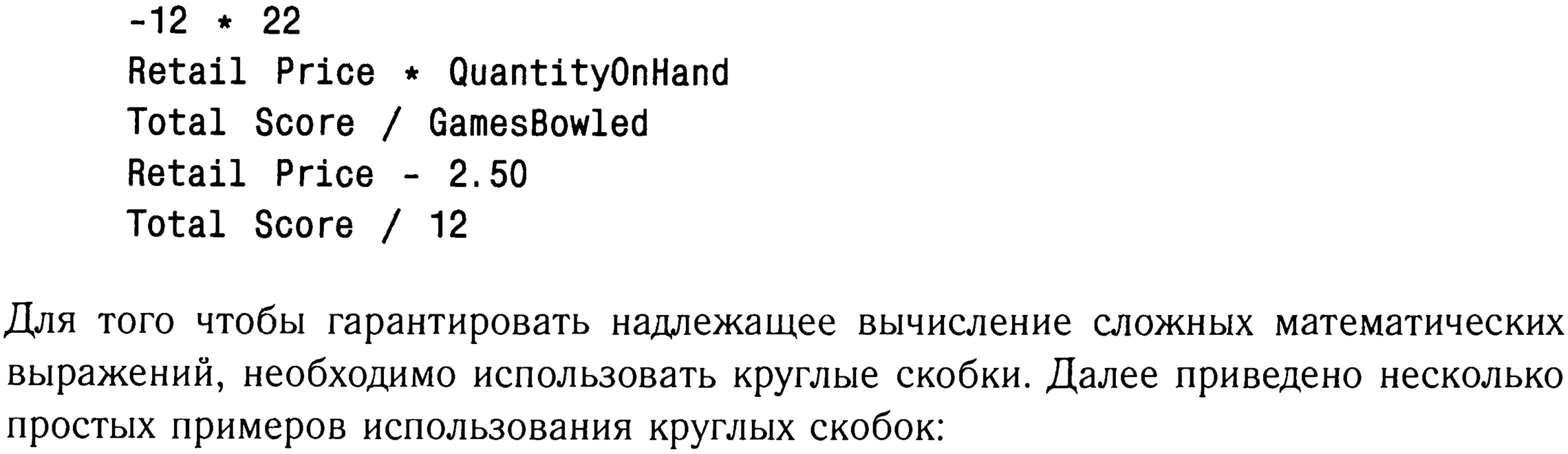


Рис. 5.7. *Синтаксическая диаграмма для математического выражения*

Уделяйте серьезное внимание порядку скобок в выражениях, потому что он влияет на возвращаемое значение. Хотя оба выражения содержат те же самые числа и операторы, при разном порядке расположения скобок возвращаются совершенно разные значения:

Выражение	`(23 * 11) + 12`
Результат	**265**

Выражение	`23 * (11 + 12)`
Результат	**529**

Легко понять, почему со скобками необходимо быть осторожным, но совсем отказываться от них тоже нельзя. Они могут быть неоценимы при работе со сложными выражениями.

Скобки также можно использовать как способ вложения операций в выражении. Вложенные операции, заключенные в скобки, СУБД раскрывает по принципу "от самых внутренних к самым внешним". Приведем пример выражения, которое содержит операции, вложенные с помощью скобок:

Выражение	`(12 * (3 + 4)) - (24 / (10 + (6 - 4)))`
Результат	**82**

Выполнение операций в выражении в действительности не настолько сложно, как кажется. Приведем пример того, как СУБД вычисляет выражение:

1. (3 + 4) = **7**
2. (6 – 4) = **2**
3. (10 + 2) = **12** *10 плюс результат второй операции*
4. (12 * 7) = **84** *12 умножить на результат первой операции*
5. (24/12) = **2** *24 разделить на результат третьей операции*
6. 84 – 2 = **82** *84 минус результат пятой операции*

В этом примере использовались цифровые литералы, но так же просто можно было использовать ссылки на столбцы или комбинацию цифровых литералов и ссылок на столбец. Главное, что необходимо усвоить: тщательно планируйте и определяйте математические выражения, чтобы они возвращали нужные результаты.

При работе с математическими выражениями убедитесь, что значения, используемые в выражении, совместимы. Это особенно справедливо для выражений, содержащих ссылки на столбцы. С этой целью можно использовать функцию CAST, как это делалось в выражениях со сцеплением. Например, имеется столбец с именем TotalLength (Общая длина) и типом данных INTEGER, в котором хранятся целые

числа, и столбец с именем Distance (Расстояние) и типом данных REAL, в котором хранятся числа с дробной частью. Чтобы сложить значение столбца Distance со значением столбца TotalLength, необходимо воспользоваться функцией CAST для преобразования значения столбца Distance в данные типа INTEGER или значения столбца TotalLength в данные типа REAL. Ниже приведено выражение, которое используется для выполнения этого:

Выражение	`TotalLength + CAST(Distance AS INTEGER)`
Результат	483

Если совместимость значений столбцов, используемых в выражении, не обеспечена, СУБД выдаст сообщение об ошибке и, возможно, также отменит выполнение операций в выражении. Большинство СУБД могут сообщить, что проблема связана с несоответствием типов данных и, следовательно, будет известно, что необходимо сделать для исправления выражения. Многие СУБД выполняют такие преобразования автоматически, не предупреждая. Но обычно они перед вычислением выражения преобразуют все числа к наиболее сложному типу данных. В предыдущем примере СУБД должна преобразовать TotalLength в REAL. REAL является более сложным типом данных, чем INTEGER, потому что все значения INTEGER могут содержаться в типе данных REAL. Однако, возможно, это не то, что требуется.

Создание математических выражений будет относительно легкой задачей, если уделить немного времени планированию и знать, как использовать функцию CAST для своих целей.

Арифметика дат и времени

Стандарт SQL определяет сложение и вычитание как операции, которые можно выполнять над датами и временем. Вопреки возможным ожиданиям, многие СУБД различаются по способу реализации этих операций. Некоторые системы баз данных позволяют определять эти операции подобно математическим выражениям, тогда как другие требуют для этих задач использования специальных “встроенных” функций. Чтобы уточнить, как эти операции выполняет конкретная СУБД, обратитесь к своей документации по системе базы данных. В данной книге приводится только общее обсуждение выражений с использованием даты и времени, чтобы дать общее представление о том, как эти операции должны работать.

Выражение с датой

На рис. 5.8 представлен синтаксис для выражений с датой. Как можно видеть, создавать выражения достаточно просто: берем одно значение и складываем его или вычитаем из второго значения. Однако при создании выражений с использованием даты не забывайте о некоторых моментах.

При использовании ссылки на столбец убедитесь в том, что его тип данных — DATE или целый цифровой (к этой категории относятся INTEGER и SMALLINT). В противном случае, возможно, следует воспользоваться функцией CAST для преобразования значений столбца в данные типа DATE. В стандарте SQL отсутствуют

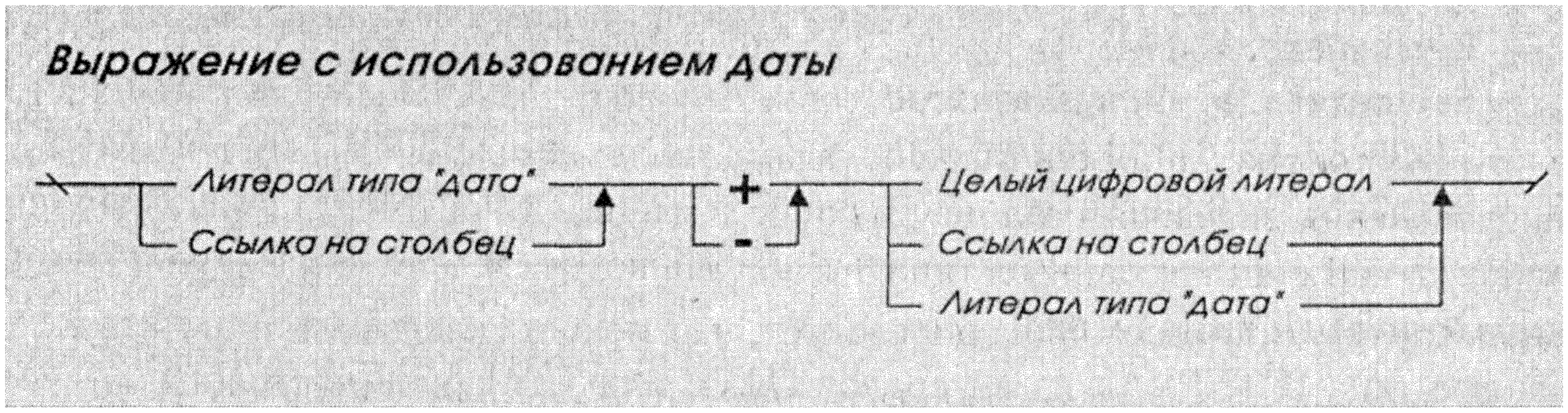


Рис. 5.8. *Синтаксическая диаграмма для выражения с использованием даты*

требования к такому преобразованию, и данный вопрос остается целиком на усмотрение поставщиков БД. Вследствие этого некоторые системы БД преобразуют значения столбца автоматически, тогда как другие требуют явного преобразования типа столбца.

Стандарт SQL позволяет взять целое цифровое значение и прибавлять его или вычитать из даты. Можно рассматривать это как добавление или вычитание дней. Таким способом можно ответить на вопросы типа "Какая дата будет спустя девять дней?" или "Какая дата была несколько дней назад?" В данном случае совершенно не имеет смысла использование чисел с десятичной дробной частью — какой может быть дата через 3,5 дня?

Необходимо принять во внимание еще один момент. Можно вычесть одну дату из другой, но невозможно *сложить* две даты, и это совершенно логично. В качестве примера возьмем базу данных персонала. Пусть нужно вычесть дату приема на работу из текущей даты, чтобы определить, как долго сотрудник работает в компании, но нужно было бы *добавить* некоторое число к текущей дате для извлечения даты следующей проверки сотрудника.

Способ определения выражения с использованием даты установит, будет ли получена в результате дата или целое цифровое значение.

В конечном итоге для выражений с датой справедливо следующее:

Date (Literal or Column) ± Non-Decimal Numeric (Literal or Column) = Date
(Дата (Литерал или столбец) ± Целое цифровое значение (Литерал или столбец) = Дата)

Date (Literal or Column) – Date (Literal or Column) =
Non-Decimal Numeric Value
(Дата (Литерал или столбец) – Дата (Литерал или столбец) =
Целое цифровое значение)

Как только это простая концепция станет понятной, можно создавать любые необходимые выражения с датой. Приведем некоторые примеры типов выражений с датой, которые можно определить:

```
'1999-05-16' - 5
'1999-11-14' + 12
```

```
ReviewDate + 90
EstimateDate - DaysRequired
'1999-07-22' - '1999-06-13'
ShipDate - OrderDate
```

Внимание! Стандарт SQL определяет, что можно складывать и вычитать данные типа INTERVAL из литералов DATE или TIME или из столбца, содержащего значение DATE или TIME. Однако большинство реализаций не поддерживает тип данных INTERVAL, определенный стандартом SQL, но позволяют выполнять сложение или вычитание двух значений типа DATE или TIME. При вычитании одного значения типа DATE или TIME из другого получается "интервал" между двумя датами или моментами времени, но типа данных DATE или TIME. При сложении одного значения DATE или TIME с другим, получим другое значение DATE или TIME. Во всех примерах данной книги предполагается, что можно выполнять как сложение, так и вычитание значений типа DATE или TIME. Уточните в документации для базы данных, как ваша СУБД обрабатывает такие выражения.

Выражения с использованием времени

Можно создавать выражения, используя также значения времени. На рис. 5.9 представлена синтаксическая структура для его использования. Выражения для даты и времени очень похожи, и те же самые правила и ограничения, которые применяются к выражению с использованием даты, применяются к выражению с использованием времени.

Выражение с использованием времени возвращает либо значение времени, либо целое цифровое значение, в зависимости от того, как определено само выражение. Можно обобщить это для выражения с использованием времени таким образом:

Время (Литерал или столбец) ± Целое цифровое значение (Литерал или столбец) = Время

Время (Литерал или столбец) – Время (Литерал или столбец) = Целое цифровое значение

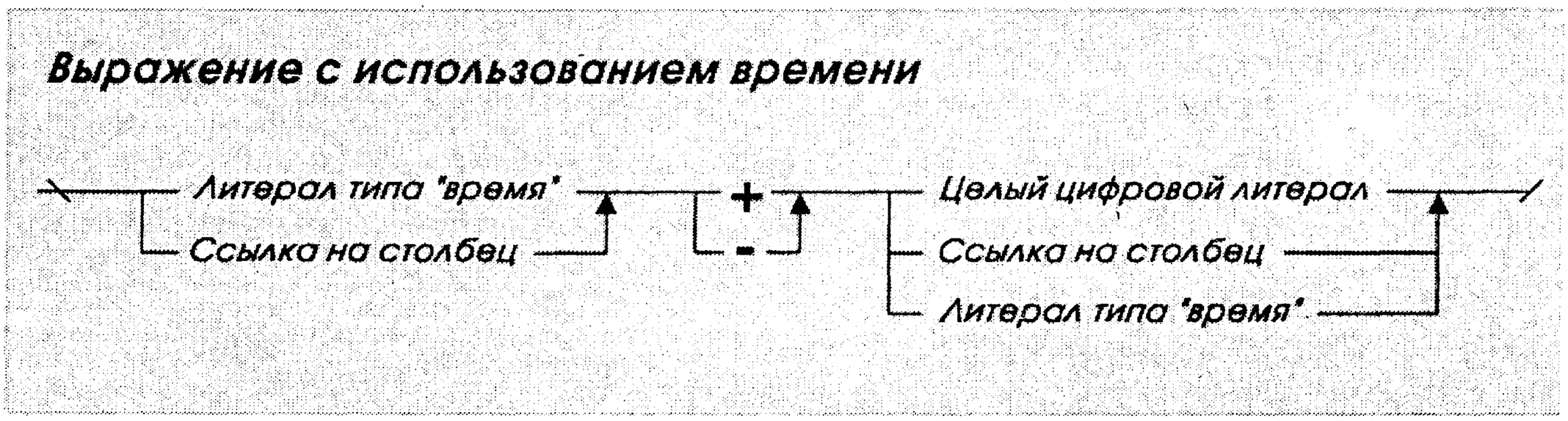


Рис. 5.9. *Синтаксическая диаграмма для выражения с использованием времени*

Представим несколько общих примеров выражений с использованием времени.

```
'14:00' + '00:22'
'19:00' - '16:30'
StartTime + '00:19'
StopTime - StartTime
```

Выражения с использованием даты и времени представлены только для общего сведения. Наша цель состоит только в том, чтобы вы концептуально понимали выражения с использованием даты и времени и имели общее представление о типах выражений, которые вы можете создавать. Конечно, досадно, что большинство СУБД совершенно не реализуют спецификации стандарта SQL для выражений с использованием времени, а многие только частично поддерживают эти спецификации для выражений с использованием даты. Однако все системы баз данных обеспечивают одну или несколько функций, которые позволяют работать с датами и временем, и поэтому изучите документацию по вашей системе базы данных, чтобы узнать, какие типы функций она предоставляет. Хотя синтаксис для этих функций будет немного отличаться от представленного здесь, общие концепции, относящиеся к арифметике для даты и времени, *будут* применимы.

Использование выражений в условии SELECT

Очень важно понимать, как использовать выражения. При работе с SQL выражения используются для множества целей, например для:

- Создания вычисляемого столбца в запросе
- Поиска конкретного значения столбца
- Фильтрации строк в наборе результата
- Подсоединения к двум таблицам в операции JOIN

На протяжении всей книги мы будем объяснять, как используются выражения. Начнем с использования основных выражений в условии SELECT.

> **Внимание!** В данной главе будет использован метод "Запрос/Преобразование/Уточнение/SQL", введенный в главе 4.

В условии SELECT основные выражения можно использовать для уточнения информации в наборе результатов и для расширения объема этой информации. Например, можно создать выражения для объединения фамилии и имени, вычислить итоговую цену продукта, определить, сколько времени потребуется для завершения проекта, или выяснить дату следующего приема для пациента. Посмотрим, как можно использовать выражение со сцеплением, математическое выражение и выражение с использованием даты в условии SELECT.

Использование выражения со сцеплением

В отличие математических выражений и выражений с датой, выражения со сцеплением используются только для повышения удобочитаемости информации, содержащейся в наборе результатов оператора SELECT. Предположим, что делается следующий запрос:

> *"Show me a current list of our employees and their phone numbers".*
> *("Показать текущий список наших сотрудников*
> *и их телефонные номера".)*

При преобразовании этого запроса в операторе SELECT можно до некоторой степени улучшить вывод набора результата, объединив имя и фамилию в один столбец. Ниже приводится один способ преобразования этого запроса:

Преобразование: Select the first name, last name, and phone number
of all our employees from the employees table
(Выбрать имя, фамилию и номер телефона
всех сотрудников из таблицы "Сотрудники")

Уточнение: Select ~~the~~ first name, last name, ~~and~~ phone number
~~of all our employees~~ from ~~the~~ employees ~~table~~
(Выбрать имя, фамилию, номер телефона
из "Сотрудники")

SQL

```
SELECT EmpFirstName || ' ' || EmpLastName,
    'Phone Number: ' || EmpPhoneNumber
FROM Employees
```

Обратите внимание, что, помимо сцепления столбца имени, пробела и столбца фамилии, мы также объединили строку с символьным литералом "Номер телефона:" со столбцом телефонного номера. Этот пример показывает, что можно легко использовать более одного выражения конкатенации в условии SELECT для повышения удобочитаемости информации в наборе результата. Так же можно объединять значения с различными типами данных, применяя функцию CAST. По существу, в следующем примере объединяются значение символьного столбца со значением цифрового столбца:

> *"Shaw me a list f all our vendors and their identification numbers".*
> *("Показать список всех наших поставщиков и их*
> *идентификационные номера".)*

Преобразование: Select the vendor name and vendor ID
from the vendors table
(Выбрать имя поставщика и идентификатор поставщика
из таблицы "Поставщики")

Уточнение: Select ~~the~~ vendor name ~~and~~ vendor ID
from ~~the~~ vendors ~~table~~
(Выбрать имя поставщика, идентификатор поставщика
из “Поставщики”)

```
SQL     SELECT “The ID Number for ” || VendName ||
            ‘ is ‘ || CAST(VendorID AS CHARACTER)
        FROM Vendors
```

Хотя выражение со сцеплением является важным инструментом в операторе SELECT, применять его следует с осторожностью. При использовании выражения со сцеплением, содержащего длинные литералы с символьными строками, не забывайте, что литералы будут появляться в каждой строке набора результатов. Можно завалить окончательный результат повторяющейся информацией вместо его улучшения. Тщательно изучите использование литералов в выражениях со сцеплением, чтобы они работали вам во благо.

Присвоение имени выражению

При использовании выражения в условии SELECT набор результата включает новый столбец, который отражает результат операции, определенной в выражении. Этот новый столбец называется вычисляемым (или производным) столбцом. Например, набор результатов для следующего оператора SELECT будет содержать три столбца: два “настоящих” и один вычисляемый:

```
SQL     SELECT EmpFirstName || ‘ ‘ || EmpLastName,
            EmpPhoneNumber, EmpCity
        FROM Employees
```

Два “настоящих” столбца это, конечно, EmpPhoneNumber и EmpCity, а вычисляемый столбец получается из выражения со сцеплением в начале условия SELECT.

В соответствии со стандартом SQL можно (но не обязательно) присвоить имя новому столбцу, используя ключевое слово AS. Однако почти каждая СУБД *требует* имени для вычисляемого столбца. Некоторые системы баз данных требуют явного предоставления имени, а другие фактически сами предоставляют его. Определите, как ваша система баз данных обрабатывает их, прежде чем перейдете к работе с примерами.

На рис. 5.10 представлен синтаксис для определения имени выражения. Для имени можно использовать любой допустимый строковый литерал (заключенный в апострофы). Некоторые системы баз данных смягчают это требование, когда присваивается имя выражению, и требуют апострофы только тогда, когда имя столбца

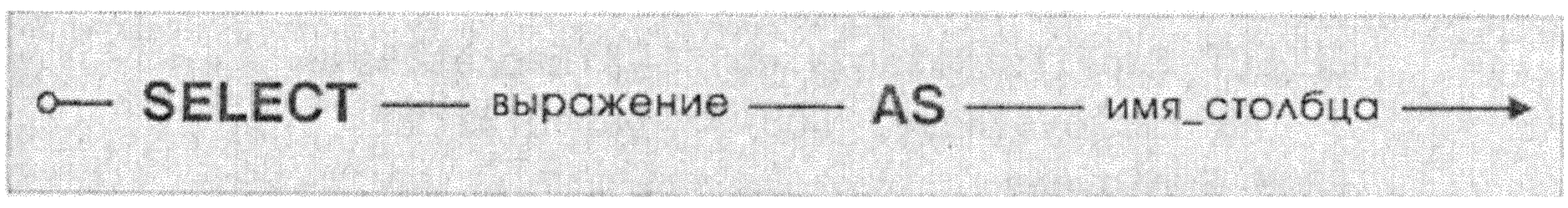


Рис. 5.10. *Присвоение имени выражению*

включает вложенные пробелы. Однако мы настоятельно рекомендуем не использовать пробелы в именах, потому что они могут вызвать проблемы в некоторых языках программирования баз данных.

Изменим оператор SELECT из предыдущего примера и присвоим имя выражению со сцеплением.

```
SQL                SELECT EmpFirstName || ' ' || EmpLastName AS
                       EmployeeName, EmpPhoneNumber, EmpCity
                   FROM Employees
```

Теперь набор результатов для этого оператора SELECT содержит три столбца с именами EmployeeName, EmpPhoneNumber и EmpCity.

Помимо присвоения имени для выражений, ключевое слово AS можно использовать для присвоения "псевдонима" реальному имени столбца. Предположим, имеется столбец с именем DOB и возникает беспокойство о том, все ли ваши пользователи или коллеги по работе знают значения этого имени. Можно исключить любую возможную неверную интерпретацию этого имени, используя алиас:

```
SQL                SELECT EmpFirstName || ' ' || EmpLastName AS
                       EmployeeName, DOB AS DateOfBirth
                   FROM Employees
```

Этот оператор SELECT создает набор результатов с двумя столбцами: EmployeeName и DateOfBirth. Теперь эффективно исключена любая возможная путаница в информации, отображенной в наборе результата.

Присвоение имен вычисляемым столбцам оказывает незначительное воздействие на процесс преобразования. Вот один из вариантов процесса преобразования для предыдущего примера:

"Give me a list of employee names and their dates of birth".
("Предоставить список имен сотрудников и дат их рождения".)

Преобразование: Select first name and last name as EmployeeName
and DOB as DateOfBirth from the employees table
(Выбрать фамилию и имя как EmployeeName
и DOB как DateOfBirth из таблицы "Сотрудники")

Уточнение: Select first name ~~and~~ || ' ' || last name as EmployeeName
and DOB as DateOfBirth from ~~the~~ employees ~~table~~
(Выбрать фамилию || ' ' ||, имя как EmployeeName
и DOB как DateOfBirth из "Сотрудники")

```
SQL                SELECT EmpFirstName || ' ' || EmpLastName
                       AS EmployeeName, DOB AS DateOfBirth
                   FROM Employees
```

Когда вы освоитесь с использованием выражений, вам не потребуется формулировать их в своем преобразуемом утверждении настолько точно, насколько это представлено здесь. По существу, вы сможете легко установить и определить необходимые выражения при построении самого оператора SELECT.

> **Внимание!** В остальной части книги мы присваиваем имена для всех вычисляемых столбцов оператора SQL по обстоятельствам.

Работа с математическими выражениями

Математические выражения, несомненно, наиболее универсальные из трех типов выражений, и вероятно будут использоваться достаточно часто. Их можно использовать для вычисления общего значения элементов строки, для определения среднего значения для указанного набора тестов, для вычисления разности двух результатов лабораторных исследований и оценки общего количества мест в здании. Секрет состоит в том, чтобы убедиться, что ваше выражение работает и что это просто функция немного более тщательного планирования.

Ниже приведен пример использования математического выражения в операторе SELECT:

"Give me a list of bowler names and their average scores".
("Предоставить список имен игроков в боулинг
и их усредненный счет".)

Преобразование: Select first name and last name as bowler name
and total score divided by games bowled as AverageScore
from the bowlers table
(Выбрать имя и фамилию как имя игрока в боулинг
и общий балл, деленный на количество сыгранных игр,
как AverageScore из таблицы "Игроки в боулинг".)

Уточнение: Select first name and last name as BowlerName
~~and~~ total score ~~divided by~~ /games bowled as AverageScore
from ~~the~~ bowlers ~~table~~
(Выбрать имя, фамилию как BowlerName общий балл /
количество сыгранных игр как AverageScore
из "Игроки в боулинг")

SQL

```
SELECT BowlerFirstName || ' ' || BowlerLastName
    AS BowlerName. TotalScore / GamesBowled
    AS AverageScore
FROM Bowlers
```

Как видно из примера, нет ограничений на использование в операторе SELECT только одного типа выражений. Скорее, можно использовать различные выражения для извлечения необходимой информации в набор результата. Приведем еще один вариант записи предыдущего оператора SQL:

```
SQL                SELECT BowlerFirstName || ' ' || BowlerLastName
                       || ' has an average score of ' ||
                       (CAST(Total Score / GamesBowled AS
                       CHARACTER)) AS BowlerAverages
                   FROM Bowlers
```

Информацию с использованием математических выражений можно представить фактически бесконечным числом способов, но следует надлежащим образом составлять свои выражения и при необходимости использовать функцию CAST.

Работа с выражениями типа "дата"

Использование выражения типа "дата" подобно использованию математических выражений тем, что значения просто добавляется или вычитается. Выражения типа "дата" можно использовать для любых видов задач. Например, можно вычислить предполагаемую дату поставки, запланировать количество дней, которое понадобится для завершения проекта, или определить для пациента дату следующего визита. Ниже приводится пример использования выражения типа "дата" в условии SELECT:

"How many days did it take to ship each order?"
("Сколько дней потребуется для поставки каждого заказа?")

Преобразование: Select the order number and ship date minus order date as DaysToShip from the orders table
(Выбрать номер заказа и дату поставки, вычтя дату заказа, как DaysToShip из таблицы "Заказы")

Уточнение: Select ~~the~~ order number ~~and~~ ship date ~~minus~~ – order date as DaysToShip from ~~the~~ orders ~~table~~
(Выбрать номер заказа, дату поставки, вычтя дату заказа, как дату из "Заказы")

```
SQL                SELECT OrderNumber, ShipDate - OrderDate AS
                       DaysToShip
                   FROM Orders
```

Подобным же образом можно использовать выражения типа "время".

"What would be the start time for each class if we began each class ten minutes later than the current start time?"
("Во сколько начнется каждая лекция, если мы будем начинать каждую лекцию на 10 минут позже, чем сейчас?")

Преобразование: Select the start time and start time plus 10
as NewStartTime from the classes table
(Выбрать время начала и время начала плюс 10 минут
как NewStartTime из таблицы “Расписание лекций”)

Уточнение: Select ~~the~~ start time ~~and~~ start time ~~plus~~ + 10
as NewStartTime from ~~the~~ classes ~~table~~
(Выбрать время начала, время начала + 10
как NewStartTime из “Расписание лекций”)

```
SQL                SELECT StartTime, StartTime + '00:10'
                       AS NewStartTime
                   FROM Classes
```

Все системы баз данных предусматривают функцию или ряд функций для работы с псевдонимами. Однако нам хотелось дать вам представление о том, как можно использовать даты и время в операторах SELECT. Снова рекомендуем обратится к документации по своей БД, чтобы уточнить, какие функции работы с данными и временем в ней предоставляются.

Краткое отступление: Типизированное выражение

Теперь вы знаете, как использовать ссылки на столбцы, значения типа “литерал” и выражения в условии SELECT. Вам также известно, как присвоить имя ссылке на столбец или выражению. Теперь мы покажем, как это все это состыковывается с более крупными деталями.

Для ссылок на столбец, на значения типа “литерал” и выражения в стандарте SQL используется термин *“типизированное выражение”* (value expression). На рис. 5.11 показано, как определяется типизированное выражение.

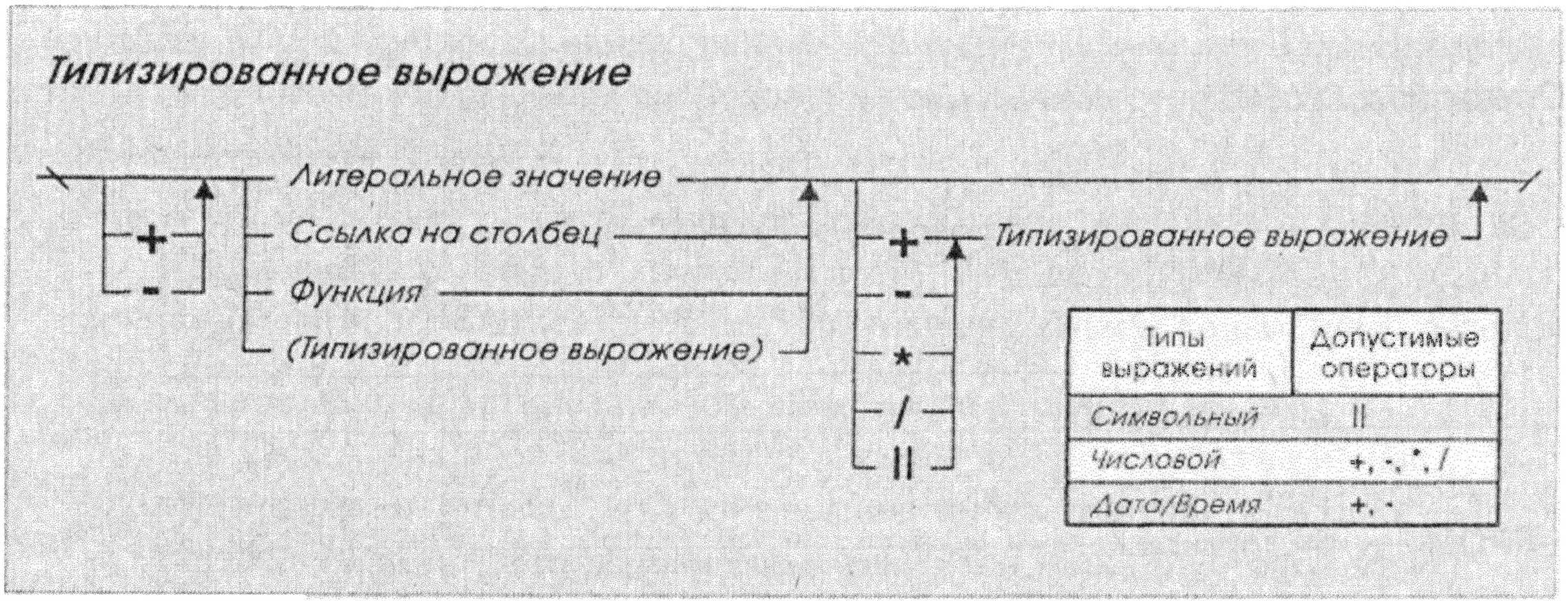


Рис. 5.11. *Диаграмма для типизированного выражения*

Рассмотрим более подробно компоненты типизированного выражения:

- Синтаксическая структура начинается с необязательного знака плюс или минус. Любой из этих знаков используется, если необходимо, чтобы типизированное выражение возвратило числовое значение со знаком. Само значение может быть числовым литералом, значением числового столбца, вызовом функции, возвращающей числовое значение (о функции CAST см. выше), или значение, возвращаемое математическим выражением. Знак плюс или минус нельзя использовать перед выражением, которое возвращает данные символьного типа.
- Первый список также включает (Типизированное выражение). Это означает, что можно использовать сложные типизированные выражения, составленные из других типизированных выражений, которые сами включают в себя конкатенацию или математические операторы. Скобки заставляют систему БД вначале вычислить это типизированное выражение (о порядке выполнения операций см. в следующей главе).
- "Типизированное выражение" также появляется после списка операторов. Возможность использования других типизированных выражений внутри типизированного выражения позволяет создавать очень сложные выражения.
- Следующий элемент синтаксиса является списком операторов. Как видно на блоке-врезке, тип выражения, используемый в начале синтаксической структуры, определяет, какие операторы можно выбрать из этого списка.

По своему определению типизированное выражение возвращает значение, которое используется некоторыми компонентами оператора SQL. Стандарт SQL определяет использование типизированного выражения в разнообразных операторах и определяемых элементах. Независимо от того, где оно используется, типизированное выражение всегда будет определяться таким же образом, как показано здесь.

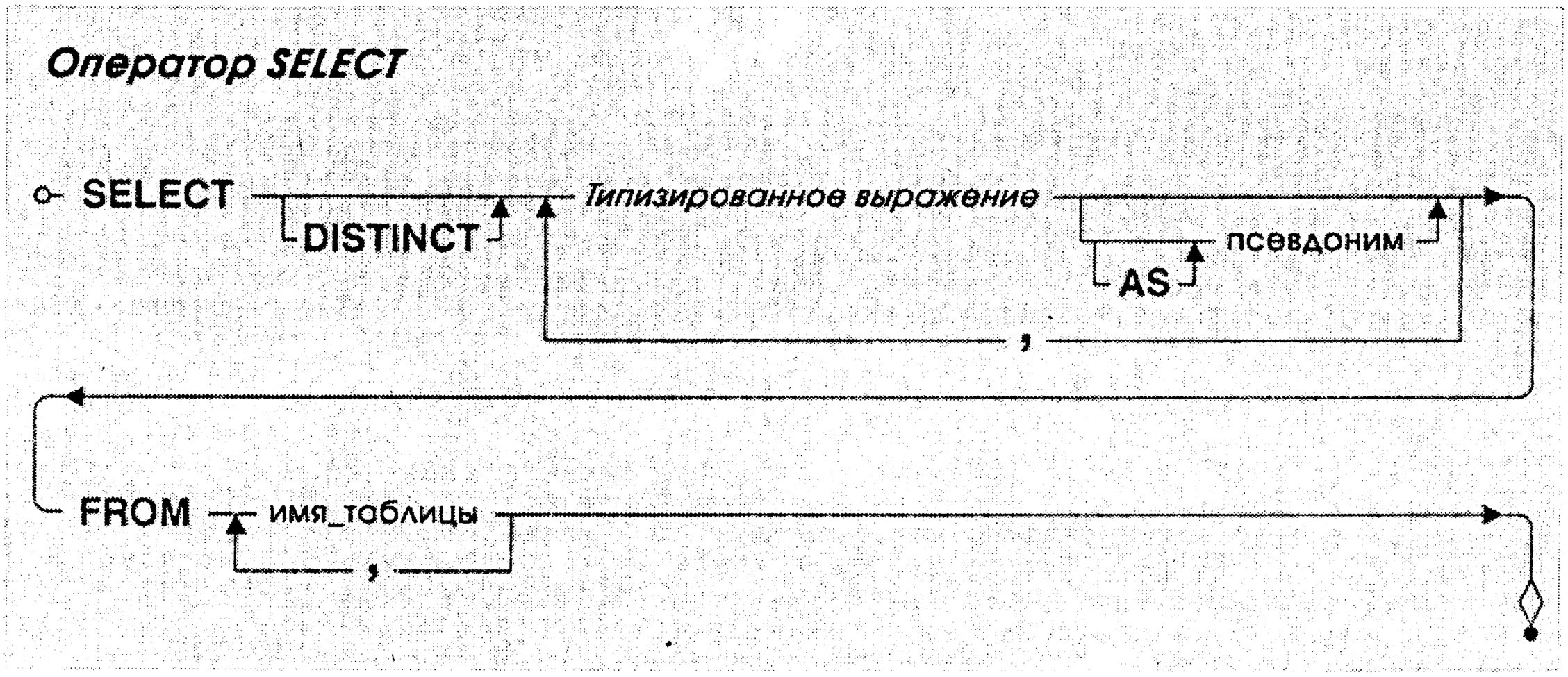


Рис. 5.12. *Синтаксическая диаграмма для оператора SELECT*

Посмотрите, как используется типизированное выражение в операторе SELECT. На рис. 5.12 представлен измененный вариант окончательной синтаксической диаграммы оператора SELECT, приведенной в главе 4. Эта новая синтаксическая структура предоставляет гибкость использования литералов, ссылок на столбцы, выражений или любых их комбинаций в рамках отдельного оператора SELECT. При необходимости можно присвоить имя своим типизированным выражениям в ключевом слове AS.

В остальной части книги термин "типизированное выражение" используется для указания ссылки на столбец, значение литерала или выражение — в зависимости от обстоятельств. В последних главах обсуждается использование типизированного выражения в других операторах и показана пара других элементов, которые представляет типизированное выражение.

Значение Null

Как известно, таблица состоит из столбцов и строк. Каждый столбец представляет характеристику предмета таблицы, а каждая строка представляет уникальный экземпляр предмета таблицы. Строку также можно представить как полный набор значений столбцов — каждая строка содержит в точности одно значение из каждого столбца таблицы. На рис. 5.13 приведен пример типичной таблицы.

В предыдущих примерах предполагалось, что каждый столбец в таблице содержит данные. Но, как ясно показано на рис. 5.13, столбец иногда может не содержать значения для конкретной строки в таблице. В зависимости от использования данных отсутствие значения может оказывать отрицательное влияние на операторы SELECT и типизированные выражения. Прежде чем мы обсудим последствия, посмотрим сначала, как SQL обрабатывает пропущенные значения.

Понятие о Null

В SQL *пропущенное* или *неизвестное* значение обозначается специальным значением *Null*. Объясним с самого начала, что Null — это *не* нуль, *не* символьная строка из одного или нескольких пробелов и *не* символьная строка "нулевой длины". Причины достаточно просты:

- Нуль может иметь очень широкое разнообразие значений. Он может представлять состояние остатка на счете, текущее количество возможных замен на билеты первого класса или текущие запасы конкретного изделия.
- Хотя можно гарантировать, что символьная строка из одного или нескольких пробелов не имеет смысла для большинства из нас, это все равно "что-то", что определенно имеет значение в SQL. Пробел является допустимым символом в той мере, в которой это касается SQL, а символьная строка, составленная из трех пробелов (' ') настолько же законна (допустима), насколько допустима символьная строка, состоящая из трех букв ('abc').

Customers

CustomerID	CustFirstName	CustLastName	CustAddress	CustCity	CustCounty	CustState
1001	Suzanne	Viescas	15127 NE 24th, #383	Redmond	King	WA
1002	Wil	Thompson	122 Spring River Drive	Duvall	King	WA
1003	Gary	Hallmark	Route 2, Box 203B	El Paso	El Paso	TX
1004	Michael	Davolio	672 Lamont Ave	Marysville		WA
1005	Kenneth	Peacock	4110 Old Redmond Rd.	Fremont		CA
1006	John	Viescas	15127 NE 24th, #383	Redmond	King	WA
1007	Laura	Callahan	901 Pine Avenue	Washington		DC
1008	Neil	Patterson	233 West Valley Hwy	Everett	Snohomish	WA

Рис. 5.13. *Обычная таблица "Клиенты"*

- Строка нулевой длины — две последовательные одиночные кавычки без пробела между ними (‘ ’) — может при определенных обстоятельствах оказаться значащей. Например, в таблице Employee значение строки с нулевой длиной в столбце с именем MiddleInitial указывает, что у конкретного служащего отсутствует второе имя.

Значение Null исключительно полезно, когда используется для указанной цели, и таблица Customers на рис. 5.13 представляет собой явный пример. В столбце CustCounty каждое Null представляет собой пропущенное или неизвестное имя округа для строки, в которой оно появилось. Для корректного использования Null необходимо понять в первую очередь, почему оно появляется.

Пропущенные значения обычно являются результатом ошибки человека. Рассмотрим, например, строку для Michael Davolio. Если при вводе данных для г-на Davolio вы не спросили у него название страны проживания, то эти данные будут рассматриваться как пропущенные и представляются в строке как Null. Когда ошибка будет обнаружена, ее можно исправить, уточнив у г-на Davolio название его страны.

Неизвестные значения появляются в таблице по множеству причин. Одна из них может быть в том, что необходимое для столбца конкретное значение все еще не определено. Допустим, имеется таблица Categories (Категории) в базе данных School Scheduling (Расписание занятий). В этой таблице отсутствует категория для нового курса лекций, который предполагается ввести начиная с осенней сессии. Другая причина, по которой в таблице могут содержаться неизвестные значения, состоит в том, что они действительно неизвестны. Воспользуемся еще раз таблицей Customers (Клиенты) на рис. 5.13 и рассмотрим строку для Kenneth Peacock. Предположим, что вводятся данные для г-на Peacock и вы запрашиваете у него название округа, в котором он живет. Если он не знает его названия, значение столбца для округа в его строке будет действительно неизвестным и представляется в этой строке как Null. Можно устранить проблему, как только название округа будет установлено.

Значение столбца также может быть Null, если ни одно из его значений неприменимо к конкретной строке. Предположим, вы работаете с таблицей Employee

(Сотрудники), которая содержит столбцы Salary (Зарплата) и HourlyRate (Почасовая ставка). Значение одного из этих столбцов всегда будет Null, поскольку невозможна одновременная оплата сотрудника как по фиксированной, так и по почасовой ставке.

Между понятиями "не применяется" и "неприменимо" существует очень незначительная разница. В предыдущем примере значение одного из столбцов действительно не применяется. Но предположим, что вы работаете с таблицей Patient (Пациенты), в которой имеется столбец HairColor (Цвет волос), и вы занимаетесь текущей корректировкой строки для пациента-мужчины. Если этот пациент лысый, то значение для указанного столбца определенно неприменимо к нему. Хотя и можно использовать значение Null для представления неприменимого значения, мы рекомендуем использовать истинное значение, например "N/A" (от Not Applicable — Неприменимо). При этом информация окажется намного понятнее спустя некоторое время.

Разрешать ли использование значения Null в таблице — зависит от способа использования данных. Рассмотрим теперь негативные последствия использования значений Null.

Проблемы, связанные с использованием Null

Основной недостаток использования Null состоит в его неблагоприятном воздействии на математические операции. Любая операция с привлечением Null дает в результате Null. Логически это разумно: если число неизвестно, то результат операции неизбежно неизвестен. Обратите внимание, как Null изменяет результат операции в следующем примере:

```
(25 * 3) + 4 = 79
(Null * 3) + 4 = Null
(25 * Null) + 4 = Null
(25 * 3) + Null = Null
```

Такой же результат получается, когда в операции используются столбцы, содержащие значения Null. Предположим, что выполняется следующий оператор SELECT и возвращает набор результатов, представленный на рис. 5.14:

```
SQL     SELECT ProductID, ProductDescription, Category,
            Price, QuantityOnHand, Price *
            QuantityOnHand AS Total Value
        FROM Products
```

Операция, представленная столбцом TotalValue, завершается успешно в том случае, когда столбцы Price и QuantityOnHand являются допустимыми цифровыми значениями. Если Price либо QuantityOnHand содержат Null, то TotalValue будет содержать Null. Как только значения Null в Price и QuantityOnHand будут заменены действительными цифровыми значениями, TotalValue будет содержать соответствующее

Products

ProductID	ProductDescription	Category	Price	QuantityOnHand	TotalValue
70001	Shur-Lok U-Lock	Accessories		12	
70002	SpeedRite Cyclecomputer		65.00	20	1.300.00
70003	SteelHead Microshell Helmet	Accessories	36.00	33	1.118.00
70004	SureStop 133-MB Brakes	Components	23.50	16	376.00
70005	Diablo ATM Mountain Bike	Bikes	1.200.00		
70006	UltraVision Helmet Mount Mirrors		7.45	10	74.50

Рис. 5.14. *Результат присутствия Null в математическом выражении*

значение. Этой проблемы можно полностью избежать, обеспечив отсутствие значений Null в столбцах, используемых в математическом выражении.

В главе 12 будет рассмотрено влияние значений Null на операторы SELECT, суммирующие информацию.

Примеры операторов

Теперь рассмотрим некоторые примеры использования таблиц для каждой из учебных баз данных.

> **Внимание!** В последующих примерах этапы преобразования и уточнения объединены, чтобы вы научились делать это самостоятельно. Хотя все еще выполняются все три этапа, вам предоставляется возможность поработать с процессом объединения в примерах операторов.

База данных заказов на закупку

"What is the inventory value of each product?"
("Какова стоимость запасов каждого товара?")

Преобразование/ Уточнение:

Select ~~the~~ product name, retail price ~~times~~ * quantity on hand as InventoryValue from ~~the~~ products ~~table~~
(Выбрать наименование продукта, розничную цену * количество в наличии как InventoryValue из "Товары")

SQL

```
SELECT ProductName,
    Retail Price * QuantityOnHand
    AS InventoryValue
FROM Products
```

Product_Inventory_Value (40 строк)

ProductName	InventoryValue
Trek 9000 Mountain Bike	$7,200.00
Eagle FS-3 Mountain Bike	$14,400.00
Dog Ear Cyclecomputer	$1,500.00
Victoria Pro All Weather Tires	
Dog Ear Helmet Mount Mirrors	$89.40
Viscount Mountain Bike	$3,175.00
<< остальные строки >>	

"How many days elapsed between the order date and the ship date for each order?"
("Сколько дней прошло от даты заказа до даты поставки каждого заказа?")

Преобразование/ Уточнение:
Select ~~the~~ order number, order date, ship date, ship date ~~minus~~ – order date as DaysElapsed from ~~the~~ orders ~~table~~
(Выбрать номер заказа, дату заказа, дату поставки, дату поставки – дата заказа как DaysElapsed из "Заказы")

SQL

```
SELECT OrderNumber, OrderDate, ShipDate,
    ShipDate - OrderDate AS DaysElapsed
FROM Orders
```

Shipping_Days_Analysis (944 строки)

OrderNumber	OrderDate	ShipDate	DaysElapsed
1	1999-07-01	1999-07-04	3
2	1999-07-01	1999-07-03	2
3	1999-07-01	1999-07-04	3
4	1999-07-01	1999-07-03	2
5	1999-07-01	1999-07-01	0
6	1999-07-01	1999-07-05	4
7	1999-07-01	1999-07-04	3
8	1999-07-01	1999-07-01	0
9	1999-07-01	1999-07-04	3
<< остальные строки >>			

База данных эстрадных мероприятий

"How long is each engagement due to run?"
("Сколько должен продолжаться каждый ангажемент?")

Преобразование/ Уточнение:

Select ~~the~~ engagement number, end date ~~minus~~ – start date ~~plus one~~ 1 as DueToRun from ~~the~~ engagements ~~table~~
(Выбрать номер ангажемента, дату окончания – дата начала плюс 1 как DueToRun из "Эстрадные мероприятия")

SQL

```
SELECT EngagementNumber, CAST(EndDate -
    StartDate + 1 AS CHARACTER) || ' day(s)'
    AS DueToRun
FROM Engagements
```

Engagement_Lengths (131 строка)

EngagementNumber	DueToRun
1	4 day(s)
2	5 day(s)
3	6 day(s)
4	7 day(s)
5	4 day(s)
6	5 day(s)
7	8 day(s)
8	8 day(s)
9	11 day(s)
10	10 day(s)
<< остальные строки >>	

Внимание! Нужно добавить "1" к выражению с датой, чтобы принять во внимание все даты ангажемента. Иначе вы получите "0 дней" для ангажемента, который начинается и заканчивается в тот же день.

"What is the net amount for each of our contracts?"
("Какова величина чистой выручки для каждого нашего контракта?")

Преобразование/ Уточнение:

Select ~~the~~ engagement number, contract price, contract price ~~times~~ * 0.12 as OurFee, contract price ~~minus~~ – (contract price ~~times~~ * 0.12) as NetAmount
from ~~the~~ engagements ~~table~~
(Выбрать номер ангажемента, стоимость контракта, стоимость контракта * 0.12 как OurFee, стоимость контракта – (стоимость контракта * 0.12) как NetAmount из "Ангажементы")

SQL

```
SELECT EngagementNumber, ContractPrice,
    ContractPrice * 0.12 AS OurFee,
    ContractPrice - (ContractPrice * 0.12)
    AS NetAmount
FROM Engagements
```

Net_Amount_Per_Contract (131 строка)

EngagementNumber	ContractPrice	OurFee	NetAmount
1	$170.00	$20.40	$149.60
2	$200.00	$24.00	$176.00
3	$590.00	$70.80	$519.20
4	$470.00	$56.40	$413.60
5	$1,130.00	$135.60	$994.40
6	$2,300.00	$276.00	$2,024.00
7	$770.00	$92.40	$677.60
8	$1,850.00	$222.00	$1,628.00
9	$1,370.00	$164.40	$1,205.60
10	$3,650.00	$438.00	$3,212.00
<< остальные строки >>			

База данных расписания занятий

"How many years has each staff member been with the school?"
("Сколько лет проработал каждый штатный сотрудник в школе?")

Преобразование/ Уточнение:

Select last name || ',' || ~~and~~ first name ~~concatenated with a comma~~ as Staff, date hired, ~~and~~ (('199-10-01' ~~minus~~ – date hired)) ~~divided by~~ / 365) as YearsWithSchool from ~~the~~ staff ~~table~~
(Выбрать фамилию || ',' || имя как Staff, дату приема на работу (('199-10-01' – дата приема на работу) / 365) как YearsWithSchool из "Персонал")

SQL

```
SELECT StfLastName || ', ' || StfFirstName
   AS Staff, DateHired, CAST(('199-10-01' -
   DateHired) / 365) AS INTEGER) AS YearsWithSchool
FROM Staff
ORDER BY StfLastName, StfFirstName
```

Length_Of_Service (27 строк)

Staff	DateHired	YearsWithSchool
Black, Alastair	1988-12-11	10
Bonnicksen, Joyce	1986-03-02	13
Buchanan, Albert	1985-08-02	14
Buchanan, Amelia	1988-05-31	11
Callahan, David	1987-01-13	12
Callahan, Laura	1989-11-02	9
Coie, Caroline	1983-01-28	16
Davis, Allan	1989-08-20	10
Davolio, Michael	1989-02-09	10
Ehrlich, Katherine	1985-03-08	14
<< остальные строки >>		

Внимание! В этом операторе SELECT выражение технически верно и работает, как предполагалось, но оно возвращает неверный ответ для любого високосного года. Эту проблему можно скорректировать, используя соответствующую функцию для арифметики дат, предоставляемую вашей системой базы данных. Большинство СУБД предоставляет свои собственные методы работы с датами и временем.

"Show me a list of staff members, their salaries, and a proposed 7% bonus for each staff member?"
("Показать список преподавателей, их зарплату и сумму предлагаемой 7-процентной премии")

Преобразование/ Уточнение:
Select ~~the~~ last name || ',' || ~~and~~ first name as StaffMember, salary, ~~and~~ salary ~~times~~ * 0.17 as Bonus from ~~the~~ staff ~~table~~
(Выбрать фамилию || ',' ||, имя как StaffMember, зарплату, зарплату * 0.07 как Bonus из "Персонал")

SQL

```
SELECT StfLastName || ', ' || StfFirstName
    AS Staff, Salary, Salary * 0.07 AS Bonus
FROM Staff
```

Proposed_Bonuses (27 строк)

Staff	Salary	Bonus
Black, Alastair	$60,000.00	$4,200.00
Bonnicksen, Joyce	$60,000.00	$4,200.00
Buchanan, Albert	$45,000.00	$3,150.00
Buchanan, Amelia	$48,000.00	$3,360.00
Callahan, David	$50,000.00	$3,500.00
Callahan, Laura	$45,000.00	$3,150.00
Coie, Caroline	$52,000.00	$3,640.00
Davis, Allan	$56,000.00	$3,920.00
Davolio, Michael	$49,000.00	$3,430.00
Ehrlich, Katherine	$45,000.00	$3,150.00
<< остальные строки >>		

База данных игроков в боулинг

"What was each bowler's monthly average score for each of the four months in the tournament?"
("Какой средний счет каждого игрока в боулинг за каждый из четырех месяцев турнира?")

Преобразование/ Уточнение:
Select last name || ',' || ~~and~~ first name ~~concatenated with a comma~~ as Bowler, total score, total score ~~divided by~~ / 4 as AverageScorePerMonth from ~~the~~ bowlers ~~table~~ ~~and~~ order by bowler

(Выбрать фамилию || ‘,’ ||, имя как Bowler, общий балл,
общий балл / 4 как AverageScorePerMonth
из “Игроки в боулинг” и упорядочить по игрокам)

SQL

```
SELECT Bowler LastName || ‘, ‘ ||
   BowlerFirstName AS Bowler, Total Score,
   TotalScore/4 AS AverageScorePerMonth
FROM Bowlers
ORDER BY Bowler
```

Average_Monthly_Score (32 строки)

Bowler	TotalScore	AverageScorePerMonth
Black, Alastair	6319	1579.75
Cunningham, David	6702	1675.5
Ehrlich, Zachary	6208	1552
Fournier, Barbara	6242	1560.5
Fournier, David	6581	1645.25
Hallmark, Alaina	6622	1655.5
Hallmark, Bailey	6291	1572.75
Hallmark, Elizabeth	6379	1594.75
Hallmark, Gary	6593	1648.25
Hernandez, Kendra	6276	1569
<< остальные строки >>		

“What was the point spread between a bowler’s handicap and raw score for each match and game played?”
(“Каков разброс в счете между гандикапом и предварительным счетом для каждого матча и сыгранной игры?)

Преобразование/
Уточнение:

Select bowler ID, match ID, game number, handicap score, raw score, handicap score ~~minus~~ – raw score as PointDifference from ~~the~~ bowler scores ~~table and~~ order by bowler ID, match ID, game number
(Выбрать идентификатор игрока в боулинг, идентификатор матча, номер игры, счет гандикапа, предварительный счет, счет гандикапа – предварительный счет как PointDifference

из “Очки_игрока”, упорядоченные по идентификатору игрока, идентификатору матча, номеру игры)

SQL

```
SELECT BowlerID, MatchID, GameNumber,
    HandiCapScore, RawScore,
    HandiCapScore-RawScore AS PointDifference
FROM Bowler_Scores
ORDER BY BowlerID, MatchID, GameNumber
```

Handicap_vs_RawScore (1344 строки)

BowlerID	MatchID	GameNumber	HandiCapScore	RawScore	PointDifference
1	1	1	192	146	46
1	1	2	192	146	46
1	1	3	199	153	46
1	5	1	192	145	47
1	5	2	184	137	47
1	5	3	199	152	47
1	10	1	189	140	49
1	10	2	186	137	49
1	10	3	210	161	49
<< остальные строки >>					

Итоги

В начале главы мы рассмотрели использование явных значений в операторе SELECT. Вы узнали, что можно использовать символьные строки, цифры, даты и время в условии SELECT и что они все вместе называются литералами. Для расширения или сужения объема информации, извлекаемого из базы данных, используются выражения. В некоторых видах операций выражения включают цифры, символьные строки или значения дат и времени.

Что касается типов данных, то существует семь основных типов данных в трех общих категориях. Каждый тип данных имеет одно или несколько вариантов уникальных наименований.

При рассмотрении выражений мы представили краткий обзор каждого из типов выражений. Мы показали, как объединить строки символов вместе, как объединить строки с другим типом данных с помощью функции CAST, как создавать математические выражения и какое влияние оказывает порядок предшествования на заданные математические операции. Мы также рассмотрели выражения с использованием дат и времени. Большинство СУБД предусматривают свои собственные методы работы с датами и временем.

Выражения можно использовать в операторе SELECT и встраивать их в условие SELECT. В выражении можно использовать литеральные значения и столбцы, а также присваивать имена столбцам, содержащим значение результата выражения. Мы познакомили вас с типизированными выражениями. Стандарт SQL использует этот термин для обозначения ссылки на столбцы, литеральные значения и выражения. Типизированные выражения можно использовать в различных условиях оператора SQL.

Затем мы обсудили значения Null. Null представляет собой пропущенное или неизвестное значение. Использование Null надлежащим образом может быть достаточно полезным при соответствующих обстоятельствах. Но значения Null оказывают отрицательное воздействие на математические операции, поскольку математическая операция, использующая значение Null, возвращает Null. Значения Null могут сделать неточной информацию в наборе результатов.

В следующей главе дается общее представление об извлечении конкретного набора информации и об использовании условия WHERE для фильтрации информации, извлекаемой оператором SELECT.

Задачи для самостоятельного решения

Ниже приведены формулировки запросов и имена решения этих запросов в учебных базах данных. Попрактикуйтесь немного и разработайте SQL для каждого запроса, а затем сверьте свой ответ с запросом, который сохранен нами в этих базах данных. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов,— важно, чтобы набор результатов был тем же.

База данных заявок на закупку

1. *"What if we adjusted each product price by reducing it 5%?"*
   ("Что будет, если уменьшить цену каждого продукта на 5%?")
   Решение можно найти в Adjusted_Wholesale_Prices (90 строк).

2. *"Show me a list of orders made by each customer in descending date order"*.
   ("Показать заказы, размещенные каждым клиентом, в порядке убывания даты".)
   (*Совет*: Возможно, для надлежащего отображения информации, потребуется упорядочить более одного столбца.)
   Решение можно найти в Orders_By_Customer_And_Date (944 строки).

3. *"Compile a complete list of vendor names and addresses in vendor name order"*.
   ("Составить полный список имен и адресов поставщиков, упорядоченный по именам поставщиков".)
   Решение можно найти в Vendor_Addresses (10 строк).

База данных эстрадных мероприятий

1. *"Give me the names of all our customers by city"*.
   ("Показать имена всех наших клиентов по городам".)
   (*Совет*: Следует использовать условие ORDER BY к одному из столбцов.)
   Решение можно найти в Customers_By_City (15 строк).

2. *"List all entertainers and their Web sites"*.
   ("Показать список всех эстрадных исполнителей и их Web-сайты".)
   Решение можно найти в Entertainer_Web_Sites (13 строк).

3. *"Show the date of each agent's first six-month performance review"*.
   ("Показать даты эстрадных мероприятий каждого агента
   за первые полгода".)
   (*Совет*: Для ответа потребуется использовать арифметику дат.)
   Решение можно найти в First_Performance_Review (8 строк).

База данных расписания занятий

1. *"Give me a list of staff members, and show them in descending order of salary"*.
   ("Показать список персонала и упорядочить его
   в порядке убывания оклада".)
   Решение можно найти в Staff_List_By_Salary (27 строк).

2. *"Can you give me a staff member phone list?"*
   ("Можно ли предоставить мне список телефонов персонала?")
   Решение можно найти в Staff_Member_Phone_List (27 строк).

3. *"List the names of all our students, and order them by the cities they live in"*.
   ("Привести список всех наших студентов и упорядочить его
   по городам проживания".)
   Решение можно найти в Students_By_City (18 строк).

База данных лиги игроков в боулинг

1. *"Show next year's tournament date for each tournament location"*.
   ("Показать дату следующего годового турнира для каждого
   места проведения".)
   Решение можно найти в Next_Years_Tourney_Dates (14 строк).

2. *"List the name and phone number for each member of the league"*.
   ("Привести список имен и номера телефонов для каждого
   участника лиги".)
   Решение можно найти в Phone_List (32 строки).

3. *"Give me a listing of each team's lineup"*.
("Дать список игроков каждой команды".)
(*Совет*: В основу этого запроса можно положить
таблицу "Игроки в боулинг".)
Решение можно найти в Team_Lineups (32 строки).

6

Фильтрация данных

"Есть у меня шестерка слуг, Проворных, удалых.
Зовут их — Как и Почему, Кто, Что, Когда и Где."

— Редьярд Киплинг, *Шестерка слуг*

Вопросы, рассматриваемые в данной главе:

- Уточнение полученного с использованием WHERE
- Определение условий поиска
- Использование нескольких условий
- Повторная встреча с NULL
  Предупреждающее замечание
- Выражение условий различными способами
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

В данной главе мы покажем, как выполнить "тонкую настройку" того, что было извлечено, с помощью фильтрации информации, используя условие WHERE.

Уточнение полученного с использованием WHERE

Оператор SELECT, с которым мы работали до сих пор, извлекал все строки из указанной таблицы и использовал их в наборе результатов оператора. Это прекрасно, если действительно нужно видеть всю информацию,.которую может предоставить таблица. Но если нужно найти только те строки, которые относятся к конкретному лицу, конкретному месту, конкретному цифровому значению или диапазону дат? Такие запросы не являются чем-то необычным. Фактически они сопровождают многие вопросы, с которыми обращаются к базе данных. Например, может быть необходимо задать вопросы следующего типа:

"Кто из наших клиентов проживает в Сиэтле?"

"Показать текущий список наших сотрудников из Беллевью и указать их номера телефонов".

"Какие лекции по музыке предлагаются в настоящее время?"

"Предоставить список занятий, которые оцениваются в три условных балла".

"У каких артистов есть собственные Web-сайты?"

"Предоставить список ангажементов для "Трио Каролины Койе".

"Предоставить список клиентов, которые разместили заказы в мае".

"Предоставить имена наших штатных сотрудников, которые были приняты на работу 16 мая 1985 г.".

"Какое расписание турниров у "Red Rooster Lanes"?"

"Кто из игроков в боулинг не был определен в команду?"

Для того чтобы ответить на эти вопросы, требуется снова расширить словарь SQL, добавив дополнительное условие к вашему оператору SELECT: условие WHERE.

Условие WHERE

Условие WHERE в операторе SELECT используется для фильтрации данных, которые оператор извлекает из таблицы. WHERE содержит *условие поиска*, применяемое оператором в качестве фильтра. Именно это условие поиска обеспечивает механизм, необходимый для выбора только требуемых строк или для удаления ненужных. СУБД применяет это условие поиска к каждой строке в логической таблице, определенной условием FROM. На рис. 6.1 представлен синтаксис оператора SELECT с условием WHERE.

Условие поиска содержит один или более *предикатов*, каждый из которых является выражением, которое тестирует одно или несколько типизированных выражений и возвращает в ответе True, False или Unknown. Несколько предикатов

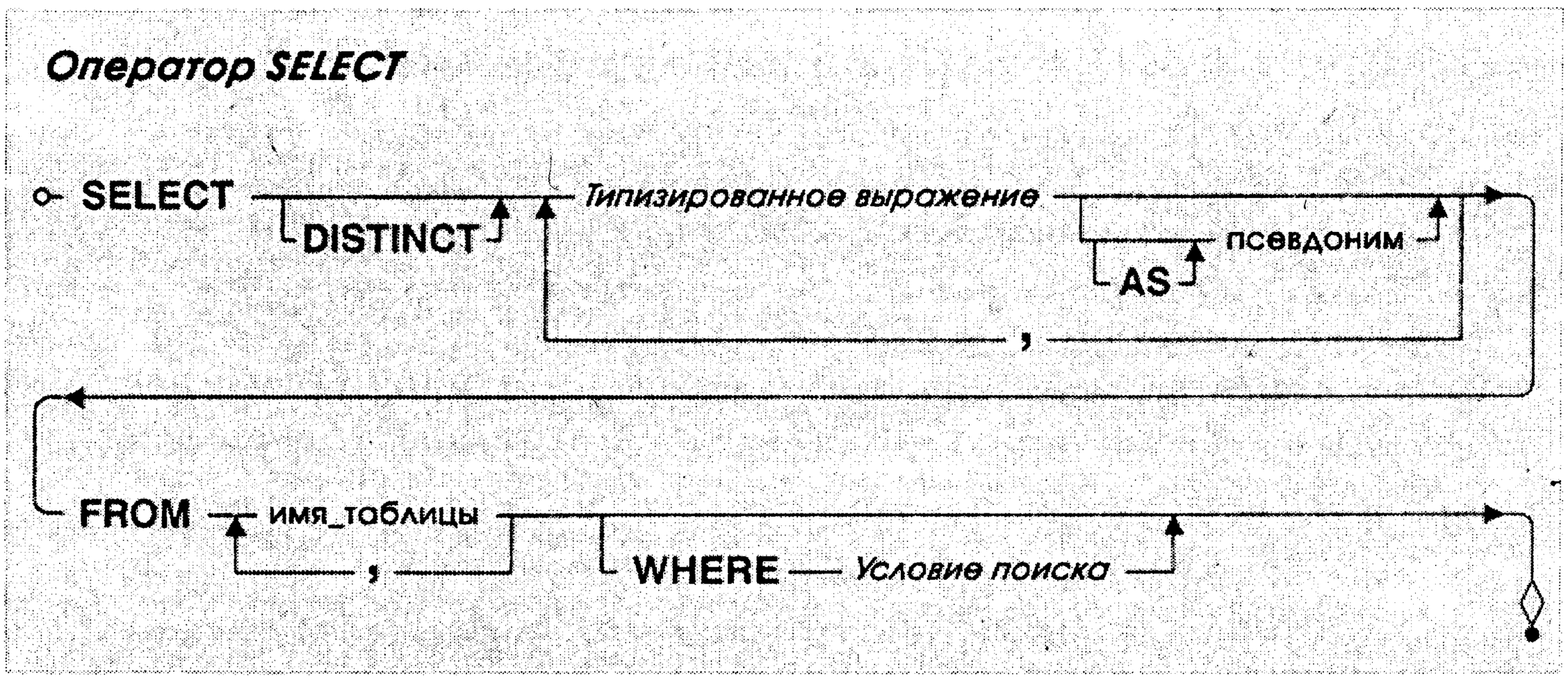


Рис. 6.1. *Оператор SELECT с условием WHERE*

можно объединять в условие поиска, используя булевы операторы AND или OR. Когда условие поиска оказывается равным True для конкретной строки, эта строка будет присутствовать в окончательном наборе результата. Когда условие поиска содержит только один предикат, термины "условие поиска" и "предикат" являются синонимами.

Типизированное выражение может содержать имена столбца, значения литерала, функции или другие типизированные выражения. При построении предиката обычно включается по крайней мере одно типизированное выражение, которое указывает столбец из таблиц, определенных в условии FROM.

Самый простой и, возможно, наиболее распространенный предикат сравнивает одно типизированное выражение (столбец) с другим (литерал). Например, если нужны только те строки из таблицы Customers, в которых значение столбца фамилии клиента — "Smith", то записывается предикат, который сравнивает столбец фамилии со значением литерала "Smith".

SQL
```
SELECT CustLastName
FROM Customers
WHERE CustLastName = 'Smith'
```

Предикат в условии WHERE эквивалентен обращению с вопросом к каждой строке в таблице Customers: "Совпадает ли фамилия клиента со "Smith"?" Когда ответ положительный (True) для произвольной строки в таблице Customers, эта строка появляется в наборе результатов.

Стандарт SQL определяет пять базовых предикатов: сравнение, BETWEEN, IN, LIKE и IS NULL.

Сравнение	Для сравнения одного типизированного выражения с другим используется один из шести операторов сравнения: (=, <>, <, >, <=, >=).
Диапазон	Предикат BETWEEN позволяет тестировать, попадает ли указанное типизированное выражение в указанный диапазон значений. Диапазон определяется с помощью двух типизированных выражений, разделенных ключевым словом AND.
Принадлежность	Используя предикат IN, можно проверить, совпадает ли значение указанного типизированного выражения с элементом заданного списка значений.
Поиск по шаблону	Предикат LIKE позволяет проверить, совпадает ли выражение типа "символьная строка" с указанным образцом символьной строки.
Null	Используйте предикат IS NULL для определения, равно ли типизированное выражение Null.

Использование условия WHERE

Вначале рассмотрим еще один пример построения простого условия WHERE. На этот раз представим подробное изложение действий по построению запроса.

> **Внимание!** В данной главе используется метод "Запрос/Преобразование/Уточнение/SQL", введенный в главе 4.

Предположим, что к базе данных обращаются со следующим вопросом:

"Какие фамилии у наших клиентов, живущих в штате Вашингтон?"

При составлении преобразуемого утверждения для этого типа запросов необходимо указывать информацию, которую желательно получить в наборе результатов, как можно более явно и недвусмысленно. Приложите некоторые усилия для перефразирования запроса, но результаты оправдают дополнительную работу. Вот как преобразуется этот конкретный запрос:

Преобразование: Select first name and last name from the customers table for those customers who live in Washington State
(Выбрать имя и фамилию из таблицы "Клиенты" для клиентов, живущих в штате Вашингтон)

Уточните это преобразование обычным образом и выполните еще две дополнительные задачи. Во-первых, поищите любые слова или фразы, которые указывают или подразумевают некоторый вид ограничений. Слова-указатели — "где", "который" и "для". Ниже приведены некоторые примеры типов фраз, которые вы пытаетесь распознать:

"...которые живут в Беллевью".

"... для всех, чей почтовый код 98125".

"...которые разместили заказы в мае".

"... для поставщиков из Калифорнии".

"...которые были приняты на работу 16 мая 1985 г.".

"... где трехзначный междугородный телефонный код равен 425".

"...для Майкла Хернандеса".

Когда такое ограничение найдется, перед вами встанет вторая задача. Изучите фразы и попытайтесь определить, какой столбец должен быть проверен, на какое значение предполагается проверить этот столбец и как предполагается выполнить проверку столбца. Ответы на эти вопросы помогут вам сформулировать условие поиска в условии WHERE. Рассмотрим эти вопросы по отношению к нашему преобразуемому утверждению.

- Какой столбец нужно проверить? **State (Штат)**

- На какое значение его необходимо проверить? **'WA' (Вашингтон)**
- Как предполагается проверить столбец? **Используя оператор сравнения на равенство**

Нужно хорошо представлять себе структуру таблицы, используемой для ответа на запрос. Если необходимо, держите под рукой копию структуры таблицы, прежде чем начнете отвечать на эти вопросы.

> **Внимание!** Иногда ответы на вопросы очевидны, а в других случаях — подразумеваются. Мы покажем на других примерах, как выделить и расшифровать правильные ответы.

Воспользуйтесь ответами на вопросы и создайте соответствующее условие. Затем вычеркните исходное ограничение и замените его на слово WHERE с только что созданным условием поиска. Так выглядит уточненное утверждение после выполнения этой задачи:

Уточнение: Select first name ~~and~~ last name from ~~the~~ customers ~~table for those customers who live in~~ where state ~~equals~~ = 'WA' Washington State

(Выбрать имя, фамилию из "Клиенты", где штат = 'WA'.

Теперь можно преобразовать это в соответствующий оператор SELECT:

SQL
```
SELECT CustFirstName, CustLastName
FROM Customers
WHERE CustState = 'WA'
```

Набор результатов полученного оператора SELECT отобразит только тех клиентов, которые проживают в штате Вашингтон. Это все определяется в условии WHERE, т. е. создаются соответствующие условия поиска и помещаются в условие WHERE. Однако подлинная работа состоит в определении условий.

Определение условий поиска

Рассмотрим поближе пять основных типов предикатов, которые можно определить.

Сравнение

Наиболее общим типом условия является тот, который использует предикат сравнения для сравнения двух типизированных выражений друг с другом. Как показано на рис. 6.2, можно определить шесть различных типов сравнений, используя операторы предиката сравнения, представленные на рис. справа.

=	Равно
<>	Не равно
<	Меньше
>	Больше
<=	Меньше, либо равно
>=	Больше, либо равно

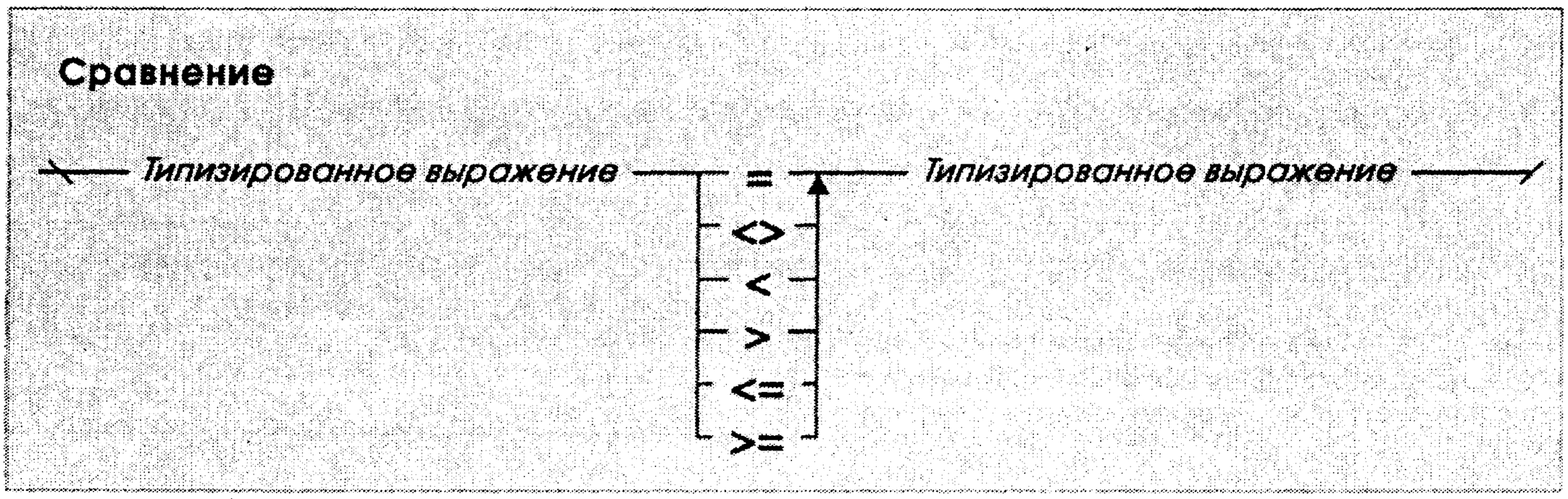


Рис. 6.2. *Синтаксическая диаграмма для условия сравнения*

Сравнение строковых значений: Предостережение

Можно легко сравнивать числовые данные или данные типа дата/время, но необходимо соблюдать величайшую осторожность при сравнении символьных строк. Например, можно не получить ожидаемого результата при сравнении двух таких внешне подобных строк, как “Mike” и “MIKE”. Определяющим фактором сравнения всех символьных строк является последовательность сортировки, используемая системой БД. Последовательность сортировки также определяет, как сортируются символьные строки, и оказывает серьезное влияние на использование других условий сравнения.

Поскольку много различных поставщиков реализовали SQL на машинах с различной архитектурой и для многих языков, отличающихся от английского, стандарт SQL для сортировки или сравнения символьных строк не определяет по умолчанию каких-либо последовательностей сортировки. Способ сортировки строк от наименьших к наибольшим зависит от программного обеспечения базы данных и во многих случаях от того, как оно было инсталлировано.

Многие СУБД используют последовательность сортировки ASCII, при которой цифры располагаются перед буквами, а все буквы верхнего регистра — перед буквами нижнего регистра. Если база данных поддерживает последовательность сортировки ASCII, символы располагаются в следующем порядке от наименьшего значения к наибольшему:

“... 0123456789 ... ABC ... XYZ ... abc ... xyz ...”

Однако в некоторых системах предлагается опция “нечувствительности к регистру”, при которой “a” рассматривается равным “A”. Если база данных поддерживает эту опцию, используя ASCII как основу, символы располагаются в следующей последовательности от наименьшего значения к наибольшему:

“... 0123456789 ... {Aa}{Bb}{Cc} ... {Xx}{Yy}{Zz} ...”

Символы, заключенные в фигурные скобки ({}), рассматриваются как равные.

СУБД для мэйнфреймов IBM используют запатентованную IBM последовательность EBCDIC. В системе баз данных, которые используют EBCDIC, первыми располагаются все буквы нижнего регистра, затем все буквы верхнего регистра,

и, наконец, цифры. Если база данных поддерживает EBCDIC, символы располагаются в следующей последовательности от наименьших значений к наибольшим:

"... abc ... xyz ... ABC ... XYZ ... 0123456789 ..."

Для того чтобы закрепить это в памяти, приведем ряд примеров значений столбцов и посмотрим, какое влияние различные последовательности сортировки оказывают на то, как СУБД определяет вышестоящие, нижестоящие или равные значения.

На рисунке справа приведена таблица значений столбца, отсортированная с использованием набора символов ASCII, с учетом регистра (вначале цифры, потом буквы верхнего, а затем нижнего регистров).

Company Name
3rd Street Warehouse
5th Avenue Market
Al's Auto Shop
Ashby's Cleaners
Zebra Printing
Zercon Productions
allegheny & associates
anderson tree farm
z-tech consulting
zorn credit services

Теперь отключим чувствительность к регистру, так чтобы буквы на нижнем и верхнем регистрах рассматривались как равные (см. рис. слева).

Company Name
3rd Street Warehouse
5th Avenue Market
Al's Auto Shop
allegheny & associates
anderson tree farm
Ashby's Cleaners
z-tech consulting
Zebra Printing
Zercon Productions
allegheny & associates
anderson tree farm
z-tech consulting
zorn credit services
Al's Auto Shop
Ashby's Cleaners
Zebra Printing
Zercon Productions
3rd Street Warehouse
5th Avenue Market

Наконец, рассмотрим, как эти значения сортируются в системе IBM при использовании последовательности сортировки EBCDIC (буквы нижнего регистра, буквы верхнего регистра и затем цифры) (см. рис. справа).

Неожиданные результаты можно получить при попытке сравнения двух символьных строк неодинаковой длины, например "John" и "John " или "Mitch" и "Mitchell". К счастью, стандарт SQL четко определяет, как это должно обрабатываться системой базы данных. Прежде чем она сравнит два символа строк неодинаковой длины, она должна добавить специальный, *определенный по умолчанию заполняющий символ* (обычно пробел) к правой стороне меньшей строки, пока та не сравняется по длине с большей строкой. Затем база данных использует свою последовательность сортировки для определения того, равны ли теперь эти две строки друг другу. В результате "John" и "John " равны (раз имеет место заполнение), а "Mitch" и "Mitchell" — нет.

Внимание! Некоторые СУБД отличаются от стандарта SQL тем, что они игнорируют конечные пробелы. Поэтому “John” и “John ” рассматриваются в таких системах как равные (замыкающие пробелы во втором элементе полностью игнорируются). Обязательно проверьте свою систему базы данных, чтобы определить, как она обрабатывает этот тип сравнений и возвращает ли она предполагаемые типы результатов.

Обязательно проверьте документацию по своей системе базы данных и определите, как она сортирует буквы верхнего регистра, буквы нижнего регистра и цифры.

Равенство и неравенство

Рассмотрим дополнительно условие сравнения на равенство, используя соответствующий оператор сравнения.

Предположим, что к базе данных обращаются со следующим запросом:

“Show me the first and last names of all the agents that were hired on March 14, 1977".
(”Показать имена и фамилии всех агентов, которые были приняты 14 марта 1977 г.".)

Поскольку искомые агенты были приняты на работу в конкретный день, можно воспользоваться условием сравнения на равенство с оператором “равно”. Преобразуем этот вопрос для определения соответствующего оператора SELECT:

Преобразование: Select first name and last name from the agents table for all agents hired on March 14, 1977
(Выбрать имя и фамилию из таблицы “Агенты” для всех агентов, принятых на работу 14 марта 1977)

Уточнение: Select first name ~~and~~ last name from ~~the~~ agents ~~table~~ ~~for all agents hired on~~ where date hired = ~~March 14, 1977~~ ‘1977-03-14’
(Выбрать имя, фамилию из “Агенты”, где дата приема на работу = ‘1977-03-14’)

SQL
```
SELECT AgtFirstName, AgtLastName
FROM Agents
WHERE DateHired = '1977-03-14'
```

В данном примере проверяются значения конкретного столбца, чтобы определить, совпадают ли какие-либо значения с указанным значением данных. По существу, мы выполнили *включающий* процесс — рассматриваемая строка из таблицы Agents будет включена в набор результатов, если *только* текущее значение столбца Date-Hired (Дата приема на работу) для этой строки совпадает с указанной датой. Но,

что если нужно выполнить совершенно противоположное действие и *исключить* определенные строки из набора результатов? В этом случае используется условие сравнения с оператором "не равно".

Допустим, предложен следующий запрос:

> *"Give me a list of vendor names and phone numbers for all our vendors, with the exception of those here in Bellevue".*
> *("Предоставить список имен и номеров телефонов всех наших поставщиков за исключением тех, которые находятся в Беллевью".)*

Данный запрос требует необходимо исключить поставщиков, расположенных в Беллевью, и для этой задачи будет использоваться условие "не равно". Фраза "за исключением" явно указывает на то, что условие "не равно" соответствует данному запросу. Помните об этом при рассмотрении процесса преобразования.

Преобразование: Select vendor name and phone number from the vendors table for all vendors except those based in 'Bellevue'
(Выбрать имя поставщика и номер телефона из таблицы "Поставщики" для всех поставщиков за исключением размещенных в Беллевью)

Уточнение: Select vendor name ~~and~~ phone number from ~~the~~ vendors ~~table for all vendors except those based in~~ where city <> 'Bellevue'
(Выбрать имя поставщика, номер телефона из "Поставщики", где город <> Беллевью)

SQL
```
SELECT VendName, VendPhone
FROM Vendors
WHERE VendCity <> 'Bellevue'
```

Внимание! Стандарт SQL использует символ <> для оператора "не равно". В некоторых СУБД используется другая форма записи, например != (в Microsoft SQL Server и Sybase) и ¬= (в IBM DB2). Обязательно уточните по документации своей СУБД соответствующее обозначение для этого оператора.

Это простое условие исключило всех поставщиков из Беллевью. Позже мы покажем другой метод исключения строк из набора результатов.

Меньше и больше

Часто бывает нужно получить строки, в которых конкретное значение столбца меньше или больше некоторого значения. При таком сравнении используются операторы сравнения "меньше" (<), "меньше или равно" (<=), "больше" (>) или

"больше или равно" (>=). Тип сравниваемых данных определяет отношение между этими значениями.

Символьная строка	Такое сравнение определяет, располагается ли значение первого выражения "до" (<) значения или "после" (>) значения второго выражения в сортирующей последовательности. Например, "*a* < *c*" можно интерпретировать, как "Располагается ли *a* раньше *c*?"
Числа	Такое сравнение определяет, является ли значение первого выражения "меньше" (<) или "больше" (>) значения второго выражения. Например, *10* > *5* можно интерпретировать, как вопрос: "*10* больше *5*?".
Дата/Время	Такое сравнение определяет, является ли значение первого выражения "раньше" (<) или "позже" (>), чем значения второго выражения. Например, '*1999-05-16*' < '*1999-12-15*' можно интерпретировать как вопрос: "16 мая 1999 г. раньше 15 декабря 1999 г.?" Даты и время оцениваются в хронологическом порядке.

Посмотрим, как можно использовать предикаты сравнения для ответа на запрос.

> *"Are there any orders where the ship date was accidentally posted earlier than the order date?"*
> *("Имеются ли заказы, для которых дата поставки случайно была указана раньше даты заказа?")*

В этом случае следует воспользоваться оператором сравнения "меньше", потому что нужно определить, была ли внесена какая-либо дата поставки, имевшая место раньше соответствующей даты заказа. Вот как преобразуется этот запрос:

Преобразование:	Select order number from the orders table where the ship date is earlier than the order date (Выбрать номер заказа из таблицы "Заказы", дата поставки которого раньше даты заказа)
Уточнение:	Select order number from ~~the~~ orders ~~table~~ where ~~the~~ ship date ~~is earlier than the~~ < order date (Выбрать номер заказа из "Заказы", для которого дата поставки < даты заказа)

SQL	`SELECT OrderNumber`
	`FROM Orders`
	`WHERE Shipdate < OrderDate`

Набор результата оператора SELECT будет включать только те строки из таблицы Orders (Заказы), для которых выполняется условие поиска.

В следующем примере для извлечения соответствующей информации требуется оператор сравнения “больше”:

> *“Are there any classes that earn more than four credits?”*
> *(“Имеются ли курсы лекций, за которые присуждают более четырех условных очков?”)*

Преобразование: Select class ID from the classes table for all classes that earn more than four credits
(Выбрать идентификатор курса лекций из таблицы “Курсы лекций” для всех курсов, за которые присуждают более четырех условных очков)

Уточнение: Select class ID from ~~the~~ classes ~~table for all classes that earn more than four~~ where credits > 4
(Выбрать идентификатор курса лекций из “Курсы лекций”, где условные очки > 4)

SQL

```
SELECT ClassID
FROM Classes
WHERE Credits > 4
```

Набор результатов, сгенерированный этим оператором SELECT, включает только те лекции, за которые начисляется четыре и больше условных очков, как, например, линейная алгебра и прикладная физика.

Теперь рассмотрим некоторые примеры со значениями, которые могут быть не только больше или меньше, но также равны значению, с которым выполняется сравнение.

> *“I need the names of everyone we’ve hired since January 1, 1989"*
> *(”Мне нужны все имена тех, кто принят к нам на работу с 1 января 1989".)*

Для этого запроса требуется использовать сравнение “больше или равно”, потому что требуется извлечь все даты приема на работу с 1 января 1989 г. до текущего дня, *включая* сотрудников, принятых на эту дату. При выполнении преобразования не забудьте указать все нужные столбцы для условия SELECT.

Преобразование: Select first name and last name as EmployeeName from the employees table for all employees hired since January 1, 1989
(Выбрать имя и фамилию как EmployeeName из таблицы “Сотрудники” для всех сотрудников, принятых на работу с 1 января 1989)

Уточнение: Select first name ~~and~~ || ' ' || last name as EmployeeName
from ~~the~~ employees ~~table for all employees hired since~~
where date hired >= ~~January 1, 1989~~ '1989-01-01'
(Выбрать имя, || ' ' || фамилию как EmployeeName
из "Сотрудники", где дата приема
на работу >= '1989-01-01')

SQL
```
SELECT FirstName || ' ' | LastName
    AS EmployeeName
FROM Employees
WHERE DateHired >= '1989-01-01'
```

Вот еще один запрос, с которым можно обратиться к базе данных:

"Show me a list of products with a retail price of fifty dollars or less".
("Показать список товаров, розничная цена которых пятьдесят долларов или меньше".)

Для этого запроса будет использоваться сравнение "меньше или равно". Это гарантирует, что набор результатов оператора SELECT будет содержать только товары ценой от одного цента до точно пятидесяти долларов. Посмотрим, как преобразовать этот запрос:

Преобразование: Select product name from the products table
for all products with a retail price of fifty dollars or less
(Выбрать наименование товара из таблицы "Товары"
для всех товаров с розничной ценой в 50 долларов
или меньше)

Уточнение: Select product name from ~~the~~ products ~~table~~
~~for all products with a~~ where retail price
~~of~~ <= 50 ~~fifty dollars or less~~
(Выбрать наименование товара из "Товары",
где розничная цена <= 50)

SQL
```
SELECT ProductName
FROM Products
WHERE RetailPrice <= 50
```

В рассмотренных нами примерах мы использовали только один тип сравнения. Позже мы покажем, как компоновать сравнения, используя AND и OR.

Диапазон

С помощью условия проверки диапазона можно проверить, попадает ли значение типизированного выражения в указанный диапазон значений. На рис. 6.3 представлена синтаксическая структура для этого условия.

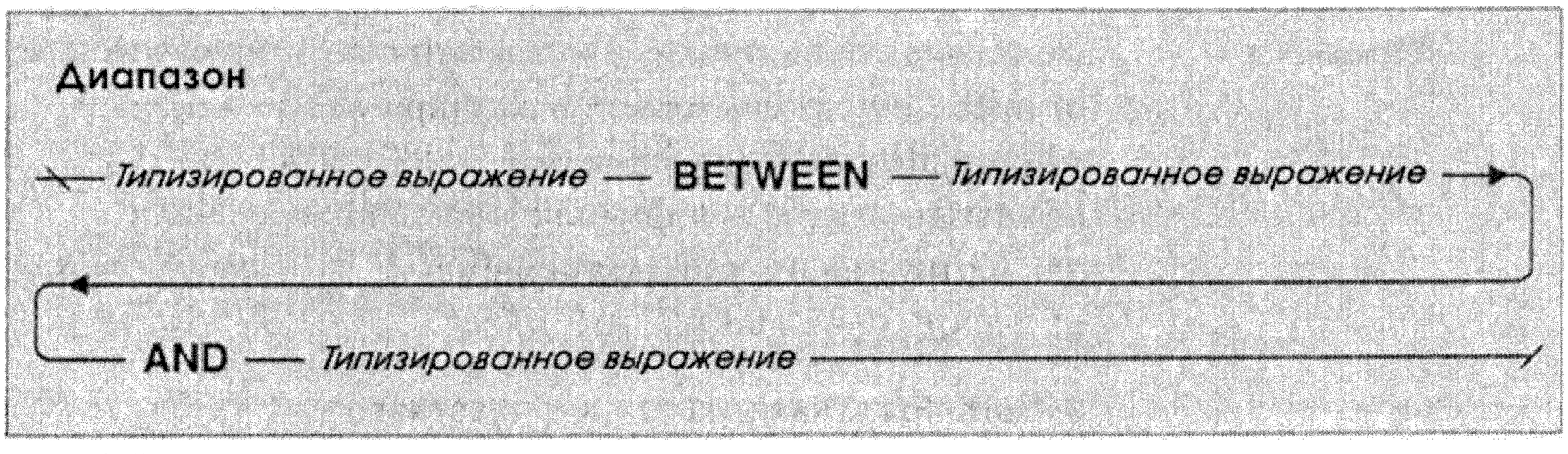


Рис. 6.3. *Синтаксическая диаграмма для условия проверки диапазона*

Условие проверки диапазона тестирует значение указанного типизированного выражения на вхождение в диапазон значений, определенный двумя другими типизированными выражениями. Предикат BETWEEN...AND определяет диапазон, используя значение второго типизированного выражения как начало, а значение третьего типизированного выражения как конец. Точки начала и конца являются частями диапазона. Строка включается в набор результатов, если только значение первого типизированного выражения попадает в указанный диапазон.

В отношении использования BETWEEN...AND необходимо знать следующее. Стандарт SQL предписывает, что выражение:

```
Значение1 BETWEEN Значение2 AND Значение3
```

это то же самое, что и

```
Значение1 >= Значение2 AND Значение1 <= Значение3
```

Это означает, что для выполнения предиката надлежащим образом Значение2 должно быть меньше или равно Значению3. Однако некоторые системы баз данных допускают, чтобы Значение2 было больше или равно Значению3. Проверьте детали в документации на свою систему базы данных.

Приведем примеры использования условий проверки диапазона:

"Which staff members were hired in July of 1986?"
("Какие сотрудники были приняты на работу в июле 1986?")

Здесь целесообразно использование условия проверки диапазона, потому что нужно извлечь имена всех тех, кто был принят на работу в пределах определенного набора дат (в данном случае между 1 и 31 июля 1986 г.) Построим соответствующий оператор SELECT.

Преобразование: Select first name and last name from the staff table where the date hired is between July 1, 1986, and July 31, 1986
(Выбрать имя и фамилию из таблицы "Персонал", для которых дата приема на работу находится между 1 июля и 31 июля 1986)

Уточнение: Select first name ~~and~~ last name from ~~the~~ staff ~~table~~ where ~~the~~ date hired ~~is~~ between ~~July 1,1986~~ '1986-07-01', and ~~July 31, 1986~~ '1986-07-31'
(Выбрать имя, фамилию из "Персонал", где дата приема на работу между '1986-07-01' и '1986-07-31')

SQL
```
SELECT FirstName, LastName
FROM Staff
WHERE DateHired
BETWEEN '1986-07-01' AND '1986-07-31'
```

Обратите внимание на то, что в преобразуемом утверждении диапазон дат указан более явно, чем в запросе. Используйте эту методику для преобразования запроса как можно более точно и таким образом определяйте соответствующий оператор SELECT.

В следующем примере показано, что условие проверки диапазона также достаточно эффективно можно использовать для данных типа символьной строки:

"Give me a list of students — along with their phone numbers — whose last name begins with the letter B".
("Предоставить список студентов — вместе с их телефонными номерами,— фамилии которых начинаются с буквы B".)

Преобразование: Select last name, first name, and phone number from the students table for all students whose last name begins with the letter 'B'
(Выбрать фамилию, имя и номер телефона из таблицы "Студенты" для всех студентов, фамилии которых начинаются с буквы 'B')

Уточнение: Select last name, first name, ~~and~~ phone number from ~~the~~ students ~~table for all students whose last name begins with the letter 'B'~~ where last name between 'b' and 'bz'
(Выбрать фамилию, имя, номер телефона из "Студенты", где фамилия между 'b' и 'bz')

SQL
```
SELECT StudLastName, StudFirstName, StudPhoneNumber
FROM Students
WHERE StudLastName
BETWEEN 'b' AND 'bz'
```

При определении диапазона для данных типа символьной строки тщательно обдумывайте значения, которые вы хотите включить. В таком запросе существует три возможных способа указания начала и конца необходимого диапазона. Результаты будут совершенно разными!

`BETWEEN 'a' AND 'c'`	Многие из вас не указали бы “a” как начало, потому что диапазон будет включать всех, имена которых начинаются с этой буквы. Однако это достаточно типичная ошибка.
`BETWEEN 'b' AND 'c'`	Указание начала и конца таким образом, вероятно, возвратит нужные результаты для нашего примера. Однако можно получить неожиданные результаты, основанные на символьных данных, которые вы пытаетесь сравнивать. Вспомните, что оператор BETWEEN *включает* в диапазон начальную и конечную точки. Следовательно, студент, фамилия которого состоит из одной буквы “c”, будет включен в набор результатов.
`BETWEEN 'b' AND 'bz'`	Это наиболее понятный и точный способ указания начала и конца. В большинстве случаев он возвратит нужный результат. Необходимо понимать свои данные, чтобы определить правильный диапазон.

До настоящего момента мы показывали, как можно сузить сферу запроса, используя широкий диапазон значений и уточненный диапазон значений. Теперь рассмотрим, как можно уточнить запросы еще больше, используя явный список значений.

Принадлежность множеству

Условие принадлежности множеству используется для проверки того, что значение типизированного выражения присутствует в списке явно заданных значений. Как видно из рис. 6.4, условие принадлежности множеству использует предикат IN чтобы определить, совпадает ли значение первого типизированного выражения с каким-либо значением в списке значений (заключенных в скобки), которые определяются одним или несколькими типизированными выражениями.

Хотя теоретически в список можно включить почти безграничное количество типизированных выражений, более разумно использовать всего несколько. У нас

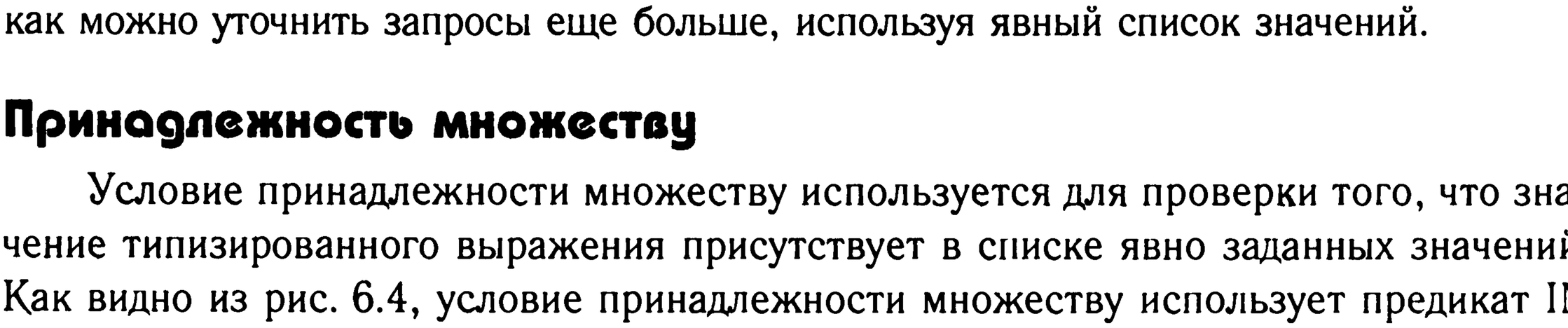


Рис. 6.4. *Синтаксическая диаграмма для условия принадлежности*

уже имеются два условия, которые можно использовать для указания более широких диапазонов значений. Условие принадлежности можно использовать более эффективно, если определяется конечный список значений.

Вот запрос, с которым нужно обратиться к базе данных:

"I need to know which bowling lanes sponsored tournaments for the following 1999 dates: June 5, July 3, and August 7".
("Мне нужно узнать, какие боулинг-клубы устраивали турниры 5 июня, 3 июля и 7 августа 1999".)

Этот тип запроса идеально подходит для условия принадлежности, потому что он направлен на поиск конкретного набора значений. Если бы запрос не был таким явным, то вместо условия принадлежности, скорее всего, использовалось бы условие проверки диапазона. Итак, выполним преобразование этого запроса:

Преобразование: Select tourney location from the tournaments table where the tourney date is in this list of dates:
June 5, 1999; July 3, 1999; August 7, 1999
(Выбрать места проведения турниров из таблицы "Турниры", для которых дата турнира указана в списке дат:
5 июня 1999, 3 июля 1999, 7 августа 1999)

Уточнение: Select tourney location from ~~the~~ tournaments ~~table~~ where ~~the~~ tourney date ~~is~~ in ~~this list of dates:~~
(~~June 5, 1999~~;'1999-06-05', ~~July 3, 1999~~;'1999-07-03', ~~August 7, 1999~~;'1999-08-07')
(Выбрать места проведения турниров из "Турниры", где дата турнира ('1999-06-05', '1999-07-03', '1999-08-07'))

SQL

```
SELECT TourneyLocation
FROM Tournaments
WHERE TourneyDate
IN ('1999-06-05', '1999-07-03', '1999-08-07')
```

Вот другой запрос, который для ответа требует условия проверки диапазона:

"Which entertainers do we represent in Seattle, Redmond and Bothell?"
("Каких эстрадных исполнителей мы представляем в Сиэтле, Редмонде и Бозелле?")

Преобразование: Select stage name from the entertainers table for all entertainers based in 'Seattle', 'Redmond', or 'Bothell'
(Выбрать псевдонимы актера из таблицы "Эстрадные артисты" для всех эстрадных артистов, размещенных в 'Сиэтле', 'Редмонде' или 'Бозелле')

Уточнение: Select stage name from ~~the~~ entertainers ~~table~~
~~for all entertainers based~~ where city
in ('Seattle', 'Redmond', ~~or~~ 'Bothell')
(Выбрать псевдонимы актера из "Эстрадные артисты",
где города в ('Сиэтл', 'Редмонд', 'Бозелл'))

SQL
```
SELECT EntStageName
FROM Entertainers
WHERE EntCity
IN ('Seattle', 'Redmond', 'Bothell')
```

Возможно, вы заметили, что в списке городов преобразованного утверждения использовано слово 'или' вместо "и", которое присутствовало в исходном запросе. Причина и логика этого проста. Существует только одна запись в столбце EntCity для данного эстрадного исполнителя. Возможно, это покажется тривиальным, но использование соответствующих слов и фраз помогает прояснить преобразование и уточнить операторы, а также гарантировать, что для запроса будет определен наиболее подходящий оператор SELECT.

До сих пор мы изучали условия, которые в качестве критерия используют полные значения. Теперь рассмотрим условие, которое позволяет использовать в этом качестве частичные значения.

Совпадение с образцом

Условие совпадения с образцом является полезным, когда нужно найти значения, подобные указанной строке-образцу, или когда только неполная часть информации используется как критерий поиска. На рис. 6.5 представлена синтаксическая структура этого типа условия.

В этом условии берется значение типизированного выражения и используется предикат LIKE для проверки, совпадает ли это значение с заданным образцом строки. Образец строки может состоять из любой логической комбинации обычных строк символов и двух специальных групповых символов: процента (%) и подчеркивания (_). Символ процента представляет ни одного или несколько произвольных обычных

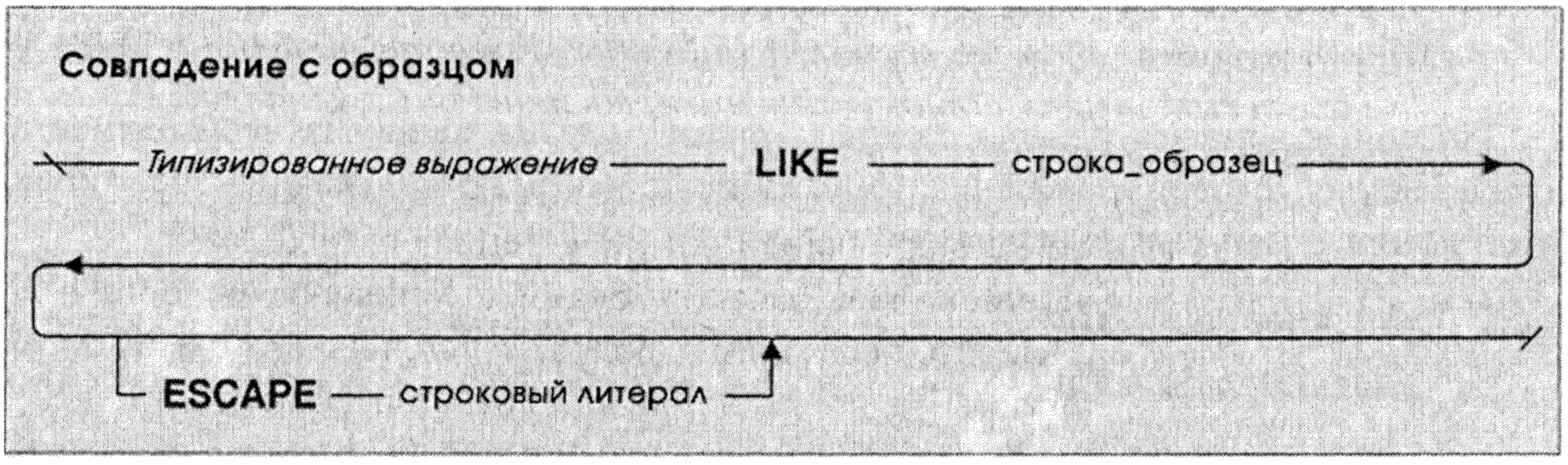


Рис. 6.5. *Синтаксическая диаграмма для условия совпадения с образцом*

символов, а символ подчеркивания представляет собой единственный произвольный обычный символ. Способ определения образца строки устанавливает, какие значения извлекаются из типизированного выражения. В таблице 6.1 показаны примеры различных типов образцов строк, которые можно определить.

Таблица 6.1

Примеры определенных образцов строк

Образец строки	Обработанный критерий	Примеры возвращенных значений
'sha%'	Строка символов может быть любой длины, но должна начинаться с "sha".	**Sha**nnon, **Sha**ron, **Sha**wn
'%son'	Строка символов может быть любой длины, но должна заканчиваться на "son".	Ben**son**, John**son**, Morri**son**
'%han%'	Строка символов может быть любой длины, но должна содержать "han".	Bu**han**an, **Han**del, Jo**han**sen, Nat**han**son
'ro_'	Строка символов может иметь длину только в три символа, а первая и вторая буквы должны быть "ro".	**Ro**b, **Ro**n, **Ro**y
'_im'	Строка символов может иметь длину только в три символа, а вторая и третья буквы должны быть "im".	J**im**, K**im**, T**im**
'_ar_'	Строка символов может иметь длину только в четыре символа, а вторая и третья буквы должны быть "ar".	B**ar**t, G**ar**y, M**ar**k
'_at%'	Строка символов может быть любой длины, но вторая и третья буквы должны быть "at".	G**at**es, M**at**thews, P**at**terson
'%ac_'	Строка символов может быть любой длины, но вторая и третья буквы от конца должны быть "ac".	Apod**ac**a, Tr**ac**y, Wall**ac**e

Посмотрим, как можно использовать условие совпадения с образцом для следующего запроса:

"Give me a list of customers whose last names begins with 'Mar'".
("Предоставить список клиентов, фамилии которых начинаются с 'Mar'".)

Подобные запросы обычно содержат фразы, указывающие на необходимость использования условия совпадения с образцом. Приведем несколько примеров типов фраз, которые могут встретиться:

"...начинается с 'Her'"

"...начинается с 'Ba'"

"...включает слово 'Park'"

"...содержит буквы 'han'"

"...содержит 'ave' в середине"

"...с 'son' в конце"

"...заканчивается на 'ez'"

Определить тип образца строки, необходимого для запроса, относительно легко. Раз известен тип образца, можно продолжить процесс преобразования.

Преобразование: Select last name and first name from the customers table where the last name begins with 'Mar'
(Выбрать фамилию и имя из таблицы "Клиенты", у которых фамилия начинается с 'Mar')

Уточнение: Select last name ~~and~~ first name from ~~the~~ customers ~~table~~ where ~~the~~ last name ~~begins with~~ like 'Mar%'
(Выбрать фамилию, имя из "Клиенты", у которых фамилия соответствует шаблону 'Mar%')

SQL
```
SELECT CustLastName, CustFirstName
FROM Customers
WHERE CustLastName LIKE 'Mar%'
```

Набор результатов для этого оператора SELECT включает такие фамилии, как "Marks", "Marshall", "Martinez" и "Marx", потому что нас интересует совпадение только первых трех букв.

Вот другой запрос, на который можно ответить, используя условие совпадения:

"Show me a list of vendor names where the word 'Forest' appears in the street address".
("Показать список поставщиков, у которых название улицы в адресе содержит слово Forest".)

Преобразование: Select vendor name from the vendors table where the street address contains the word 'Forest'
(Выбрать имя поставщика из таблицы "Поставщики", у которого название улицы в адресе содержит слово 'Forest')

Уточнение: Select vendor name from ~~the~~ vendors ~~table~~ where ~~the~~ street address ~~contains the word~~ like '%Forest%'
(Выбрать имя поставщика из "Поставщики", у которого улица типа '%Forest%')

SQL
```
SELECT VendName
FROM Vendors
WHERE VendStreetAddress LIKE '%Forest%'
```

В данном случае строка из таблицы Vendors (Поставщики) включается в набор результатов, только если адрес содержит такие названия улиц, как “Forest Park Place”, “Forest Ridge Avenue”, “Evergreen Forest Drive” или “Black Forest Road”.

Хотя с помощью соответствующих групповых символов можно вести поиск для любого образца строки, однако, если значения, которые нужно извлечь, сами одержат символ процента или подчеркивания, возникает проблема. Например, так произойдет при попытке извлечь значение ‘MX_445’, потому что оно содержит символ подчеркивания. Из этого затруднительного положения можно выйти, используя опцию ESCAPE предиката LIKE, как показано на рис. 6.5.

Опция ESCAPE позволяет назначить *единообразный* строковый литерал, называемый *управляющим символом*, для указания того, как СУБД должна интерпретировать символ процента или символ подчеркивания внутри образца строки. Поместите управляющий символ после ключевого слова ESCAPE и заключите его в одиночные кавычки, как для любого литерала типа символьной строки. Когда в образце строки управляющий символ предшествует групповому символу, СУБД интерпретирует этот групповой символ *буквально*.

Вот пример использования опции ESCAPE:

> *“Show me a list of products that have product codes beginning with ‘G_00’ and ending in a single number or letter”.*
> *(“Показать список товаров с кодом товара, который начинается с ‘G_00’, а заканчивается одной цифрой или буквой”.)*

Преобразование: Select product name and product code from the products table where the product code begins with ‘G_00’ and ends in a single number or letter
(Выбрать наименование и код продукта из таблицы “Продукты”, для которых код начинается с ‘G_00’ и заканчивается отдельной цифрой или буквой)

Уточнение: Select product ~~name~~ and product code from ~~the~~ products ~~table~~ where ~~the~~ product code ~~begins with~~ like ‘G_00_’ ~~and ends in a single number or letter~~
(Выбрать наименование и код продукта из “Продукты”, для которых код типа ‘G_00_’)

SQL

```
SELECT ProductName, ProductCode
FROM Products
WHERE ProductCode Like 'G\_00_' ESCAPE '\'
```

Очевидно, что для ответа на этот вопрос необходимо использовать опцию ESCAPE. В противном случае СУБД интерпретирует символ подчеркивания в образце строки как групповой символ. Обратите внимание на то, что управляющий символ включен в уточнение. Это гарантирует, что при определении оператора SELECT вы не забудете использовать опцию ESCAPE.

Этот оператор SELECT будет извлекать такие коды товаров, как G_002 и G_00X. Поскольку мы хотим отыскать один из двух символов, которые определены в стандарте как групповые символы, *необходимо* включить условие ESCAPE. Если в запросе будет указано `LIKE 'G_00_'`, СУБД возвратит строки, в которых первой буквой в коде товара будет G, во второй позиции — *любой* символ (поскольку указан групповой символ), нули в третьей и четвертой позициях и любой символ в пятой позиции. Когда в качестве символа ESCAPE определяется "\", СУБД игнорирует управляющий символ, но интерпретирует первый символ подчеркивания буквально, а не как групповой символ. Поскольку непосредственно перед вторым символом подчеркивания управляющий символ не используется, СУБД интерпретирует второй символ подчеркивания как настоящий групповой символ.

Помните, что символ, используемый как управляющий, не должен быть частью значения, которое вы пытаетесь найти. Поэтому не следует использовать "&" как управляющий символ, если осуществляется поиск таких значений, как "Martin & Lewis", "Smith & Kearns" или "Hernandez & Viescas". Управляющий символ действует только на групповой символ, расположенный непосредственно за ним. Однако при необходимости можно использовать в образце строки столько управляющих символов, сколько требуется.

Null

Теперь обсудим поиск *неизвестных* значений. Из главы 5 вы узнали, что значение Null *не* представляет собой нуль, символьную строку из одного или нескольких пробелов или символьную строку нулевой длины (т. е. не содержащую в себе символов), потому что каждый из этих элементов может быть значащим во множестве различных ситуаций. Null *представляет* собой пропущенное или неизвестное значение. Для извлечения значений Null из типизированного выражения используется *условие Null*, представленное на рис. 6.6.

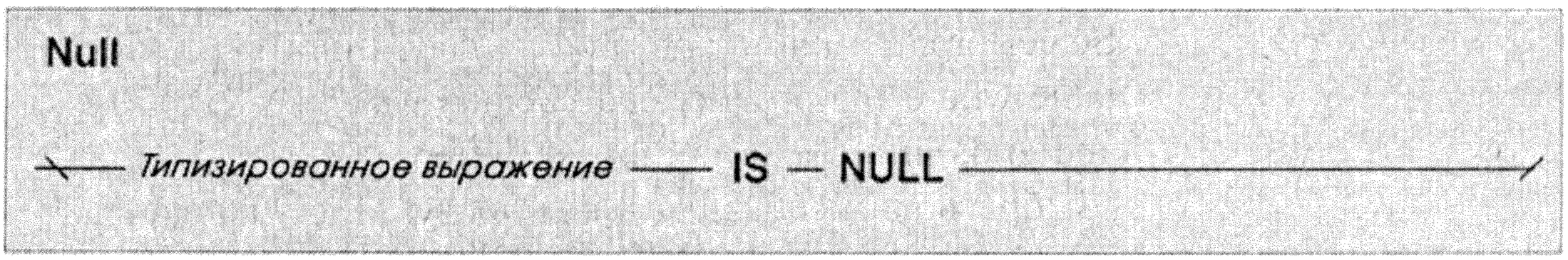


Рис. 6.6. *Синтаксическая диаграмма для условия Null*

В этом условии берется значение типизированного выражения и с помощью предиката IS NULL определяется, является ли оно значением Null. Это достаточно простая операция. Рассмотрим использование условия Null в примерах.

> *"Give me a list of customers who didn't specify what county they live in".*
> *("Предоставить список клиентов, которые не указали,*
> *в каком округе они живут".)*

Преобразование: Select first name and last name as Customer from the customers table where the county name is unspecified (Выбрать имя и фамилию как Customer из таблицы "Клиенты", для которых не определен округ)

Уточнение: Select first name || ' ' || ~~and~~ last name as Customer from ~~the~~ customers ~~table~~ where ~~the~~ country name is null ~~unspecified~~ (Выбрать имя, || ' ' || фамилию как Customer из "Клиенты", где название округа is null)

```
SQL                SELECT CustFirstName || ' ' || CustLastName
                       AS Customer
                   FROM Customers
                   WHERE CustCounty IS NULL
```

В наборе результатов для этого оператора SELECT будут присутствовать имена только тех клиентов, которые не знали или не могли вспомнить, в каком округе они живут, или те, кто проживает в Вашингтоне, округ Колумбия.

Вот еще один запрос, с которым можно обратится к базе данных:

"Which engagements do not yet have a contract price?"
("У каких ангажементов до сих пор нет договорной цены?")

Преобразование: Select engagement number and contract price from the engagements table for any engagement that does not have a contract price (Выбрать номер и договорную цену ангажемента из таблицы "Ангажементы" для всех ангажементов, у которых отсутствует договорная цена)

Уточнение: Select engagement number ~~and~~ contract price from ~~the~~ engagements ~~table for any engagement that does not have a~~ where contract price is null (Выбрать номер, договорную цену ангажемента из "Ангажементы", где договорная цена Null)

```
SQL                SELECT EngagementNumber, ContractPrice
                   FROM Engagements
                   WHERE ContractPrice IS NULL
```

По внешнему виду данный запрос кажется простым: отыскать все ангажементы, договорная цена которых "0". Но впечатление может быть обманчивым и внушить неверные предположения. Если агентство эстрадных мероприятий использует "0" как договорную цену для любого рекламируемого ангажемента, то ноль является допустимым, имеющим смысл значением. Поэтому любая договорная цена, о которой еще договариваются, несомненно, должна быть значением Null.

Данный пример еще раз демонстрирует, что необходимо понимать свои данные и составлять имеющие смысл, точные запросы к базе данных. Если при выполнении оператора SELECT вы видите, что информация, представленная в наборе результата, является ошибочной, не стоит сразу паниковать. Первым вашим импульсом, вероятно, будет желание переписать весь оператор SELECT, чтобы исправить какую-то роковую ошибку в синтаксисе. Однако прежде, чем сделать что-либо радикальное, еще раз просмотрите данные, с которыми работаете, и убедитесь в том, что четко понимаете, как их использовать. Как только будет достигнуто лучшее понимание этих данных, может оказаться, что достаточно внести только небольшие изменения в оператор SELECT, чтобы получить нужную информацию.

> **Внимание!** Для поиска значений Null в типизированном выражении требуется применять условие Null. Такие условия, как `<ValueExpression> = Null` недействительны, поскольку значение типизированного выражения невозможно сравнить с чем-либо, что по определению является неизвестным.

Исключение строк с помощью NOT

Исключить строки из набора результатов можно с помощью оператора NOT. Мы уже показали один простой способ исключения строк из набора результатов, используя условия сравнения на равенство оператора "не равно". Можно также исключить строки в других типах условий, используя оператор NOT. Как показано на рис. 6.7, этот оператор является необязательной компонентой предикатов BETWEEN, IN, LIKE и IS NULL. При включении оператора NOT оператор SELECT проигнорирует все строки, которые удовлетворяют условию, указанному любым из этих предикатов. В наборе результатов будут присутствовать те строки, которые не *удовлетворяют* условию.

Приведенные далее примеры показывают, как можно использовать NOT в качестве части условия поиска:

> *"Show me a list of all the orders we've taken, except for those posted in July". ("Показать список всех принятых заказов, за исключением зарегистрированных в июле".)*

Подобный запрос требует определить оператор SELECT, который исключает строки, удовлетворяющие конкретному условию, и обычно содержат фразы, которые указывают на необходимость использования оператора NOT как части условия поиска, например:

> *"...которые не начинаются с 'Her'".*
>
> *"...которые отсутствуют в административном отделе или в отделе кадров".*
>
> *"...у кого указан номер факса".*
>
> *"...кто был принят на работу до 1 июня или после 31 августа".*

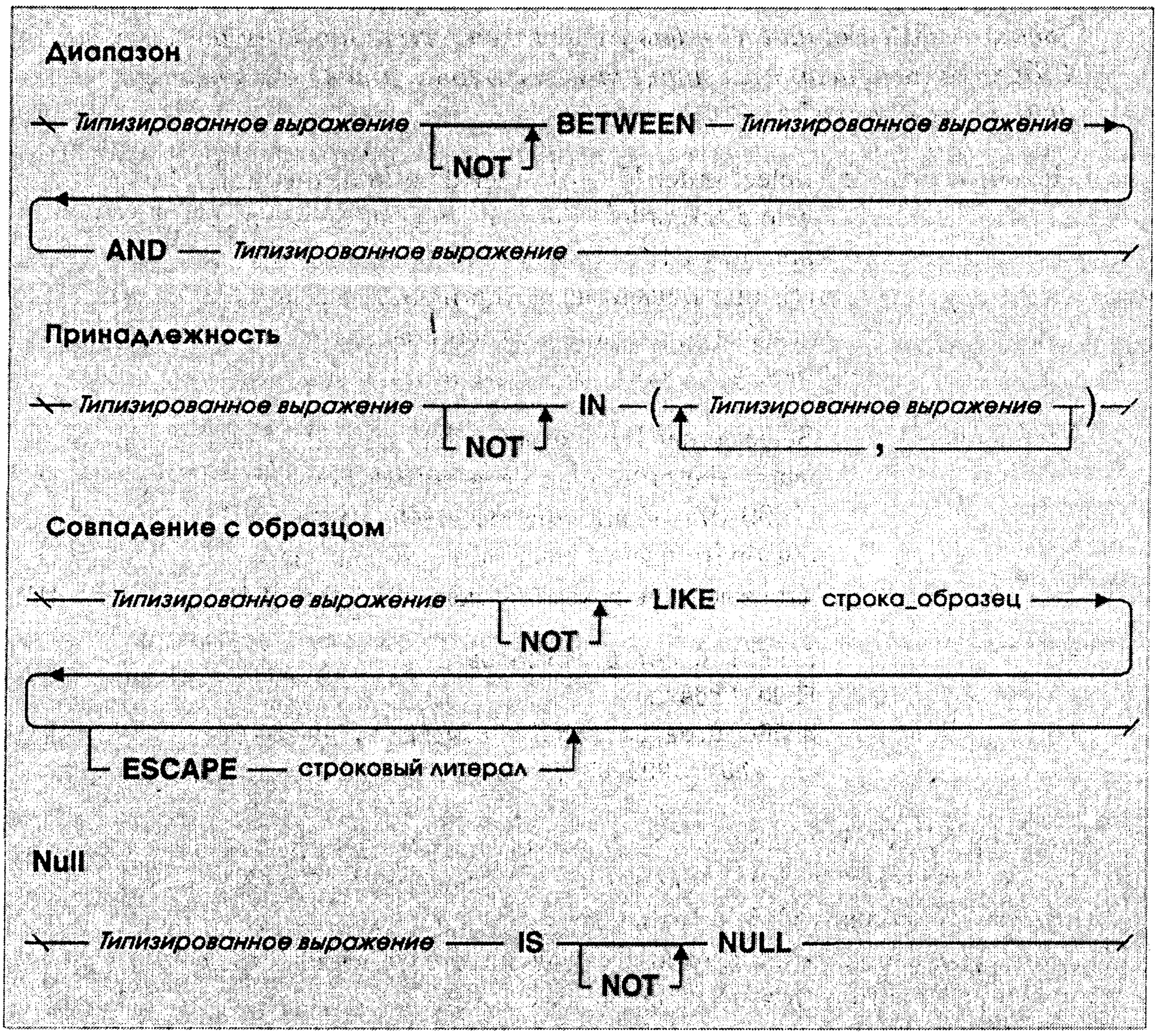


Рис. 6.7. *Синтаксическая структура оператора NOT*

Иногда необходимо выполнить некоторую дедуктивную работу, чтобы преобразовать фразу правильно. Некоторые фразы, например третья из приведенных выше, не содержат явного указания на необходимость применения оператора NOT. В данном случае требование подразумевается, поскольку нужно исключить всех, у кого нет номера факса. При работе с запросами, которые содержат фразы такого типа, необходимо более тщательно анализировать их и, возможно, переделать, чтобы определить, нужно ли исключать определенные строки из набора результатов. К сожалению, для этого нет легкого практичного способа, но, если проявить немного настойчивости и поупражняться, станет легче определять, требуется ли для конкретного запроса оператор NOT.

Как только установлено, что требуется ли исключить какую-либо информацию из набора результатов, можно перейти к процессу преобразования.

"Show me a list of all the orders we've taken, except for those posted in July". ("Показать список всех принятых заказов, за исключением принятых в июле".)

Преобразование: Select order ID and order date from the orders table where the order date does not fall between July 1, 1999, and July 31, 1999
(Выбрать идентификатор и дату заказа из таблицы "Заказы", для которых дата заказа не попадает между 1 июля 1999 и 31 июля 1999)

Уточнение: Select order ID ~~and~~ order date from ~~the~~ orders ~~table~~ where ~~the~~ order date ~~does~~ not ~~fall~~ between ~~July 1, 1999~~, '1999-07-01' and ~~July 31, 1999~~ '1999-07-31'
(Выбрать идентификатор и дату заказа из "Заказы", где дата заказа не находится между '1999-07-01' и '1999-07-31')

SQL
```
SELECT OrderID, OrderDate
FROM Orders
WHERE OrderDate NOT BETWEEN '1999-07-01'
    AND '1999-07-31'
```

Этот оператор SELECT выдает набор результатов, который *не* содержит заказов, зарегистрированных между 1 и 31 июля 1999 г. Однако в нем будут содержаться все другие заказы из таблицы Orders (включая, возможно, строки с 1998 или 2000 г.). Используя несколько условий, можно в дальнейшем ограничить строки, передаваемые в набор результатов, только теми заказами, которые были приняты в 1999 г.

Посмотрим, как обрабатывается следующий запрос:

"I need the identification numbers of all faculty members who are not professors or associate professors".
("Мне нужны идентификационные номера всех преподавателей, которые не являются профессорами или адьюнкт-профессорами".)

Преобразование: Select staff ID and title from the faculty table where the title is not 'professor' or 'associate professor'
(Выбрать идентификатор и должность сотрудников из таблицы "Преподавательский состав, у которых должность не профессор или адьюнкт-профессор)

Уточнение: Select staff ID ~~and~~ title from ~~the~~ faculty ~~table~~ where ~~the~~ title ~~is~~ not in ('professor' ~~or~~ 'associate professor')
(Выбрать идентификатор, должность сотрудника из "Преподавательский состав", где должность не ('профессор', 'адьюнкт-профессор'))

```
SQL                    SELECT StaffID, Title
                       FROM Faculty
                       WHERE Title
                       NOT IN ('Professor', 'Associate Professor')
```

В данном случае необходимо исключить любого сотрудника, должность которого совпадает с указанием в запросе, поэтому используется условие принадлежности с оператором NOT для получения в наборе результатов правильных строк.

Исключение строк из набора результата станет для вас достаточно простым действием, как только вы освоитесь с анализом и перефразируете свои запросы в соответствие с ситуацией. Для этого необходимо уметь определить тип условия, требуемого для ответа на данный запрос.

Использование нескольких условий

Ответ на сложный запрос можно получить, используя несколько условий. Рассмотрим пример:

> *"Give me the first and last names of customers who live in Seattle and whose last name starts with the letter 'H'".*
> *("Предоставить имена и фамилии клиентов, проживающих в Сиэтле, у которых фамилия начинается с буквы Н".)*

Для ответа на этот запрос требуются условие сравнения на равенство и условие совпадения с образцом. Как же объединить эти два необходимые условия в одно условие поиска? Ответ лежит в способе, которым в стандарте SQL определяется синтаксис для условия поиска (см. рис. 6.8).

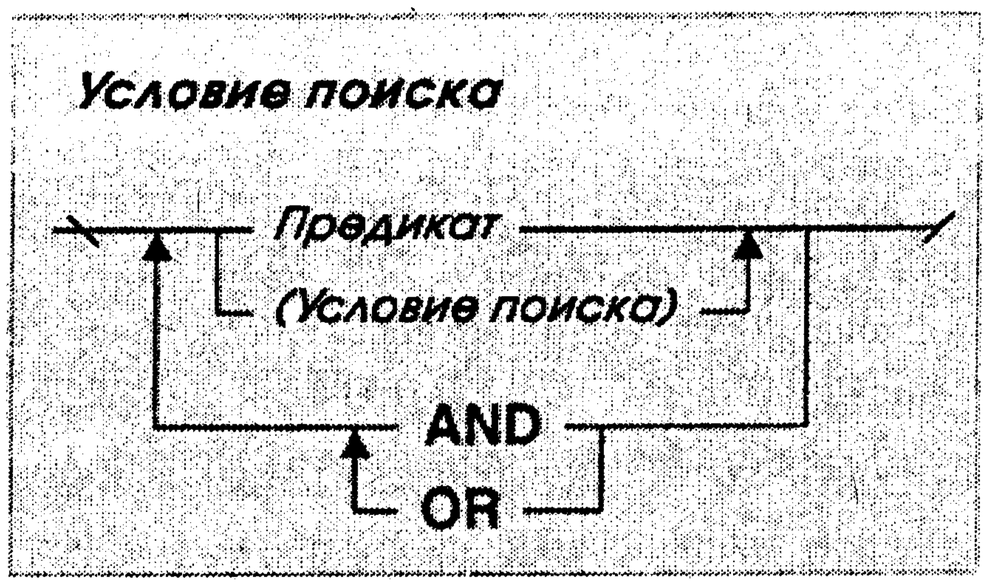


Рис. 6.8. *Синтаксическая диаграмма для условия поиска*

Знакомство с AND и OR

Два или более условий можно объединить, используя операторы AND и OR, а полный набор условий, объединенных
для ответа на данный запрос, составляет единое условие поиска. Как показано на диаграмме, можно также объединить некоторое составное условие поиска с другими условиями, заключив это составное условие в скобки. Все это позволяет создавать очень сложные условия WHERE, которые точно регулируют, какие выбранные строки должны включаться в набор результатов.

Использование AND

Первый способ объединения двух или более условий состоит в использовании оператора AND. Этот оператор применяется, когда для включения строки в набор результата необходимо удовлетворить ряду объединенных условий. Воспользуемся

примером запроса, приведенным в начале этой части, и применим оператор AND во время процесса преобразования.

> *"Give me the first and last names of customers who live in Seattle and whose last name starts with the letter 'H'"*
> *("Предоставить имя и фамилию клиентов, проживающих в городе Сиэтл, у которых фамилия начинается с 'H'".)*

Преобразование: Select the first name and last name from the customers table where the city is 'Seattle' and the last name begins with 'H'
(Выбрать имя и фамилию из таблицы "Клиенты", где город — "Сиэтл" и фамилия начинается с 'H')

Уточнение: Select the first name ~~and~~ last name from ~~the~~ customers ~~table~~ where ~~the~~ city ~~is~~ = 'Seattle' and ~~the~~ last name ~~begins with~~ like 'H%'
(Выбрать имя, фамилию из "Клиенты", где город = 'Seattle' и фамилия соответствует шаблону 'H%')

SQL
```
SELECT CustFirstName, CustLastName
FROM Customers
WHERE CustCity = 'Seattle'
AND CustLastName LIKE 'H%'
```

Здесь во внимание принято как условие сравнения на равенство, так и условие совпадения с образцом, необходимые для запроса. Оба условия будут удовлетворены, что обеспечивается использованием оператора AND. Любая строка, которая не удовлетворяет каждому из условий, будет исключена из набора результатов.

Можно объединить в цепочку любое количество условий, необходимых для ответа на требуемый запрос. Для того чтобы строка была включена в набор результатов, *должны* быть удовлетворены *все* условия, которые были объединены с помощью оператора AND,
т. е. все условие поиска целиком должно быть оценено как True (Истина). На рис. 6.9 представлен результат объединения двух предикатов типа "выражение" с помощью оператора AND. Если любое из выражений оценивается как False (Ложь), строка не выбирается.

Первое выражение \ Второе выражение	AND	Истина	Ложь
	Истина	Истина (Строки выбираются)	Ложь (Строки отвергаются)
	Ложь	Ложь (Строки отвергаются)	Ложь (Строки отвергаются)

Рис. 6.9. *Результат объединения двух предикатов типа "выражение" в операторе AND*

Использование OR

Второй способ объединения двух или более условий состоит в использовании оператора OR. Этот оператор применяется, когда для включения строки в набор результатов должно быть удовлетворено любое из *объединяемых* условий. Ниже приведен пример использования оператора OR в условии поиска:

> *"I need the name, city, and state of every staff member who lives in Seattle or is from the state of Oregon".*
> *("Мне нужно имя, город и штат каждого сотрудника, проживающего в Сиэтле или в штате Орегон".)*

Преобразование: Select first name, last name, city, and state from the staff table where the city is 'Seattle' or the state is 'OR'
(Выбрать имя, фамилию, город и штат из таблицы "Персонал", где город "Сиэтл" или штат "OR")

Уточнение: Select first name, last name, city, ~~and~~ state from ~~the~~ staff ~~table~~ where ~~the~~ city ~~is~~ = 'Seattle' or ~~the~~ state ~~is~~ = 'OR'
(Выбрать имя, фамилию, город, штат из "Персонал", где город = 'Seattle' или штат = 'OR')

SQL

```
SELECT StfFirstName, StfLastName, StfCity, StfState
FROM Staff
WHERE StfCity = 'Seattle' OR StfState = 'OR'
```

В данном случае во внимание принимается как условие сравнения на равенство, необходимое для ответа на запрос, так и необходимость обеспечить выполнение *одного* из условий, что гарантируется использованием оператора OR. Если строка удовлетворяет какому-либо условию, она будет включена в набор результатов. Для прояснения вопроса на рис. 6.10 показан результат объединения двух предикатов типа "выражение" оператором OR.

Определить необходимость использования оператор AND для объединения условий относительно просто. Однако с оператором OR дело обстоит сложнее.

Второе выражение

Первое выражение / OR	Истина	Ложь
Истина	Истина (Строки выбираются)	Истина (Строки выбираются)
Ложь	Истина (Строки выбираются)	Ложь (Строки отвергаются)

Рис. 6.10. *Результат объединения двух предикатов типа "выражение" оператором OR*

Рассмотрим, например, следующий запрос:

> *"Show me a list of vender names and phone numbers for all venders based in Washington and California".*

("Показать список имен и телефонных номеров поставщиков, находящихся в штатах Вашингтон и Калифорния".)

Первым желанием будет использовать оператор AND, поскольку условие очевидно: нужны имена поставщиков в Вашингтоне *и* Калифорнии. К сожалению, это ошибка. Подумайте: ведь поставщик будет размещаться *либо* в Вашингтоне, *либо* в Калифорнии, потому что *можно ввести только одно значение для штата в столбец "Штат" для некоторого поставщика.* Реальное условие намного более понятно, не так ли? Как уже говорилось, необходимо ввести в привычку изучение и анализ своих запросов по мере того, как они становятся все более сложными, и как можно более тщательно просматривать предполагаемые условия.

Продолжим рассмотрение и выполним преобразование этого запроса:

"Show me a list of vender names and phone numbers for all vendors based in Washington and California".

Преобразование: Select name, phone number, and state from the vendors table where the state is 'WA' or 'CA'
(Выбрать имя, номер телефона и штат из таблицы "Поставщики", для которых штат 'WA' или 'CA')

Уточнение: Select name, phone number, ~~and~~ state from ~~the~~ vendors ~~table~~ where ~~the~~ state ~~is~~ = 'WA' or state = 'CA'
(Выбрать имя, номер телефона, штат из "Поставщики", для которых штат = 'WA' или штат = 'CA')

SQL
```
SELECT VendName, VendPhoneNumber, VendState
FROM Vendors
WHERE VendState = 'WA' OR VendState = 'CA'
```

Необходимо принять во внимание оба условия сравнения на равенство и с помощью оператора OR гарантировать, что какое-либо из них будет удовлетворено. Обратите внимание, что "state" (штат) появляется в условии поиска на шаге уточнения и в операторе SQL дважды. Это необходимо, поскольку любое условие сравнения следует одному и тому же синтаксису:

```
"Value Expression <оператор сравнения> Value Expression"
```

Помните, что недопустимо пропускать любое из условий, ключевых слов или определенных терминов из синтаксической структуры, если только это не указано явно как необязательный элемент. Поэтому такое условие, как `WHERE VendState = 'WA' OR 'CA'` является совершенно неверными. Позже мы более подробно рассмотрим последовательность, в которой осуществляется оценка операторов, а в данном случае система баз данных оценивает выражение в строгой последовательности слева направо. Итак, в первую очередь будет оцениваться `VendState = 'WA'`.

Для любой заданной строки результатом будет True, если указан штат Вашингтон, и False, если нет. Затем над этим результатом (True или False) выполняется операция 'OR' со значением литерала — которое не является значением True или

False! Ваша СУБД может возвратить в этот момент ошибку ('CA' — строковый литерал — неверный тип данных для оператора OR) или возвратить только те строки, где штат — Вашингтон.

Всегда следите за тем, чтобы ваши условия были полностью и правильно определены. В противном случае условие поиска для оператора SELECT вызовет ошибку.

> **Внимание!** Этот пример использовался для пояснения обычной ловушки, в которую попадают при использовании оператора OR. Однако если предполагается, что можно использовать такое условие принадлежности, как "WHERE VendState IN ('WA', 'CA')", то это абсолютно правильно. Иногда для выражения некоторого условия существует более одного способа.

Совместное использования AND и OR

Для ответов на особенно сложные запросы можно использовать AND и OR вместе. Например, используя оба оператора, можно ответить на запрос следующего типа:

> *"I need to see the names of staff' members who have a 425 area code and a phone number that begins with 555, along with anyone who was hired between October 1 and December 31 of 1999".*
> *("Мне необходимо узнать имена сотрудников, междугородный телефонный код которых — 425 и номер телефона начинается с 555, вместе с теми, кто был принят на работу в промежутке между 1 октября и 31 декабря 1999".)*

Сейчас вы должны легко определить, какие типы условий необходимы для данного запроса. Этих условий три: условие сравнения на равенство — для поиска междугородного телефонного кода, условие совпадения с образцом — для поиска телефонных номеров и условие диапазона — для поиска тех сотрудников, которые приняты на работу с октября по конец декабря. Теперь требуется только определить, как предполагается объединить эти условия.

Условие сравнения и условие совпадения с образцом необходимо объединить в операторе AND, потому что они определяют искомые номера телефонов, и оба условия должны быть удовлетворены, чтобы строка была включена в набор результатов. Эта совокупность условий затем интерпретируется как единый блок и объединяется с условием диапазона с помощью оператора OR. Теперь строка будет включаться в набор результатов только в том случае, если она будет удовлетворять *совокупному* условию, либо условию диапазона.

Вот снова запрос и преобразование:

> *"I need to see the names of stuff members who have a 425 area code and a phone number that begins with 555, along with anyone who was hired between October 1 and December 31 of 1999".*

Преобразование: Select first name, last name, area code, phone number, and date hired from the staff table where the area code is 425 and the phone number begins with '555' or the date hired falls between October 1, 1999, and December 31, 1999
(Выбрать имя, фамилию, междугородный телефонный код, номер телефона и дату приема на работу из таблицы "Персонал", где междугородный телефонный код — 425 и номер телефона начинается с '555' или дата приема на работу приходится между 1 октября 1999 и 31 декабря 1999)

Уточнение: Select first name, last name, area code, phone number, ~~and~~ date hired from ~~the~~ staff ~~table~~ where ~~the~~ area code ~~is~~ = 425 and ~~the~~ phone number ~~begins with~~ like '555%' or ~~the~~ date hired ~~falls~~ between ~~October 1, 1999~~ '1999-10-01' and ~~December 31, 1999~~ '1999-12-31'
(Выбрать имя, фамилию, междугородный телефонный код, номер телефона, дату приема на работу из "Персонал", где междугородный телефонный код = 425 и номер телефона соответствует шаблону '555%' или дата приема на работу между '1999-10-01' и '1999-12-31')

SQL
```
SELECT StfFirstName, StfLastName, StfAreaCode,
    StfPhoneNuTber, DateHired
FROM Staff
WHERE (StfAreaCode:' = '425'
    AND StfPhoneNumber LIKE '555%')
OR DateHired
    BETWEEN '1999-10-01' AND '1999-12-31'
```

Этот пример четко показывает ситуацию, когда можно использовать условие поиска внутри условия поиска. В данном случае необходимо объединить условие сравнения и условие совпадения образца в операторе AND, а затем интерпретировать его как единый блок. Когда объединенный набор условий рассматривается как единый блок, то по определению он становится условием поиска и его можно заключить в скобки (как и сделано в примере).

А теперь другой пример с использованием AND и OR:

> *"I need the name of every professor or associate professor who was hired on May 16, 1989".*
> *("Нужны имена всех профессоров и адъюнкт-профессоров, принятых на работу 16 мая 1989".)*

Преобразование: Select first name, last name, and date hired from the staff table where the title is 'professor' or 'associate professor' and the date hired is May 16, 1989
(Выбрать имя, фамилию и дату приема на работу из таблицы "Персонал" тех, у которых должность 'профессор' или 'адъюнкт-профессор' и дата приема на работу 16 мая 1989)

Уточнение: Select first name, last name, ~~and~~ date hired from ~~the~~ staff ~~table~~ where ~~the~~ title ~~is~~ = 'professor' or title = 'associate professor' and ~~the~~ date hired ~~is~~ = ~~May 16, 1989~~ '1989-05-16'
(Выбрать имя, фамилию, дату приема на работу из "Персонал" тех, для которых должность = 'профессор' или должность = 'адъюнкт-профессор' и дата приема на работу = '1989-05-16')

SQL

```
SELECT StfFirstName, StfLastName, Title, Datehired
FROM Staff
WHERE (Title = 'Professor' OR Title =
    'Associate Professor')
AND DateHired = '1989-05-16'
```

Вы уже вероятно догадались, что два условия, объединенные в операторе OR, интерпретируются как единое условие поиска. Этот пример просто подтверждает, что можно определить условие поиска либо с оператором AND, либо OR. Но основной момент состоит в том, чтобы заключить условие поиска в скобки.

Исключение строк: Второй подход

Если возникает легкое ощущение "дежа вю", не стоит тревожиться. Как вы уже знаете, оператор NOT является опцией предикатов BETWEEN, IN, LIKE и IS NULL. Но, как показано на рис. 6.11, NOT также является опцией как первое ключевое слово условия поиска, и оно позволяет исключить строки из набора результатов таким же образом, как при использовании NOT в предикате. Этот конкретный оператор NOT используется *перед* отдельным условием (предикатом) или вложенным условием поиска. И опять одно и то же условие можно выразить различными способами.

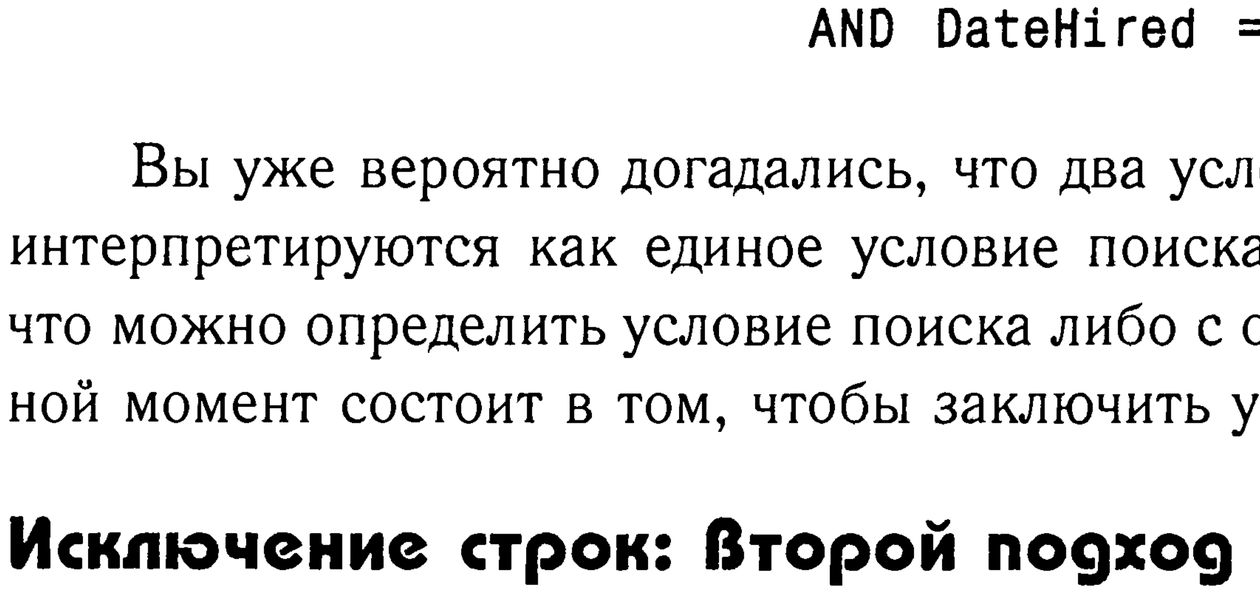


Рис. 6.11. *Включение оператора NOT в условие поиска*

Предположим, что к базе данных обращаются со следующим запросом:

> *"Show me the location and date of any tournament not being held at Bolero Lanes, Imperial Lanes, or Thunderbird Lanes".*
> *("Показать места проведения и даты всех турниров, которые проводятся не в Болеро Лэйнс, Империал Лэйнс или Зандербирд Лэйнс".)*

Вероятно, вы уже догадались, что для ответа на этот запрос будет использоваться условие принадлежности. Теперь просто требуется решить, как его определить. Одним из подходов является использование оператора NOT в предикате:

```
WHERE Tourney Location NOT IN ('Bolero Lanes',
    'Imperial Lanes', 'Thunderbird Lanes')
```

Другим подходом является использование оператора NOT как первого ключевого слова до условия поиска:

```
WHERE NOT TourneyLocatIon IN ('Bolero Lanes', 'Imperial Lanes',
    'Thunderbird Lanes')
```

Любое условие исключит из набора результатов турниры, которые проводятся в Болеро Лэйнс, Империал Лэйнс или Зандербирд Лэйнс. Однако одним из преимуществ использования NOT перед условием поиска является то, что его можно применить к условию сравнения. (Вспомните, что синтаксическая структура условия сравнения не включает NOT, как необязательный оператор.) Но теперь можно использовать условие сравнения для исключения строк из набора результатов. В следующем примере показано, как можно использовать этот тип условий:

> *"Show me the bowlers who live outside of Bellevue".*
> *("Показать игроков, не проживающих в Беллевью".)*

Преобразование: Select first name, last name, and city from the bowlers table where the city is not 'Bellevue'
(Выбрать имя, фамилию и город из таблицы "Игроки в боулинг", где город — не 'Беллевью')

Уточнение: Select first name, last name, ~~and~~ city from ~~the~~ bowlers ~~table~~ where ~~the~~ city ~~is~~ not = 'Bellevue'
(Выбрать имя, фамилию, город из "Игроки в боулинг", где город не = 'Беллевью')

SQL

```
SELECT BowlerFirstName, BowlerLastName, BowlerCity
From Bowlers
WHERE NOT BowlerCity = 'Bellevue'
```

Да, можно выразить это условие как `WHERE BowlerCity <> 'Bellevue'`. Этот пример просто подчеркивает, что условие можно выразить различными способами.

Примите во внимание проблему, которая может возникнуть, когда определяется условие поиска с двумя операторами NOT, которые будут *включать* строки вместо того, чтобы *исключать* их. Вот пример:

"Which staff members are not a teacher or a teacher's aide?"
("Кто из сотрудников не является преподавателем
или помощником преподавателя?")

Преобразование: Select first name, last name, and title from the staff table where the title is not "teacher" or "teacher's aide"
(Выбрать имя, фамилию и должность из таблицы "Персонал", где должность не "преподаватель" и не "помощник преподавателя")

Уточнение: Select first name, last name, ~~and~~ title from ~~the~~ staff ~~table~~ where ~~the~~ title ~~is~~ not in ("teacher" ~~or~~ "teacher's aide")
(Выбрать имя, фамилию, должность из "Персонал", где должность не из ("преподаватель", "помощник преподавателя"))

SQL
```
SELECT StfFirstName, StfLastNaffie, Title
FROM Staff
WHERE NOT Title
NOT IN ('Teacher', 'Teacher''s Aide')
```

Внимание! Поспорим, что удивление вызывают две одиночные кавычки в литерале типа символьной строки — `'Teacher''s Aide'`. Стандарт SQL диктует, что одиночная кавычка используется для отделения символьной строки или литерала дата/время. Когда необходимо вставить одиночную кавычку в литерал символьной строки, следует "информировать" о ней систему базы данных путем ввода одиночной кавычки два раза. Если этого не сделать, одиночная кавычка будет действовать как конечный ограничитель символьной строки. `"s Aide'"`, возникающее после второй одиночной кавычки, будет генерировать синтаксическую ошибку!

Предполагается, конечно, что один из двух операторов NOT появляется по ошибке. Можно выполнять этот оператор SELECT, но он будет отправлять неверные строки в набор результатов. В данном случае два оператора NOT аннулируют друг друга, как двойное отрицание в арифметике или в языке, а предикат IN теперь определяет, какие строки отправляются в набор результатов. Поэтому вместо поиска тех, кто *не является* преподавателем или помощником преподавателя, вы будете искать *только* преподавателей или помощников преподавателей. Хотя таким образом невозможно осознанно определить преподавателя или помощника преподавателя в условии поиска, это вполне можно сделать случайно. Это часто является просто ошибкой, вызывающей большинство проблем.

Порядок предшествования

Стандарт SQL определяет, как СУБД должна анализировать единые условия в условии поиска и порядок выполнения этой оценки. Из этой главы уже стало известно, *как* база данных анализирует каждый тип условий. Теперь покажем, как база данных определяет, *когда* анализировать каждое отдельное условие.

По умолчанию база данных анализирует условия слева направо. Это особенно верно в случае простых условий. В приведенном ниже примере оператор SELECT вначале осуществляет поиск строк, для которых дата поставки совпадает с датой заказа, а затем определяет, какие строки содержат клиента с номером 1001. Строки, которые удовлетворяют обоим условиям, отправляются в набор результатов.

SQL

```
SELECT CustomerID, OrderDate, ShipDate
FROM Orders
WHERE ShipDate = OrderDate
    AND CustomerID = 1001
```

Для того чтобы оператор SELECT выполнил поиск номера конкретного клиента прежде анализа даты поставки, просто переставьте позиции условий.

Когда условие поиска содержит различные типы отдельных условий, база данных оценивает их в конкретном порядке, исходя из *оператора*, используемого в каждом условии. Стандарт SQL определяет следующий порядок действий при анализе операторов (см. табл. справа).

Порядок анализа	Тип оператора
1	Знак плюс (+), знак минус (–)
2	Умножение (*), деление (/)
3	Сложение (+), вычитание (–)
4	=, <>, <, >, <=, >=, BETWEEN, IN, LIKE, IS NULL
5	NOT
6	AND
7	OR

В следующем примере оператор SELECT содержит такое условие поиска, которое заставляет систему БД обратиться к порядку предшествования. В данном случае база данных выполняет операцию сложения, осуществляет сравнения и определяет, удовлетворены ли все условия. Любая строка, которая удовлетворяет обоим условиям, заносится в набор результатов.

SQL

```
SELECT CustomerID, OrderDate, ShipDate
FROM Orders
WHERE CustomerID = 1001 OR
    ShipDate = OrderDate + 4
```

Расположение условий в порядке приоритетов

Можно значительно повысить точность условий поиска, если понимать порядок предшествования. Это поможет правильно сформулировать условие для рассматриваемого запроса. Но нужно быть осторожным и определять только однозначные условия, которые не могут дать непредвиденные результаты.

С помощью следующего примера рассмотрим эту потенциальную проблему:

```
SQL                SELECT CustFirstName, CustLastName, CustState,
                       CustZipCode
                   FROM Orders
                   WHERE CustLastName = 'Patterson'
                       AND CustState = 'CA'
                       OR CustZipCode LIKE '%9'
```

В этом случае трудно определить реальную цель условия поиска, потому что ее можно интерпретировать двумя способами:

1. Осуществляется поиск кого-то с именем "Patterson" из штата Калифорния *или* кого-то с почтовым кодом, который заканчивается на 9.
2. Осуществляется специальный поиск всех с именем "Patterson" *и* кого-то, кто проживает в Калифорнии или у кого почтовый код заканчивается на 9.

Можно избежать этой неоднозначности и сделать условие поиска более понятным, используя скобки для объединения и назначения приоритета определенным условиям. Например, чтобы получить первую интерпретацию условия поиска, нужно определить условие WHERE следующим образом:

```
WHERE (CustLastName = 'Patterson' AND CustState = 'CA')
    OR CustZipCode LIKE '%9'
```

Скобки гарантируют, что база данных проанализирует и оценит два условия сравнения *до* выполнения тех же процессов для условия совпадения с образцом.

Можно получить вторую интерпретацию и определить условие WHERE следующим образом:

```
WHERE CustLastName = 'Patterson' AND (CustState = 'CA'
    OR CustZipCode LIKE '%9')
```

В данном случае база данных анализирует и выполняет оценку первого условия сравнения *после* выполнения этих же процессов для второго условия сравнения и условия совпадения с образцом.

Смысл заключения условий в скобки должен быть хорошо вам понятен. Применение скобок может оказать серьезное влияние на результат условия поиска.

Можно использовать любое количество условий, заключенных в скобки, и даже вкладывать их. А вот как база данных обрабатывает условия в скобках:

- Условия в скобках обрабатываются до условий, не заключенных в скобки.
- Два или более условий в скобках обрабатываются слева направо.

- Вложенные условия в скобках обрабатываются от внутренних к внешним.

Как только база данных начинает анализировать некоторое условие в скобках, она оценивает все выражения в этом условии, используя обычный порядок предшествования. Если внимательно преобразовывать запрос и эффективно использовать скобки в условии поиска, можно достичь лучших результатов.

Чем меньше — тем лучше

База данных первоначально анализирует условия слева направо, и порядок предшествования применяется, когда определяются и используются сложные условия. Способ использования скобок в условии поиска оказывают прямое воздействие на результат. Дадим простой общий совет по ускорению процесса условия поиска: запрашивайте меньше, т. е. выбирайте только те столбцы, которые необходимы для выполнения запроса, и как можно больше конкретизируйте условия запроса, чтобы база данных обрабатывала наименьшее возможное количество строк. Если требуется использовать несколько условий, убедитесь в том, что условие, исключающее большинство строк из набора результатов, обрабатывается в первую очередь (именно здесь действительно полезно понимание порядка предшествования).

Продемонстрируем все это на примере, который уже использовался ранее:

```
SQL                SELECT CustomerID, OrderDate, ShipDate
                   FROM Orders
                   WHERE ShipDate = OrderDate
                      AND CustomerID = 1001
```

В данном случае строка должна выполнить оба условия для того, чтобы она была включена в набор результатов. Размещение предикатов в таком порядке может задать вначале поиск каждой даты поставки, которая совпадает с соответствующей датой заказа. В зависимости от количества строк в таблице базе данных может потребоваться некоторое время для оценки этого условия. Затем база данных будет осуществлять поиск строк, удовлетворяющих первому условию, чтобы определит, какие из них содержат идентификатор клиента, равный 1001.

Вот лучший способ для определения условия:

```
SQL                SELECT CustomerID, OrderDate, ShipDate
                   FROM Orders
                   WHERE CustomerID = 1001
                      AND ShipDate = OrderDate
```

Теперь база данных скорее всего вначале выполнит поиск идентификатора клиента. Это условие, вероятно, даст небольшое количество строк, и поэтому базе данных потребуется меньше времени для поиска тех строк, которые совпадают с предикатом даты поставки.

Этот метод следует сделать обычной практикой и применять его при определении условий поиска. Потребуется много времени, чтобы гарантировать быстрое и эффективное выполнение операторов SELECT. Обязательно изучите документацию на систему базы данных и узнайте, какие другие методы можно использовать для еще большей оптимизации оператора SELECT.

Повторная встреча с NULL: Предупреждающее замечание

Сейчас подходящий момент напомнить о Null. Как вы знаете, Null представляет собой отсутствие значения, и выражение, обрабатывающее значение Null, возвращает Null. Это справедливо также и для условий поиска. Предикат, который выполняет оценку значения Null, *может никогда не иметь значения True*. Мало того — он также может никогда не иметь значения False! Стандарт SQL определяет результат любого предиката, в состав операндов которого входит Null, как Unknown (Неизвестно). Вспомните, что для выборки строки предикат должен иметь значение True, поэтому при значениях False или Unknown она отклоняется.

		Второе выражение		
	AND	**Истина**	**Ложь**	**Неизвестно**
Первое выражение	**Истина**	Истина (Строки выбираются)	Ложь (Строки отвергаются)	Неизвестно (Строки отвергаются)
	Ложь	Ложь (Строки отвергаются)	Ложь (Строки отвергаются)	Ложь (Строки отвергаются)
	Неизвестно	Неизвестно (Строки отвергаются)	Ложь (Строки отвергаются)	Неизвестно (Строки отвергаются)

Рис. 6.12 *Результат объединения двух предикатов типа выражение в операторе AND*

Чтобы помочь прояснить вопрос, посмотрим еще раз на таблицы для значений "Истина" на рис. 6.12 и 6.13, которые были первоначально представлены на рис. 6.9 и 6.10. Но на этот раз включим результат Unknown (Неизвестно), который можно получить, если используется значение Null.

Можно видеть, что результат Unknown анализа предиката на столбце с Null в действительности усложняет картину! Например, предположим, что имеется простой предикат сравнения: A = B. Если либо A, либо B для данной строки имеет значение Null, то результат сравнения — Unknown.

Первое выражение \ Второе выражение	Истина	Ложь	Неизвестно
Истина	Истина (Строки выбираются)	Истина (Строки выбираются)	Истина (Строки выбираются)
Ложь	Истина (Строки выбираются)	Ложь (Строки отвергаются)	Неизвестно (Строки отвергаются)
Неизвестно	Истина (Строки выбираются)	Неизвестно (Строки отвергаются)	Неизвестно (Строки отвергаются)

OR

Рис. 6.13 *Результат объединения двух предикатов типа "выражение" в операторе OR*

Поскольку результат не True, то строка не будет выбрана. Если к тому же A = B не является True, то можно ожидать, что NOT (A = B) будет True. НЕТ! Это также будет Unknown. Рис. 6.14 поможет понять, почему это так.

(Выражение)	NOT (Выражение)
True	**False**
False	**True**
Unknown	**Unknown**

Рис. 6.14. *Результат применения NOT к значению True/False/Unknown*

Предположим, что к базе данных обращаются со следующим запросом:

> *"Let me see the names and phone numbers of King county residents whose last name is Hernandez".*
> *("Показать имена и номера телефонов лиц, постоянно проживающих в округе Кинг, фамилия которых Hernandez".)*

Преобразование: Select first name, last name, and phone number
from the customers table where the county name is 'King'
and the last name is 'Hernandez'
(Выбрать имя, фамилию и номер телефона
из таблицы "Клиенты", где название округа 'Кинг'
и фамилия 'Hernandez'.

Уточнение: Select first name, last name, ~~and~~ phone number
from ~~the~~ customers ~~table~~ where ~~the~~ county ~~name~~
~~is~~ = 'King' and ~~the~~ last name ~~is~~ = Hernandez
(Выбрать имя, фамилию, номер телефона из "Клиенты",
где графство = 'Кинг' и фамилия = 'Hernandez')

SQL
```
SELECT CustFirstName, CustLastName, CustPhoneNumber
FROM Customers
WHERE CustLastName = 'Hernandez'
    AND CustCounty = 'King'
```

Для того чтобы строка была включена в набор результатов, она должна удовлетворять *обоим* условиям. Если название округа, либо фамилия имеют значение Null, база данных полностью пренебрегает этой строкой.

Рассмотрим следующий запрос:

"Show me the names of all staff members who are graduate counselors or were hired on September 1, 1999".
("Показать имена всех штатных сотрудников из таблицы "Персонал", которые являются дипломированными адвокатами или приняты на работу 1 сентября 1999".)

Преобразование: Select last name and first name from the staff table where the title is 'graduate counselor' or date hired is September 1,1999
(Выбрать фамилию и имя из таблицы "Персонал", где должность 'дипломированный адвокат' или дата приема на работу 1 сентября 1999)

Уточнение: Select last name ~~and~~ first name from ~~the~~ staff ~~table~~ where ~~the~~ title ~~is~~ = 'graduate counselor' or date hired ~~is~~ = ~~September 1, 1999~~ '1999-09-01'
(Выбрать фамилию и имя из "Персонал", где должность = 'дипломированный адвокат' или дата приема на работу = '1999-09=01')

SQL
```
SELECT StfLastName, StfFirstName
FROM Staff
WHERE Title = 'Graduate Counselor'
    OR DateHired = '1999-09-01'
```

Можно подумать, что значения Null должны оказывать такое же влияние на условия, объединенные в OR, как и на условия, объединенные в AND, но на самом деле это не так. У строки все еще остается шанс, что она будет включена в набор результатов, пока она удовлетворяет *любому* из этих условий. Посмотрите еще раз на рис. 6.13. На основании значений Title (Должность) и DateHired (Дата принятия на работу) в таблице 6.2 представлено, как база данных определяет, заносить ли строку в набор результатов, когда предикаты объединяются с OR.

Таблица 6.2

Определение набора результата и OR

Название должности	Значение DateHired	Результат
Дипломированный адвокат	**1999-09-01**	Строка включается в набор результатов, поскольку удовлетворяет обоим условиям.
Дипломированный адвокат	1999-11-15	Строка включается в набор результатов, поскольку удовлетворяет первому условию.
Секретарь учебного заведения	**1999-09-01**	Строка включается в набор результатов, поскольку удовлетворяет второму условию.
Дипломированный адвокат	Null	Строка включается в набор результатов, поскольку удовлетворяет первому условию.
Null	**1999-09-01**	Строка включается в набор результатов, поскольку удовлетворяет второму условию.
Null	Null	Строка исключается из набора результатов, поскольку не удовлетворяет ни одному из условий.

Когда предполагается, что набор результатов отображает неверную информацию, следует проверить все столбцы, используемые как критерий, на условие Null. Это даст возможность обработать все значения Null надлежащим образом, а затем можно снова выполнить исходный оператор SELECT. Например, если вы полагаете, что в наборе результатов могут быть пропущены несколько дипломированных адвокатов, можно выполнить следующий оператор SELECT:

```
SQL                SELECT StfLastName, StfFirstName, Title
                   FROM Staff
                   WHERE Title IS NULL
```

Если в столбце Title имеется значение Null, этот оператор SELECT выдаст набор результатов, содержащий имена всех штатных сотрудников, для которых в базе данных не указана должность. Теперь можно рассмотреть эти данные по обстановке, а затем возвратиться к исходному оператору SELECT.

Проблема Null все еще не решена жюри. В главе 12 мы вновь встретимся со значениями Null при обсуждении операторов SELECT, которые суммируют данные.

Выражение условий различными способами

Одним из преимуществ является то, что можно выразить указанное условие различными способами. Проверим это, рассматривая следующий запрос:

> *"Give me the name of every employee that was hired in October of 1999". ("Предоставить имена всех сотрудников, которые были приняты на работу 1 октября 1999".)*

Для того чтобы ответить на этот запрос, нужно отыскать даты приема на работу, которые попадают между 1 октября и 31 октября 1999 г. Это условие можно определить двумя способами:

```
DateHired BETWEEN '1999-10-01' AND '1999-10-31'
DateHired >= '1999-10-01' AND DateHired <= '1999-10-31'
```

Каждое из этих условий отправит одни и те же строки в набор результатов. Какое условие выбрать для использования — это просто вопрос предпочтения. Некоторые находят, что первое выражение легче понимать, тогда как другие предпочитают второе выражение.

Вот несколько других примеров условий равенства.

"Show me the venders who are based in California, Oregon, or Washington". ("Показать поставщиков, которые размещаются в Калифорнии, Орегоне или Вашингтоне".)

```
VendState IN ('CA', 'OR', 'WA')
VendState = 'CA' OR VendState = 'OR' OR VendState = 'WA'
```

"Give me a list of customers whose last name begins with 'H'". ("Предоставить список клиентов, фамилии которых начинаются с 'H'".)

```
CustLastName >= 'H' AND CustLastName <= 'HZ'
CustLastName BETWEEN 'H' AND 'HZ'
CustLastName LIKE 'H%'
```

"Show me all the students who do not live in Seattle or Redmond". ("Показать всех студентов, не проживающих в Сиэтле или Редмонде".)

```
StudCity <> 'Seattle' AND StudCity <> 'Redmond'
StudCity NOT IN ('Seattle', 'Redmond')
NOT (StudCity = 'Seattle' OR StudCity = 'Redmond')
```

Мы не можем "неверно" определить условие, но можем сделать его непригодным для работы, грубо игнорируя синтаксис (это вызовет ошибку при проверке условия). Однако некоторые СУБД оптимизируют определенные типы условий для ускорения обработки, делая их предпочтительнее других эквивалентных условий. Обратитесь к документации по своей СУБД и выясните, имеются ли какие-либо предпочтительные методы определения условий.

Примеры операторов

Рассмотрим некоторые примеры различных типов условий поиска, используя таблицы каждой из учебных баз данных.

> **Внимание!** Мы снова объединили этапы преобразования и уточнения во всех примерах, чтобы вы поняли, как объединять процесс.

База данных заказов

"Show me all the orders for customer number 1001".
("Показать все заказы для клиента номер 1001".)

Преобразование/ Уточнение:
Select ~~the~~ order number ~~and~~ customer ID from ~~the~~ orders ~~table~~ where ~~the~~ customer ID ~~is~~ = 1001
(Выбрать номер заказа, идентификатор клиента из "Заказы", где идентификатор клиента = 1001)

SQL

```
SELECT OrderNumber, CustomerID
FROM Orders
WHERE CustomerID = 1001
```

Order_for_Customer_1001 (44 строки)

OrderNumber	CustomerID
2	1001
7	1001
16	1001
52	1001
55	1001
107	1001
137	1001
138	1001
151	1001
154	1001
<< остальные строки >>	

"Show me an alphabetized list of products with names that begin with 'Dog'".
("Показать список товаров в алфавитном порядке с наименованиями, которые начинаются с 'Dog'".)

Преобразование/ Уточнение:

Select ~~the~~ product name from ~~the~~ products ~~table~~ where ~~the~~ product name like 'Dog%' ~~and~~ order by product name
(Выбрать наименование продукта из "Продукты", где наименование продукта соответствует шаблону 'Dog%', упорядочить по наименованию продукта)

SQL

```
SELECT ProductName
FROM Products
WHERE ProductName LIKE 'Dog%'
ORDER BY ProductNam
```

Внимание! Просто напоминаем, что условие ORDER BY помещается *в конец* оператора SELECT. Если необходимо, повторно просмотрите раздел "Сортировка информации" в главе 4.

Products_That_Begin_With_DOG (4 строки)

ProductName
Dog Ear Aero-Flow Floor Pump
Dog Ear Cyclecomputer
Dog Ear Helmet Mount Mirrors
Dog Ear Monster Grip Gloves

База данных эстрадных мероприятий

"Show me an alphabetical list of entertainers based in Bellevue, Redmond, or Woodinville".
("Показать список эстрадных артистов, проживающих в Беллевью, Редмонде или Вудинвилле".)

Преобразование/ Уточнение:

Select stage name, phone number ~~and~~ city from ~~the~~ entertainers ~~table~~ where ~~the~~ city ~~is~~ in ('Bellevue', 'Redmond' ~~or~~ 'Woodinville') ~~and~~ order by stage name
(Выбрать наименование площадки, номер телефона, город из "Эстрадные артисты", где город в ('Беллевью', 'Редмонд', 'Вудинвилль'), упорядоченные по наименованию площадки)

SQL

```
SELECT EntStageName, EntPhoneNumber, EntCity
FROM Entertainers
WHERE EntCity IN ('Bellevue', 'Redmond','Woodinville')
ORDER BY EntStageName
```

Eastside_Entertainers (7 строк)

EntStageName	EntPhoneNumber	EntCity
Albert Buchanan	555-2531	Bellevue
Carol Peacock Trio	555-2691	Redmond
Jazz Persuasion	555-2541	Bellevue
JV & the Deep Six	555-2511	Redmond
Katherine Ehrlich	555-0399	Woodinville
Modern Dance	555-2631	Woodinville
Susan McLain	555-2301	Bellevue

"Show me all the engagements that run for four days".
("Показать все ангажементы, которые заключены на четыре дня".)

Преобразование/ Уточнение:

Select engagement number, start date, ~~and~~ end date
from ~~the~~ engagements ~~table~~ where the
(end date ~~minus~~ – start date) ~~equals~~ = 3
(Выбрать номер ангажемента, дату начала,
дату окончания из "Ангажементы",
где (дата окончания – дата начала) = 3)

Four-Day_Engagements (16 строк)

EngagementNumber	StartDate	EndDate
1	1999-07-01	1999-07-04
5	1999-07-11	1999-07-14
13	1999-07-17	1999-07-20
17	1999-07-29	1999-08-01
21	1999-07-30	1999-08-02
56	1999-09-24	1999-09-27
58	1999-09-30	1999-10-03
59	1999-09-30	1999-10-03
63	1999-10-17	1999-10-20
70	1999-10-22	1999-10-25
<< остальные строки >>		

```
SQL                SELECT EngagementNumber, StartDate, EndDate
                   FROM Engagements
                   WHERE (EndDate - StartDate) = 3
```

Внимание! Ангажементы продолжаются от даты начала до даты окончания. При вычитании StartDate из EndDate получим на один день меньше, чем общее количество дней ангажемента. По этой причине результат сравнивается с 3, а не с 4.

База данных расписания занятий

"Show me an alphabetical list of all the staff members and their salaries if they make between $40 000 and $50 000 a year".
("Показать в алфавитном порядке список всех штатных сотрудников и их оклады, если они находятся между 40 000 и 50 000 долл. в год".)

Преобразование/ Уточнение:

Select first name, last name, and salary from ~~the~~ staff ~~table~~ where ~~the~~ salary ~~is~~ between 40 000 and 50 000, ~~then~~ order by last name, ~~and~~ first name
(Выбрать имя, фамилию, оклад из "Персонал", где оклад между 40 000 и 50 000, упорядочить по фамилии, имени)

```
SQL                SELECT StfFirstName, StfLastName, Salary
                   FROM Staff
                   WHERE Salary BETWEEN 40000 AND 50000
                   ORDER BY StfLastName, StfFirstName
```

Staff_Salary_40K_TO_50K (14 строк)

StfLastName	StfFirstName	Salary
Buchanan	Albert	$45,000.00
Buchanan	Amelia	$48,000.00
Callahan	David	$50,000.00
Callahan	Laura	$45,000.00
Ehrlich	Katherine	$45,000.00
Fuller	Ann	$44,000.00
Leverling	Janet	$50,000.00
Maynez	Consuelo	$48,000.00
Patterson	Ann	$45,000.00
Piercy	Gregory	$45,000.00
<< остальные строки >>		

"Show me a list of students whose last name is 'Kennedy' or who live in Seattle".
("Показать список студентов, фамилия которых "Кеннеди" или которые живут в Сиэтле".)

Преобразование/ Уточнение:

Select first name, last name, ~~and~~ city from ~~the~~ students ~~table~~ where ~~the~~ last name ~~is~~ = 'Kennedy' or ~~the~~ city ~~is~~ = 'Seattle'
(Выбрать имя, фамилию, город из "Студенты", где фамилия = 'Kennedy' или город = 'Seattle')

SQL

```
SELECT StdFirstName, StdLastName, StdCity
FROM Students
WHERE StdLastName = 'Kennedy' OR StdCity = 'Seattle'
```

Seattle_Students_And_Students_Named_Kennedy (5 строк)

StudFirstName	StudLastName	StudCity
Sally	Callahan	Seattle
Sara	Kennedy	Portland
John	Kennedy	Portland
Kendra	Bonnicksen	Seattle
David	Nathanson	Seattle

База данных лиги игроков в боулинг

"List the ID numbers of the teams that won one or more of the first ten matches in Game 3".
("Привести список идентификационных номеров команд, которые выиграли одну или более из первых десяти партий в игре 3".)

Преобразование/ Уточнение:

Select ~~the~~ team ID, match ID, ~~and~~ game number from ~~the~~ match_games ~~table~~ where ~~the~~ game number ~~is~~ = 3 and ~~the~~ match ID ~~is~~ between 1 and 10
(Выбрать идентификатор команды, идентификатор матча, номер игры из "Партии/Игры", где номер игры = 3 и идентификатор партии между 1 и 10)

SQL

```
SELECT WinningTeamID, MatchID, GameNumber
FROM Match_Games
WHERE GameNumber = 3 AND MatchID BETWEEN 1 AND 10
```

Game3_Top_Ten_Matches (10 строк)

WinningTeamID	MatchID	GameNumber
1	1	3
3	2	3
5	3	3
7	4	3
3	5	3
4	6	3
5	7	3
8	8	3
2	9	3
1	10	3

"List the bowlers in teams 3, 4, and 5 who have a handicap of 40 or less".
("Предоставить список игроков в боулинг команд 3, 4 и 5, у которых гандикап 40 или меньше".)

Преобразование/ Select first name, last name, team ID, ~~and~~ current
Уточнение: handicap from ~~the~~ bowlers ~~table~~ where ~~the~~ team ID ~~is either~~ in (3, 4, ~~or~~ 5) and ~~the~~ current handicap ~~is less than or equal to~~ <= 40
(Выбрать имя, фамилию, идентификатор команды, текущий гандикап из "Игроки в боулинг", где идентификатор команды из (3, 4 или 5) и текущий гандикап <= 40)

SQL
```
SELECT BowlerFirstName, BowlerLastName, TeamID,
    CurrentHandicap
FROM Bowlers
WHERE TeamID IN (3, 4, 5) AND CurrentHandicap <= 40
```

Low_Handicap_Bowlers_Teams_3_Through_5 (6 строк)

BowlerFirstName	BowlerLastName	TeamID	CurrentHandicap
David	Cunningham	3	36
Susan	McLain	3	33
Gary	Hallmark	4	39
Kathryn	Patterson	4	34
Michael	Hernandez	5	39
John	Viescas	5	29

Итоги

В данной главе обсуждалось понятие фильтрации информации, получаемой в наборе результатов, посредством использования условия поиска в условии WHERE. Условие поиска использует комбинации предикатов для фильтрации данных, которые направляются в набор результатов. Предикаты являются специальными тестами, которые можно применять к типизированным выражениям. Можно определить пять основных типов предикатов.

Вы узнали, как сравнивать значения и проверять, попадает ли некоторое значение в указанный диапазон, совпадает ли значение с некоторым из значений, определенным в списке, или оно является частью конкретного образца строки. Для исключения строк из набора результатов можно использовать оператор NOT.

Можно объединять несколько условий в операторах AND и OR. Строка должна удовлетворять всем условиям, объединенным в AND, прежде чем она может быть включена в набор результатов, и должна удовлетворять *только одному* из условий, если они объединены в OR. Для ответов на сложные запросы можно использовать AND и OR вместе. NOT может использоваться на двух различных уровнях в условии поиска.

База данных анализирует условия в определенном порядке, исходя из операторов, используемых в каждом условии. Для изменения порядка, в котором БД анализирует условия, и для гарантии того, что не были определены неоднозначные условия, используются скобки.

При повторном рассмотрении значений Null мы показали, что Null во многом оказывает такое же влияние на условия, как и на выражения. Необходимо проверить наличие значений Null, если вы подозреваете, что в наборе результатов отражается неверная информация.

Одно и то же условие можно выразить различными способами. Например, можно использовать три различных типа условий для поиска лица, фамилия которого начинается с буквы “H”.

В следующей главе мы рассмотрим понятие *множеств* и типы операций, которые можно над ними выполнять. Изучив множества, вы сможете определять операторы SELECT, используя несколько таблиц.

Задачи для самостоятельного решения

Ниже приводятся формулировки запросов и имена решений этих запросов в учебных базах данных. Попрактикуйтесь и разработайте SQL для каждого запроса, а затем сверьте свой ответ с запросом, который сохранен нами в этих базах данных. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов,— важно, чтобы набор результатов был тем же.

База данных заявок на закупку

1. *"Give me the names of all vendors based in Ballard, Bellevue, and Redmond"*.
   ("Предоставить имена всех поставщиков, расположенных в Бэлларде, Беллевью и Редмонде".)
   Решение можно найти в Ballard_Bellevue_Redmond_Vendors (3 строки).

2. *"Show me an alphabetized list of products with a retail price of $125.00 or more"*.
   ("Показать список товаров в алфавитном порядке с розничной ценой от 125,00 долл. и выше".)
   (*Совет*: Можно расположить список в алфавитном порядке, используя условие, рассмотренное в предыдущей главе.)
   Решение можно найти в Product_Priced_Over_125 (13 строк).

3. *"Which vendors do we work with that don't have a Web site"*.
   ("У кого из поставщиков, с которыми мы сотрудничаем, отсутствует Web-сайт?")
   Решение можно найти в Vendors_With_No_Website (4 строки).

База данных эстрадных мероприятий

1. *"Let me see a list of all engagements that occurred during August of 1999"*.
   ("Предоставить список всех ангажементов, заключенных за август 1999".)
   Решение можно найти в August_1999_Engagements (21 строка).

2. *"Show me any engagements in August of 1999 that start between noon and 5 PM"*.
   ("Показать ангажементы за август 1999 г., которые начинались между 12:00 и 17:00".)
   Решение можно найти в August_Dates_Beetween_Noon_and_Five (17 строк).

3. *"List all the engagements that start and end on the same day"*.
   ("Предоставить список всех ангажементов, которые начинаются и заканчиваются в один и тот же день".)
   (*Совет*: Для ответа на этот запрос воспользуйтесь арифметикой для дат.)
   Решение можно найти в Single_Day_Engagements (6 строк).

База данных расписания занятий

1. *"Show me which staff members use a post office box as their address"*.
   ("Показать, кто из штатного персонала использует абонентский ящик как свой адрес".)
   Решение можно найти в Staff_Using_POBoxes (6 строк).

2. *"Can you show me which students live outside of the Pacific Northwest?"*
("Можно ли показать, кто из студентов живет не в Пасифик Нортвест?")
Решение можно найти в Students_Residing_Outside_PNW (5 строк).

3. *"List all the subjects that have a subject code starting 'MUS'"*.
("Привести список всех предметов, для которых код начинается с 'MUS'".)
Решение можно найти в Subjects_With_MUS_In_SubjectCode (4 строки).

База данных лиги игроков в боулинг

1. *"Give me a list of the tournaments held during August 1999".*
("Дать список турниров, проведенных начиная с августа 1999".)
Решение можно найти в August_1999_Tournament_Schedule (4 строки).

2. *"What are the tournaments schedules for Bolero, Red Rooster, and Thunderbird Lanes?"*.
("Какой график проведения турниров для Болеро, Ред Рустер и Зандербирд Лэйнс?")
Решение можно найти в Eastside_Tournaments (6 строк).

3. *"List the bowlers who live on the eastside (you know — Bellevue, Bothell, Duvall, Redmond, and Woodinvill) and whose handicap is between 45 and 55".*
("Привести список игроков в боулинг, которые проживают в восточной части (вы знаете — Беллевью, Бозель, Дюваль, Редмонд и Вудинвилль) и гандикап которых находится между 45 и 55".)
Решение можно найти в High_Handicap_Eastside_Bowlers (7 строк).

Часть III

Работа с несколькими таблицами

7

Мышление множествами

"Легкая улыбка и радушный прием
создают восхитительный праздник".

— Вильям Шекспир
Комедия ошибок, акт III

Вопросы, рассматриваемые в данной главе:

- Что такое множество
- Операции над множествами
- Пересечения
- Разность
- Объединение
- Операции над множествами в SQL
- Итоги

К настоящему моменту вы узнали, как создать набор информации, обращаясь с запросом к конкретным столбцам или выражениям со столбцами (SELECT), как отсортировать строки (ORDER BY) и как ограничить возвращенные строки (WHERE). До сих пор наше внимание было сосредоточено на упражнениях, использующих одну таблицу. Но, что если понадобится информация, которая содержится в нескольких таблицах? Что если нужно сравнить или сопоставить информацию из одной и той же или из разных таблиц?

Можно получать удовольствие от чистки картофеля или пучка моркови и нарезки его кружочками или кубиками. С этого момента большинство задач будет включать получение данных из *нескольких* таблиц. Мы не только собираемся показать вам, как приготовить хорошее тушеное блюдо — мы собираемся научить вас, как стать шеф-поваром!

Концепции, изложенные в данной главе, необходимо понимать для того, чтобы уметь успешно объединять два или более наборов информации. Мы дадим здесь краткий обзор некоторых специальных синтаксических структур, установленных в стандарте SQL, которые непосредственно поддерживают чистое определение этих концепций. Однако многие современные коммерческие реализации SQL пока еще

не поддерживают этот "чистый" синтаксис. В последующих главах мы покажем, как реализовать эти концепции, используя синтаксис SQL, который поддерживается большинством основных СУБД. После этого мы станем не "буквой закона", но, скорее, "духом закона".

Что такое множество

Если вы учились в физико-математической школе, вы изучали теорию множеств. Если вы знакомились с алгеброй множеств, то, вероятно, задумывались, пригодится ли это вам когда-нибудь.

Теперь же вы пытаетесь изучить реляционные базы данных и этот необычный язык, названный SQL, для построения приложений, решения задач или просто для получения ответов на свои вопросы. Если вы были внимательны на занятиях алгеброй, решение задач, особенно сложных, в SQL будет намного легче.

Как ни удивительно, но вы работаете с множествами с самого начала этой книги. Из главы 1 вы узнали об основной структуре реляционной базы данных — таблицах, содержащих записи, которые составлены из одного или нескольких полей. (Вспомним, что в SQL записи называются строками, а поля — столбцами.) Каждая таблица в базе данных представляет собой *множество* информации об одном предмете. И каждая таблица должна содержать *набор* информации, связанной с одним и только одним предметом или действием.

В главе 4 мы объяснили, как построить основной оператор SELECT в SQL для извлечения *набора* результатов с информацией, которая содержит конкретные столбцы одной таблицы, и как отсортировать такой набор результатов. В главе 5 мы показали, как собрать новое *множество* информации из таблицы, записывая выражения, которые оперируют с одним или несколькими столбцами. Из главы 6 вы узнали, как можно ограничить *множество* информации, извлеченной из таблиц, добавляя к запросу фильтр (условие WHERE).

Как можно видеть, множество может быть совсем небольшим, как данные одного столбца из одной строки в одной таблице. Фактически в SQL можно построить запрос, который вовсе не возвращает строк,— пустое множество (иногда оно полезно и помогает обнаружить, что что-то *не существует*). Множество также может состоять из нескольких столбцов (включая столбцы, созданные в выражениях) нескольких строк, извлеченных из нескольких таблиц. Каждая строка в наборе результатов является *элементом* множества. Значения в столбцах являются специальными *атрибутами* каждого элемента — элементами данных, которые описывают элемент множества.

Операции над множествами

Как уже говорилось, д-р Э. Ф. Кодд придумал реляционную модель, на которой основываются большинство современных баз данных и SQL. В основе этой модели использованы две ветви математики — теория множеств и исчисление предикатов первого порядка.

Вам необходимо изучить, как использовать наборы результатов с информацией для решения сложных задач. Такие задачи обычно требуют использования одной из обычных операций над множествами для связывания данных из двух или нескольких таблиц. Иногда нужно получить два различных набора результатов из одной и той же таблицы, а затем объединить их для получения ответа.

Три наиболее общих операции над множествами следующие:

- **Пересечение** — Используется для выделения общих элементов в двух или более различных множествах. "Покажите рецепты, в которые входят как мясо молодого барашка, так *и* рис". "Покажите клиентов, заказавших как мотоциклы, так *и* шлемы".
- **Разность** — Используется для поиска элементов, которые имеются в одном множестве, но отсутствуют в другом. "Покажите мне рецепты, в которые входит мясо молодого барашка, *но не* нужен рис". "Покажите мне клиентов, заказавших мотоциклы, *но не* заказавших шлемы".
- **Объединение** — Используется для объединения двух или более подобных множеств. "Покажите мне рецепты, в которые входят *либо* мясо молодого барашка, *либо* рис". "Покажите мне клиентов, заказавших *либо* мотоциклы, *либо* шлемы".

В следующих трех разделах поясняются эти основные операции над множествами. В разделе "SQL-операции над множествами" дается обзор того, как эти операции реализуются в "чистом" SQL.

Пересечение

Пересечение двух множеств содержит общие элементы из этих двух множеств. Рассмотрим пересечение как с точки зрения теории множеств, так и с точки зрения решения коммерческих задач.

Пересечение в теории множеств

Пересечение является очень мощным математическим инструментом, часто используемым учеными и инженерами. Ученый может быть заинтересован в том, чтобы найти общие точки двух множеств в выборках химических или физических данных. Например, ученый-химик, занимающийся исследованиями в области фармацевтики, может располагать двумя соединениями, которые оказывают определенное благоприятное воздействие. Обнаружение общих черт (пересечение) у этих двух соединений может помочь открыть, что именно обеспечивает их эффективность. Инженер может искать точку пересечения для двух сплавов, один из которых твердый, но хрупкий, а второй — мягкий, но эластичный.

Рассмотрим пересечение в действии, исследуя два множества чисел. В этом примере каждое отдельное число является элементом множества. Первое множество выглядит следующим образом:

1, 5, 8, 9, 32, 55, 78

Второе множество чисел следующее:

3, 7, 8, 22, 55, 71, 99

Пересечением этих двух множеств являются числа, имеющиеся в каждом из множеств. Ответ:

8, 55

Индивидуальные вхождения — элементы — каждого множества не обязательно должны быть просто отдельными значениями. При решении задач в SQL вы, вероятно, имели дело с множествами строк.

Согласно теории множеств, когда элемент множества является чем-то большим, чем просто число или значение, то каждый элемент (или объект) множества имеет несколько атрибутов или бит данных, которые описывают свойства каждого элемента. Например, ваш любимый рецепт тушеного мяса — это сложный элемент множества рецептов, который включает в себя множество различных компонентов. Каждый компонент является атрибутом вашего сложного элемента — рецепта тушеного мяса.

Для того чтобы найти пересечение двух сложных элементов множества, необходимо найти элементы, которые совпадают по всем атрибутам. Также все элементы каждого множества, которые сравниваются, должны иметь одинаковое количество и тип атрибутов. Предположим, что имеется сложное множество, подобное представленному ниже, в котором каждая строка является элементом множества (рецепта тушеного мяса), а каждый столбец обозначает конкретный атрибут (компонент).

Картофель	Вода	Мясо молодого барашка	Горошек
Рис	Куриный бульон	Курица	Морковь
Паста	Вода	Тофу	Молодой горошек в стручках
Картофель	Бульон из говядины	Говядина	Капуста
Паста	Вода	Свинина	Лук

Второе множество может выглядеть следующим образом:

Картофель	Вода	Мясо молодого барашка	Лук
Рис	Куриный бульон	Индюшатина	Морковь
Паста	Отвар из овощей	Тофу	Молодой горошек в стручках
Картофель	Бульон из говядины	Говядина	Капуста
Бобы	Вода	Свинина	Лук

Пересечением этих двух множеств является один элемент, все атрибуты которого совпадают в обоих множествах:

Картофель	Бульон из говядины	Говядина	Капуста

Пересечение наборов результатов

Если предыдущие примеры имели для вас вид строк в таблице или в наборе результатов, то вы на верном пути! Когда вы имеете дело со строками из набора данных, извлеченных с помощью SQL, то атрибутами являются отдельные столбцы. Предположим, например, что имеется множество строк, возвращенных запросом (это рецепты из поваренной книги Джона):

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная баранина	Картофель	Вода	Баранина	Горошек
Тушеная курица	Рис	Куриный бульон	Курица	Морковь
Вегетарианские тушеные овощи	Паста	Вода	Тофу	Молодой горошек в стручках
Тушеное мясо по-ирландски	Картофель	Бульон из говядины	Говядина	Капуста
Тушеная свинина	Паста	Вода	Свинина	Лук

Набор результатов второго запроса может иметь следующий вид (это рецепты из поваренной книги Майкла):

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная баранина	Картофель	Вода	Баранина	Горошек
Тушеная индейка	Рис	Куриный бульон	Индейка	Морковь
Вегетарианские тушеные овощи	Паста	Овощной отвар	Тофу	Молодой горошек в стручках
Тушеное мясо по-ирландски	Картофель	Бульон из говядины	Говядина	Капуста
Тушеная свинина	Бобы	Вода	Свинина	Лук

Пересечением этих двух множеств являются два элемента, атрибуты которых совпадают в обоих множествах, т. е. это два рецепта, которые одновременно имеются у Майкла и Джона:

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная баранина	Картофель	Вода	Баранина	Горошек
Тушеное мясо по-ирландски	Картофель	Бульон из говядины	Говядина	Капуста

Иногда легче понять, как работает пересечение, используя представление множеств в виде диаграмм. Это элегантный и вместе с тем простой способ графического отображения множеств информации и графического представления того, как множества пересекаются или налагаются. Возможно, вы слышали о диаграммах Эйлера-Венна. (Леонард Эйлер — швейцарский математик XVIII в., а Джон Венн использовал этот конкретный тип логических диаграмм в 1880 г. Поэтому понятно, что "мышление множествами" не является какой-то особенной современной концепцией!)

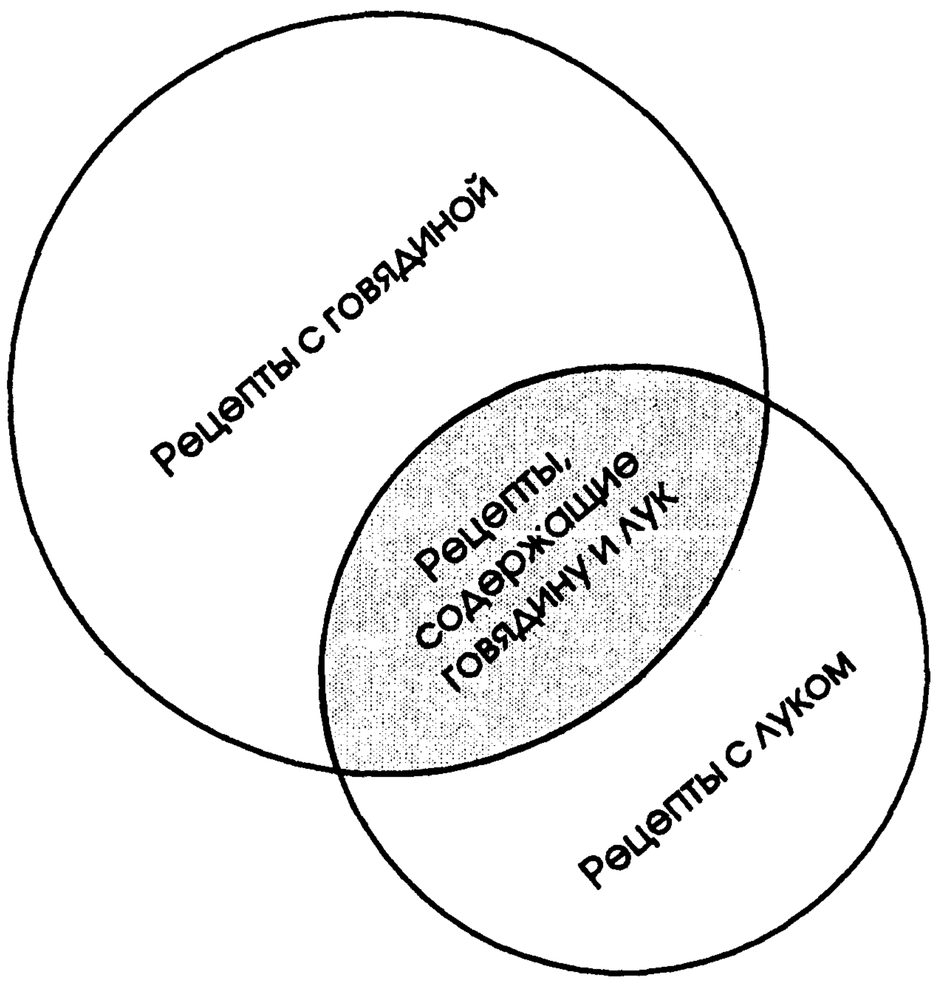


Рис. 7.1. *Поиск рецептов, содержащих как говядину, так и лук*

Предположим, имеется удачно выполненная база данных, состоящая из всех ваших любимых рецептов. Вам чрезвычайно нравится, как лук улучшает вкус говядины, поэтому хотелось бы найти все рецепты, содержащие как говядину, так и лук. На рис. 7.1 представлена диаграмма для множества, которая помогает наглядно представить решение этой задачи.

Верхний круг представляет собой множество рецептов, в которых используется говядина. Нижний круг представляет множество рецептов, которые содержат лук. Там, где два круга перекрываются, находятся рецепты, включающие оба компонента. Вначале в SQL извлекаются все рецепты, содержащие говядину. Во втором запросе SQL извлекает все рецепты, в которых есть лук. Можно использовать специальное ключевое слово SQL — INTERSECT,— чтобы скомпоновать два запроса для получения окончательного ответа.

Если ваша таблица с рецептами имеет такой вид, как представленные выше примеры, то можно просто сделать запрос:

> *"Show me the recipes that have beef as the meat ingredient and onions as the vegetable ingredient".*
> *("Показать рецепты, в которых мясной компонент — это говядина, а овощной компонент — лук".)*

Преобразование: Select the recipe name from the recipes table where meat ingredient is beef and vegetable ingredient is onions
(Выбрать наименования рецептов из таблицы "Рецепты", где мясной компонент — говядина, а овощной компонент — лук)

Уточнение: Select ~~the~~ recipe name from ~~the~~ recipes ~~table~~ where meat ingredient ~~is~~ = beef and vegetable ingredient ~~is~~ = onions

```
                    (Выбрать наименования рецептов из "Рецепты",
                    где мясной компонент = говядина,
                    овощной компонент = лук)

SQL                 SELECT RecipeName
                    FROM Recipes
                    Where MeatIngredient = 'Beef'
                        AND VegetableIngredient = 'Onions'
```

Однако если вы помните содержание главы 2, то понимаете, что одна таблица "Рецепты", вероятно, не даст этого. А что же относительно рецептов, которые содержат другие компоненты, а не мясо и овощи? Ведь некоторые рецепты содержат множество компонентов, а другие всего лишь несколько. Правильно спроектированная база данных рецептов включает отдельную таблицу Recipe_Ingredients (Компоненты_рецептов) с одной строкой на каждый компонент рецепта. Каждая строка компонента будет содержать только один компонент, поэтому не может быть отдельной строки с говядиной и луком одновременно. Вначале потребуется найти все строки с говядиной, затем все строки с луком и, наконец, их пересечение на RecipeID.

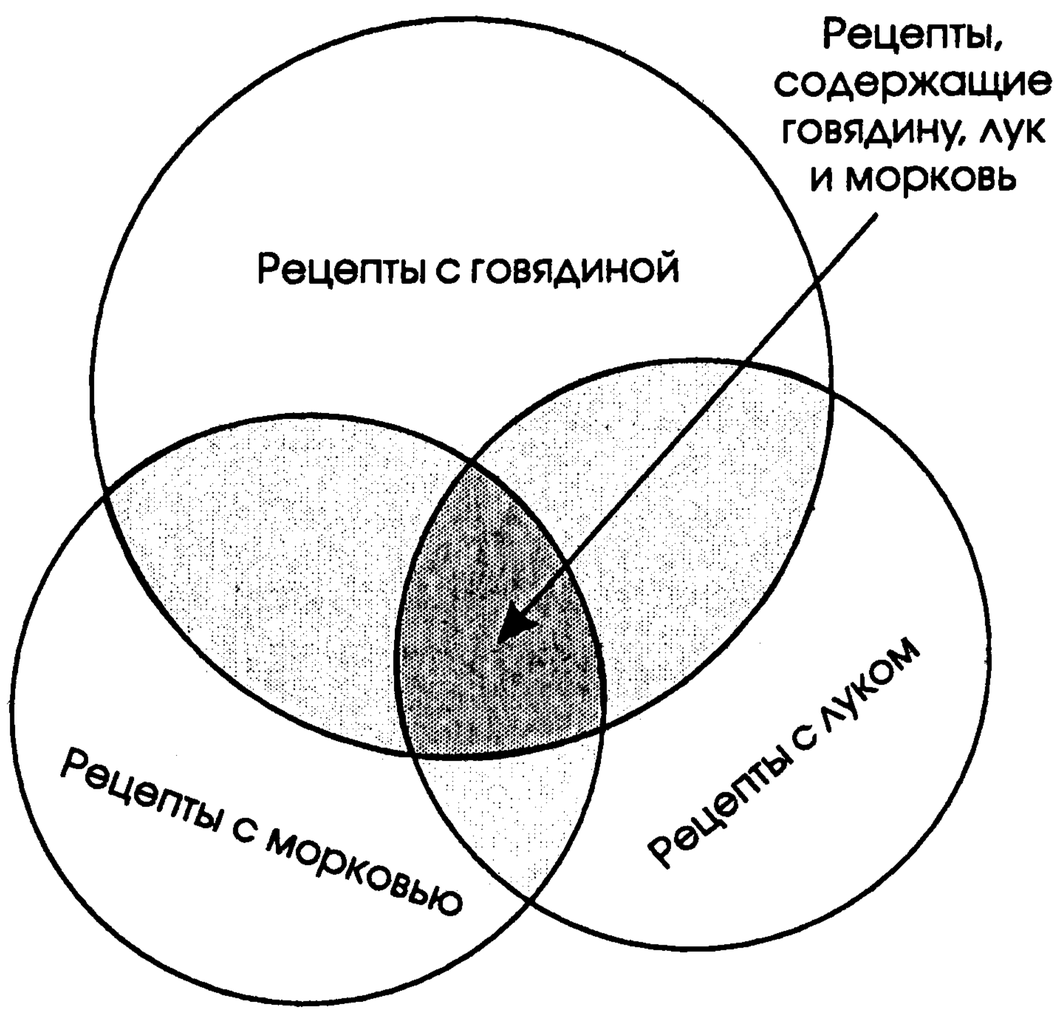


Рис. 7.2. *Рецепты с говядиной, луком и морковью*

А как насчет немного более сложных проблем? Пусть, скажем, вам захотелось добавить в нашу смесь еще и морковь. Диаграмма для множества, показывающая решение в графическом виде, представлена на рис. 7.2.

Уловили секрет? Основной момент состоит в том, что при решении проблем, включающих сложные критерии, диаграмма для множеств может стать неоценимым способом представления решения, выраженного пересечением наборов результатов SQL.

Задачи, которые можно решить с помощью INTERSECT

Ключевое слово INTERSECT может использоваться для поиска совпадений между двумя и более множествами информации. Приведем небольшой пример задач, которые можно решить, используя метод пересечения, для данных из учебных БД.

"Показать клиентов и сотрудников, имеющих одинаковые имена".

"Найти всех клиентов заказавших мотоцикл, которые также заказали шлем".

"Привести список эстрадных артистов, которые отыграли ангажементы для клиентов Бонниксен и Росалес".

"Показать студентов, имеющих средний балл 85 или выше по курсу "Искусство", и тех, кто также имеет средний балл 85 или выше по курсу "Вычислительная техника".

"Найти игроков в боулинг, которые имеют предварительный счет 155 или выше на Зандербирд Лэйнс и Болеро Лэйнс".

"Показать рецепты, содержащие говядину и чеснок".

Одним из ограничений использования чистого пересечения является то, что значения должны совпадать во всех столбцах каждого набора результатов. Это хорошо работает, когда нужно пересечение двух или более множеств из одной и той же таблицы, например клиенты, которые заказали мотоциклы, и клиенты, которые заказали шлемы. Это также хорошо работает, когда нас интересуют множества из таблиц, имеющих подобные столбцы — например имена клиентов и имена сотрудников. Однако во многих случаях желательно найти решения, которые требуют совпадения только нескольких значений столбцов из каждого набора. Для этого типа задач SQL предоставляет операцию JOIN — пересечение по значениям ключа. Вот пример задач, которые можно решать с помощью JOIN.

"Показать клиентов и сотрудников, проживающих в одном городе". (JOIN по названию города.)

"Привести список заказчиков и забронированных ими эстрадных артистов". (JOIN по номеру ангажемента.)

"Найти агентов и эстрадных артистов, имеющих одинаковый почтовый индекс". (JOIN по почтовому индексу.)

"Показать студентов и их преподавателей, которые имеют одинаковые имена". (JOIN по имени.)

"Найти игроков в боулинг, имеющих одинаковое количество очков". (JOIN по текущему среднему.)

"Вывести на экран все компоненты для рецептов, которые содержат морковь". (JOIN по идентификатору компонента.)

В следующей главе мы покажем решение этих задач (и многих других) с использованием JOIN, поскольку совсем немного коммерческих реализаций SQL поддерживают INTERSECT.

Разность

Чему равна разность 21 и 10? Если ответ 11, то вы на верном пути! В операции разности (иногда также называемой вычитанием, минусованием или исключением) берется одно множество значений и из него удаляется множество значений другого

множества. То, что остается, является множеством значений первого множества, которые *отсутствуют* во втором множестве. EXCEPT является ключевым словом, используемым в стандарте SQL.

Разность в теории множеств

Разность является другим очень мощным математическим инструментом. Ученого может интересовать поиск того, что отличает два множества химических или физических выборочных данных. Например, ученый-химик, занимающийся исследованиями в области фармацевтики, может располагать двумя соединениями, которые кажутся очень похожими, но одно обеспечивает определенное благоприятное воздействие, а второе — нет. Обнаружение отличий этих двух соединений может помочь раскрыть, почему одно работает, а другое — нет. Инженер может располагать двумя подобными схемами, одна из которых работает лучше, чем вторая. Поиск различия в этих двух схемах может оказаться решающим для исключения структурных дефектов в будущих сооружениях.

Рассмотрим, как работает разность, исследуя два множества чисел. Первое множество следующее:

1, 5, 8, 9, 32, 55, 78

Второе множество такое:

3, 7, 8, 22, 55, 71, 99

Разностью первого и второго множества чисел являются числа, которые имеются в первом множестве, но отсутствуют во втором. Ответ:

1, 5, 9, 32, 78

Эту операцию разности можно перевернуть. Тогда разностью второго и первого множества будет:

3, 7, 22, 71, 99

Элементами каждого множества не обязательно должны быть просто значения. Решая задачи с помощью SQL, скорее всего вы будете работать с множествами строк.

Когда элемент множества является чем-то более сложным, чем просто отдельное число или значение, каждый элемент множества имеет несколько атрибутов (битов информации, которые описывают свойства каждого из элементов). Например, ваш любимый рецепт тушеного мяса является сложным элементом множества всех рецептов, которые содержат много различных компонентов. Можно рассматривать каждый компонент как атрибут сложного элемента — рецепта тушеного мяса.

Чтобы найти разность двух множеств, состоящих из сложных элементов, необходимо найти элементы во втором множестве, которые совпадают по всем атрибутам, с элементами первого множества. Не забывайте, что все элементы в каждом

из сравниваемых множеств должны иметь одинаковое количество и тот же тип атрибутов. Удалите из первого множества все совпадающие элементы, найденные во втором множестве, и результат будет представлять собой их разность. Предположим, что имеется сложное множество, в котором каждая строка представляет собой элемент множества (рецепт тушеного мяса), а каждый столбец обозначает конкретный атрибут (компонент рецепта):

Картофель	Вода	Мясо молодого барашка	Горошек
Рис	Куриный бульон	Курица	Морковь
Паста	Вода	Тофу	Молодой горошек в стручках
Картофель	Бульон из говядины	Говядина	Капуста
Паста	Вода	Свинина	Лук

Второе множество может выглядеть так:

Картофель	Вода	Мясо молодого барашка	Лук
Рис	Куриный бульон	Индюшатина	Морковь
Паста	Отвар из овощей	Тофу	Молодой горошек в стручках
Картофель	Бульон из говядины	Говядина	Капуста
Бобы	Вода	Свинина	Лук

Разностью этих двух множеств являются объекты в первом множестве, которые отсутствуют во втором множестве:

Картофель	Вода	Мясо молодого барашка	Горошек
Рис	Куриный бульон	Курица	Морковь
Паста	Вода	Тофу	Молодой горошек в стручках
Паста	Вода	Свинина	Лук

Разность наборов результатов

Когда дело касается строк множества данных, извлеченных в SQL, атрибутами являются отдельные столбцы. Предположим, что имеется множество строк, возвращенных запросом (это рецепты из поваренной книги Джона):

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная баранина	Картофель	Вода	Баранина	Горошек
Тушеная курица	Рис	Куриный бульон	Курица	Морковь
Вегетарианские тушеные овощи	Паста	Вода	Тофу	Молодой горошек в стручках
Тушеное мясо по-ирландски	Картофель	Бульон из говядины	Говядина	Капуста
Тушеная свинина	Паста	Вода	Свинина	Лук

Набор результатов второго запроса может иметь следующий вид (это рецепты из поваренной книги Майкла):

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная баранина	Картофель	Вода	Баранина	Горошек
Тушеная индейка	Рис	Куриный бульон	Индейка	Морковь
Вегетарианские тушеные овощи	Паста	Овощной отвар	Тофу	Молодой горошек в стручках
Тушеное мясо по-ирландски	Картофель	Бульон из говядины	Говядина	Капуста
Тушеная свинина	Бобы	Вода	Свинина	Лук

Разность между рецептами Джона и Майкла представляет собой рецепты из поваренной книги Джона, которые *отсутствуют* в поваренной книге Майкла:

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная курица	Рис	Куриный бульон	Курица	Морковь
Вегетарианские тушеные овощи	Паста	Вода	Тофу	Молодой горошек в стручках
Тушеная свинина	Паста	Вода	Свинина	Лук

Можно посмотреть на задачу с другой стороны. Предположим, нужно найти рецепты из поваренной книги Майкла, которых *нет* в поваренной книге Джона. Вот ответ:

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная индейка	Рис	Куриный бульон	Индейка	Морковь
Вегетарианские тушеные овощи	Паста	Овощной отвар	Тофу	Молодой горошек в стручках
Тушеная свинина	Бобы	Вода	Свинина	Лук

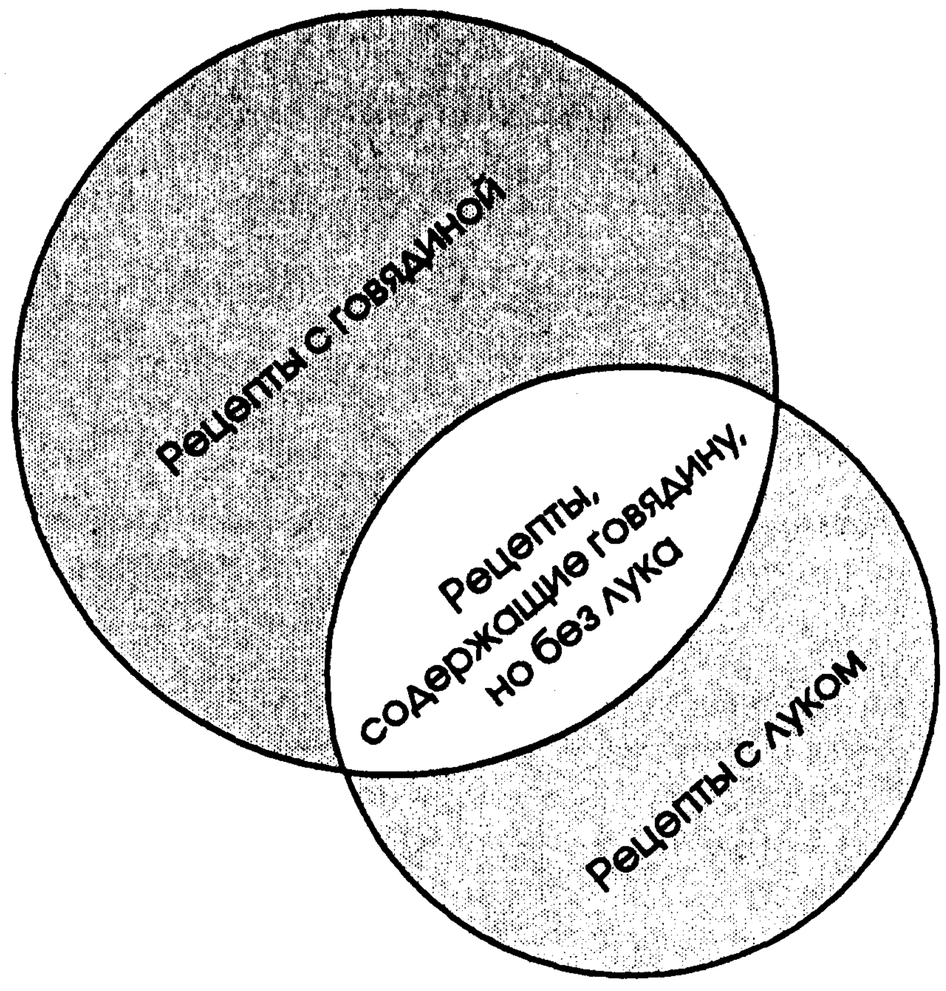


Рис. 7.3. *Рецепты с говядиной, но без лука*

Снова воспользуемся графическим представлением множеств, чтобы наглядно показать, как работает операция разности. Предположим, что имеется БД, содержащая все ваши любимые рецепты. Вам очень не нравится вкус лука с говядиной, поэтому нужно найти все рецепты с говядиной, но без лука. На рис. 7.3 представлена диаграмма множеств, которая помогает наглядно представить решение этой задачи.

Верхний круг представляет собой множество рецептов, в которых используется говядина. Нижний круг представляет множество рецептов, которые содержат лук. Там, где два круга перекрываются, находятся рецепты, включающие оба компонента. Затемненная часть верхнего круга, которая не является частью перекрывающейся области, представляет множество рецептов, содержащих говядину, но не содержащих лук. Подобным образом часть нижнего круга, которая не является частью перекрывающейся области, представляет собой множество рецептов, содержащих лук, но без говядины.

Вначале в SQL запрашивается извлечение всех рецептов, содержащих говядину. Затем извлекаются все рецепты, содержащие лук. Специальное ключевое слово SQL — EXCEPT — связывают два запроса для получения окончательного ответа.

Не попали ли вы снова в ловушку? Если таблица рецептов выглядит подобно нашим примерам, то вы, вероятно, думаете, что можно просто запросить:

> *"Show me the recipes that have beef as the meat ingredient and that do not have onions as the vegetable ingredient."*
> *("Показать рецепты, которые в качестве мясного компонента содержат говядину и не содержат в качестве овощного компонента лук".)*

Преобразование:
Select the recipe name from the recipes table where meat ingredient is beef and vegetable ingredient is not onions
(Выбрать наименование рецепта из таблицы "Рецепты", где мясным компонентом является говядина, а овощным компонентом не является лук)

Уточнение:
Select ~~the~~ recipe name from ~~the~~ recipes ~~table~~ where meat ingredients ~~is~~ = beef and vegetable ingredient ~~is not~~ <> onions
(Выбрать наименование рецепта из "Рецепты", где мясной компонент = говядина и овощной компонент <> лук)

```
SQL                SELECT RecipeName
                   FROM Recipes
                   WHERE MeatIngredient = 'Beef
                      AND VegetableIngredient <> 'Onions'
```

Здесь снова встает вопрос относительно рецептов, которые содержат другие компоненты, а не мясо и овощи. Ведь некоторые рецепты содержат множество компонентов, а другие — лишь несколько. Правильно спроектированная база данных рецептов включает отдельную таблицу Recipe_Ingredients (Компоненты_рецептов) с одной строкой для каждого компонента в каждом рецепте. Каждая строка компонента будет содержать только один компонент, поэтому не может быть строки с говядиной и луком одновременно. Вначале потребуется найти все строки с говядиной, затем все строки с луком и, наконец, найти их разность на RecipeID.

А как насчет немного более сложных проблем? Пусть, скажем, вам захотелось добавить в нашу смесь еще и морковь. Диаграмма для множества, показывающая решение, представлена на рис. 7.4.

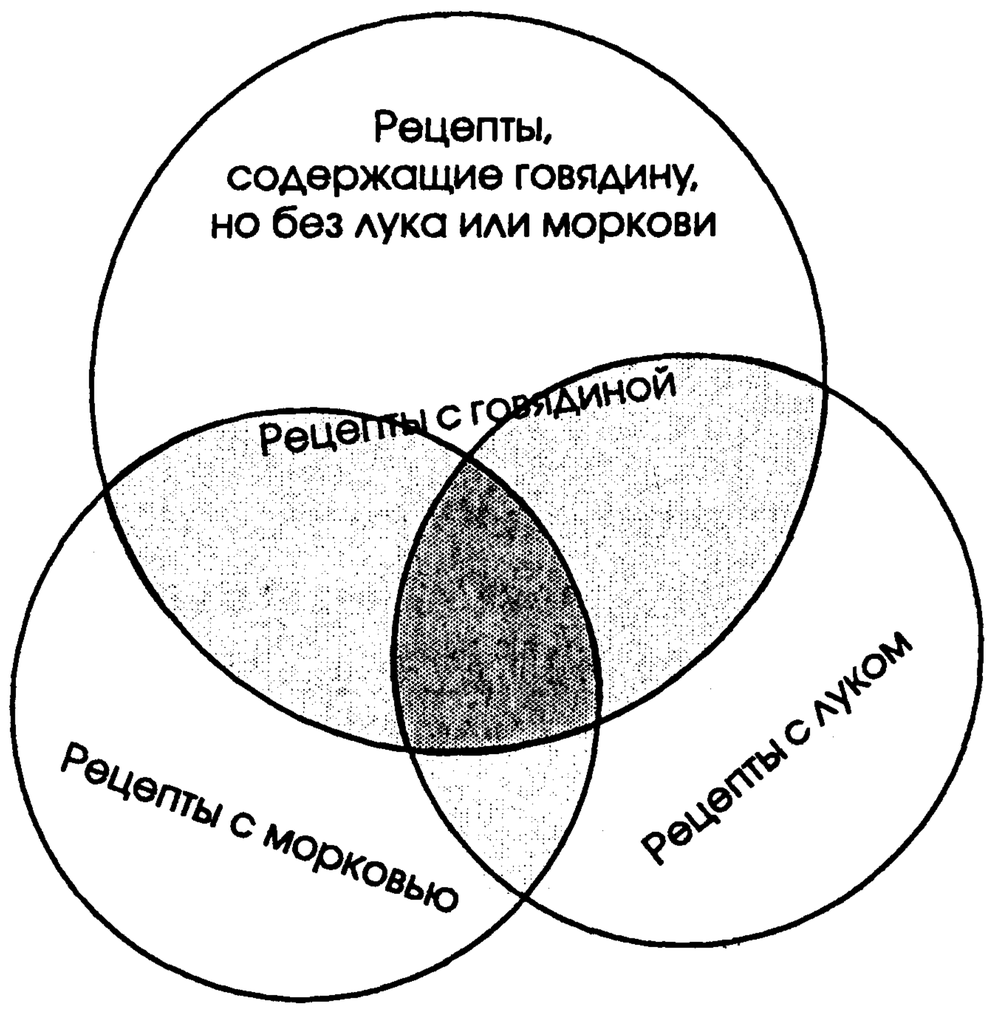


Рис. 7.4. *Рецепты с говядиной, но без лука или моркови*

Вначале нужно найти множество рецептов, содержащих говядину, затем получить разность либо с множеством рецептов, содержащих лук, либо с множеством, содержащим морковь. Возьмите этот результат и получите разность с оставшимся множеством (луком или морковью), чтобы оставить только те рецепты, которые содержат говядину, но не содержат морковь или лук (область штриховки светлого тона на верхнем круге).

Задачи, которые можно решить, используя разность

В отличие от пересечения (при котором осуществляется поиск общих элементов двух множеств) разность ищет элементы, которые имеются в одном множестве, но *отсутствуют* в другом. Приведем здесь небольшой пример задач, которые можно решить, используя метод разности, с данными из учебных баз данных:

"Показать клиентов, имена которых не совпадают с именами сотрудников".

"Найти всех клиентов заказавших мотоцикл, но не заказавших шлем".

"Привести список эстрадных артистов, которые отыграли ангажементы для клиентов Бонниксен, но не сыграли ни одного ангажемента для клиентов Росалес".

"Показать студентов, имеющих средний балл 85 или выше по курсу "Искусство", но не имеющих среднего балла 85 или выше по курсу "Вычислительная техника".

"Найти игроков в боулинг, которые имеют предварительный счет 155 или выше на Зандербирд Лэйнс, но не на Болеро Лэйнс".

"Показать рецепты, содержащие говядину, но без чеснока".

Одним из ограничений использования чистой разности является то, что значения должны совпадать во всех столбцах каждого набора результатов. Это хорошо работает, если нужна разность двух или более множеств из одной и той же таблицы — например клиенты, которые заказали мотоциклы, и клиенты, которые заказали шлемы. Это также хорошо работает, когда нас интересует разность множеств из таблиц, имеющих подобные столбцы, например имена клиентов и имена сотрудников.

Однако во многих случаях желательно найти решения, которые требуют совпадения только нескольких значений столбцов из каждого набора. Для этого типа задач SQL предоставляет операцию OUTER JOIN, представляющую собой пересечение множеств значений ключей, которое включает несовпадающие значения из одного или из обоих множеств. Вот пример задач, которые можно решать с помощью OUTER JOIN.

"Показать клиентов, которые не проживают в том же городе, что и любой из сотрудников". (OUTER JOIN по имени города.)

"Привести список заказчиков и эстрадных артистов, которых они не пригласили". (OUTER JOIN по номеру ангажемента.)

"Найти агентов, не имеющих одинакового почтового индекса с любым из эстрадных артистов". (OUTER JOIN по почтовому индексу.)

"Показать студентов, которые не имеют одинаковых имен с любым из преподавателей". (OUTER JOIN по имени.)

"Найти игроков в боулинг, имеющих среднее количество очков 150 или выше, которые никогда не имели в игре меньше 125 очков". (OUTER JOIN по идентификатору игрока из двух различных таблиц.)

"Вывести на экран все компоненты для рецептов, в которых нет моркови". (OUTER JOIN по идентификатору рецепта.)

В главе 9 будет показано решение этих задач (и многих других) с использованием OUTER JOIN, поскольку немногие коммерческие реализации SQL поддерживают EXEPT.

Объединение

До сих пор обсуждался поиск элементов, которые являются общими в двух множествах (пересечение) и элементов, которые различаются между собой (разность). Третий тип операций с множествами вызывает “сложение” двух множеств (объединение).

Объединение в теории множеств

Объединение позволяет объединить два множества с подобной информацией в одно множество. Ученого может интересовать объединение двух множеств химических или физических выборочных данных. Например, ученый-химик, занимающийся исследованиями в области фармацевтики, может располагать двумя различными множествами соединений, которые оказывают определенное благоприятное воздействие. Химик может объединить эти два множества для получения общего списка всех действующих соединений.

Рассмотрим процесс объединения, исследуя два множества чисел. Первое множество чисел следующее:

1, 5, 8, 9, 32, 55, 78

Второе множество чисел:

3, 7, 8, 22, 55, 71, 99

Объединение этих двух множеств чисел составляют числа из обоих множеств, объединенные в одно новое множество:

1, 5, 8, 9, 32, 55, 78, 3, 7, 22, 71, 99

Значения, общие для обоих множеств (8 и 55), появляются в ответе только один раз. Последовательность чисел в наборе результатов не обязательно располагается в некотором конкретном порядке. Если в системе базы данных выполняется UNION, то возвращенные значения, не обязательно будут расположены последовательно, если явно не включено условие ORDER BY. В SQL можно также запросить UNION ALL, если вы хотите видеть двойные элементы.

Элементами каждого множества не обязательно должны быть только одиночные значения. Скорее всего, при работе с SQL вы будете сталкиваться с множествами строк.

Чтобы можно было объединить два или более множеств сложных элементов, все элементы в каждом из объединяемых множеств должны иметь одинаковое количество и тип атрибутов. Предположим, имеется сложное множество, в котором каждая строка представляет элемент множества (рецепт тушеного мяса), а каждый столбец обозначает конкретный атрибут (компонент рецепта).

Картофель	Вода	Мясо молодого барашка	Горошек
Рис	Куриный бульон	Курица	Морковь
Паста	Вода	Тофу	Молодой горошек в стручках
Картофель	Бульон из говядины	Говядина	Капуста
Паста	Вода	Свинина	Лук

Второе множество выглядит так:

Картофель	Вода	Мясо молодого барашка	Лук
Рис	Куриный бульон	Индюшатина	Морковь
Паста	Отвар из овощей	Тофу	Молодой горошек в стручках
Картофель	Бульон из говядины	Говядина	Капуста
Бобы	Вода	Свинина	Лук

Объединение этих двух множеств представляет собой множество объектов из обоих множеств; повторяющиеся строки исключаются:

Картофель	Вода	Мясо молодого барашка	Горошек
Рис	Куриный бульон	Курица	Морковь
Паста	Вода	Тофу	Молодой горошек в стручках
Картофель	Бульон из говядины	Говядина	Капуста
Паста	Вода	Свинина	Лук
Картофель	Вода	Мясо молодого барашка	Лук
Рис	Куриный бульон	Индюшатина	Морковь
Паста	Отвар из овощей	Тофу	Молодой горошек в стручках
Бобы	Вода	Свинина	Лук

Объединение наборов результатов с использованием UNION

Между множествами сложных объектов и строками наборов результатов совсем небольшое различие. При работе со строками множества данных, извлеченных

в SQL, атрибутами являются отдельные столбцы. Предположим, что имеется множество строк, возвращенных запросом (это рецепты из поваренной книги Джона):

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная баранина	Картофель	Вода	Баранина	Горошек
Тушеная курица	Рис	Куриный бульон	Курица	Морковь
Вегетарианские тушеные овощи	Паста	Вода	Тофу	Молодой горошек в стручках
Тушеное мясо по-ирландски	Картофель	Бульон из говядины	Говядина	Капуста
Тушеная свинина	Паста	Вода	Свинина	Лук

Набор результатов второго запроса может иметь следующий вид (это рецепты из поваренной книги Майкла):

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная баранина	Картофель	Вода	Баранина	Горошек
Тушеная индейка	Рис	Куриный бульон	Индейка	Морковь
Вегетарианские тушеные овощи	Паста	Овощной отвар	Тофу	Молодой горошек в стручках
Тушеное мясо по-ирландски	Картофель	Бульон из говядины	Говядина	Капуста
Тушеная свинина	Бобы	Вода	Свинина	Лук

Объединением этих двух множеств являются все строки обоих множеств. Может быть, Джон и Майкл решат написать вместе еще поваренную книгу!

Рецепт	Крахмал	Бульон	Мясо	Овощи
Тушеная баранина	Картофель	Вода	Баранина	Горошек
Тушеная курица	Рис	Куриный бульон	Курица	Морковь
Вегетарианские тушеные овощи	Паста	Вода	Тофу	Молодой горошек в стручках
Тушеное мясо по-ирландски	Картофель	Бульон из говядины	Говядина	Капуста
Тушеная свинина	Паста	Вода	Свинина	Лук
Тушеная индейка	Рис	Куриный бульон	Индейка	Морковь
Вегетарианские тушеные овощи	Паста	Овощной отвар	Тофу	Молодой горошек в стручках
Тушеная свинина	Бобы	Вода	Свинина	Лук

Рис. 7.5. *Поиск рецептов, содержащих либо говядину, либо лук*

Предположим, что имеется удачно выполненная база данных, состоящая из всех ваших любимых рецептов. Вам чрезвычайно нравится рецепты как с говядиной, так и с луком, и поэтому хотелось бы найти все рецепты, содержащие любой из этих компонентов. На рис. 7.5 представлена диаграмма для множества, которая помогает наглядно представить решение этой задачи.

Верхний круг представляет собой множество рецептов, в которых используется говядина. Нижний круг представляет множество рецептов, содержащих лук. Объединение этих двух кругов дает все рецепты, содержащие любой из компонентов, с исключением повторяющихся строк там, где два круга перекрываются. В SQL вначале запрашивается извлечение всех рецептов, содержащих говядину. Во втором запросе запрашивается извлечение всех рецептов, в которых есть лук. Ключевое слово SQL — UNION — объединяет два запроса для получения окончательного ответа.

Еще раз повторим, что проектирование структуры базы данных рецептов с единой таблицей не является хорошей идей. Правильно спроектированная база данных рецептов должна иметь отдельную таблицу Recipe_Ingredients (Компоненты_рецепта) с отдельной строкой для компонента каждого рецепта. В каждой строке компонента будет храниться только один компонент, поэтому в одной строке не могут одновременно находиться как говядина, так и лук. Вначале потребуется найти все рецепты, содержащие строку “говядина”, а затем найти все рецепты, в которых имеется строка “лук”, и после этого объединить их.

Задачи, которые можно решить с помощью UNION

UNION позволяет объединить строки из двух подобных множеств с дополнительным преимуществом в виде отсутствия двойных строк. Вот пример задач, которые можно решить с помощью метода объединения, для данных из учебных баз данных:

“Показать имена и адреса всех клиентов и сотрудников”.

“Привести список всех клиентов, заказавших велосипед, вместе со всеми клиентами, заказавшими шлем”.

“Привести список эстрадных артистов, которые отыграли ангажементы для клиентов Бонниксен, объединив их с эстрадными артистами, которые отыграли ангажементы для клиентов Росалес”.

"Показать студентов, имеющих средний балл 85 или выше по курсу "Искусство", вместе с теми студентами, которые также имеют средний балл 85 или выше по курсу "Вычислительная техника".

"Найти игроков в боулинг, которые имеют предварительный счет 155 или выше на Зандербирд Лэйнс, объединив их с игроками в боулинг на Болеро Лэйнс, у которых предварительный счет 140 или выше".

"Показать рецепты, содержащие говядину, вместе со списком рецептов, включающих чеснок".

Как и с другими "чистыми" операциями над множествами, одним из ограничений является то, что значения должны совпадать по всем столбцам в каждом наборе результата. Это хорошо работает при объединении двух или более множеств одной и той же таблицы — например клиенты, заказавшие велосипеды, и клиенты, заказавшие шлемы. Это также хорошо работает при выполнении UNION над множествами из таблиц, имеющих подобные столбцы,— например имена и адреса клиентов, имена и адреса сотрудников. Более подробно использование UNION рассматривается в главе 10.

Во многих случаях при объединении строк из одной таблицы иным образом также полезно использование DISTINCT (для исключения дублирования строк) со сложным критерием в объединенной таблице. О таком способе решения задач с использованием операторов JOIN мы расскажем в главе 8.

Операции с множествами в SQL

Теперь кратко рассмотрим реализацию множеств в SQL.

Сравнение "классических" операций над множествами с операциями SQL

Пока еще немногие коммерческие системы баз данных поддерживают непосредственно операции пересечения множеств или разность множеств. Однако текущий стандарт SQL четко определяет, как должны быть реализованы эти операции, полагая, что они являются достаточно важными, чтобы, по крайней мере, служить основанием для обзора синтаксиса.

Поиск общих значений — INTERSECT

Предположим, вы пытаетесь решить следующую простую на вид задачу:

"Show me the orders that contain both a bike and a helmet".
("Показать заказы, содержащие как велосипед, так и шлем".)

Преобразование: Select the distinct order numbers from the order details table where the product number is in the list of bike and helmet product numbers
(Выбрать несовпадающие номера заказов из таблицы "Детали заказа", где номер товара присутствует в списке номеров товаров для велосипеда и шлема)

Уточнение: Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike and helmet product numbers
(Выбрать несовпадающие номера заказов из "Детали заказа", где номер товара в номерах товаров для велосипеда и шлема)

SQL

```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 10, 11, 25, 26)
```

Внимание! Читатели, хорошо знакомые с SQL, могут спросить, почему мы не объединили (JOIN) Order_Details (Детали_заказа) с Products (Товары) и не выполнили поиск велосипеда и шлема по наименованиям товаров. Дело в том, что мы еще не представили концепцию JOIN, поэтому строим показанный выше пример на единой таблице, применяя IN и список известных номеров товаров для велосипеда и шлема.

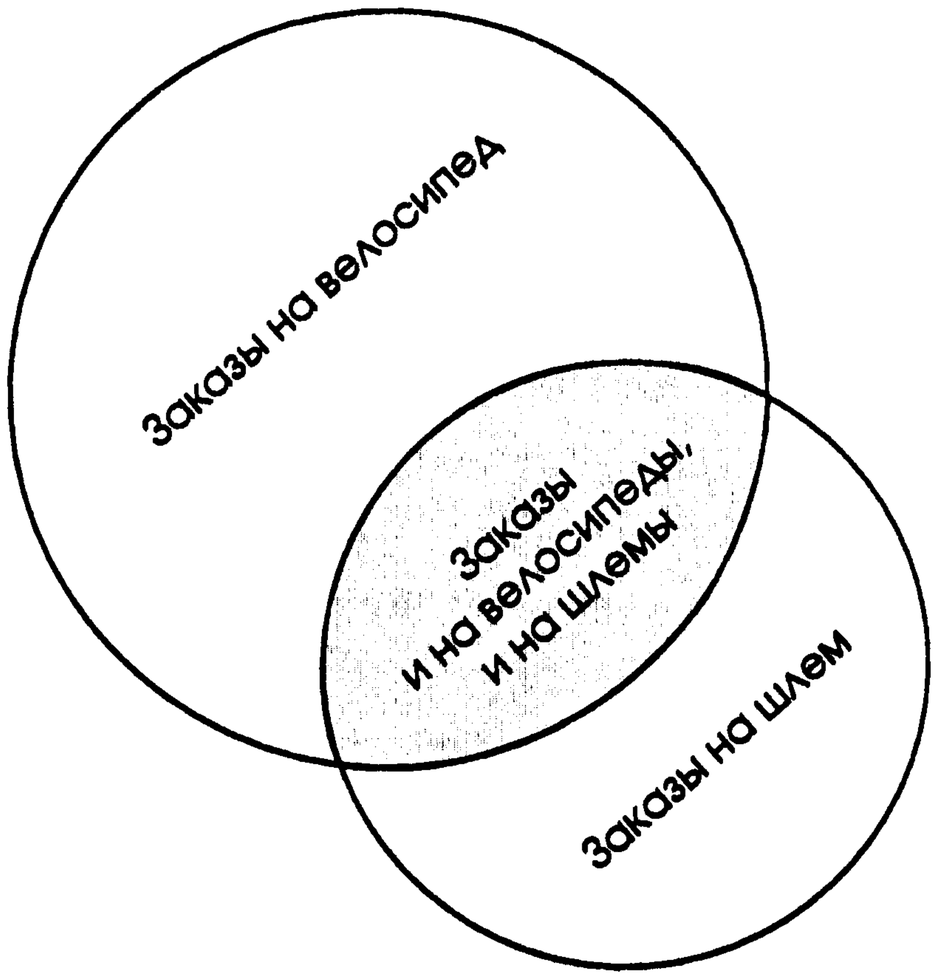


Рис. 7.6. *Единственная возможная связь между двумя множествами заказов*

Вначале это кажется фокусом, но ответ включает заказы, которые содержат *либо* велосипед, *либо* шлем, а требовалось найти такие, которые содержат *оба* товара. Если представить графически "Заказы на велосипеды" *и* "Заказы на шлемы" как два отдельных множества, будет легче понять проблему. На рис. 7.6 показано возможное отношение между двумя множествами заказов с использованием диаграммы для множеств.

Фактически отсутствует способ предварительного предсказания, какими могут быть отношения между двумя множествами данных. На рис. 7.6 некоторые заказы содержат в списке заказанных товаров велосипед, но не содержат шлем. Другие содержат шлем, но нет велосипеда. В области перекрытия,

или пересечения, двух множеств находятся заказы, которые содержат как велосипед, так и шлем. На рис. 7.7 представлен другой случай, когда *все* заказы, содержащие шлем, также содержат велосипед, но некоторые заказы, содержащие велосипед, не содержат шлема.

Рис. 7.7. *Все заказы на шлем содержат также заказ на велосипеды*

Наличие в запросе слова both (“оба”) предполагает, что вы, вероятно, разделите решение на отдельные множества данных, а затем каким-либо образом свяжете эти два множества. (Ваш запрос также требуется разделить на две части.)

“Show me the orders that contain a bike”.
(“Показать заказы, содержащие велосипед”.)

Преобразование: Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers
(Выбрать неповторяющиеся номера заказов из таблицы “Детали заказов”, где номер товара находится в списке номеров товаров “велосипеды”)

Уточнение: Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers
(Выбрать неповторяющиеся номера заказов из “Детали заказов”, где номер товара в номерах товаров “велосипеды”)

SQL

```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
```

“Show me the orders that contain a helmet”.
(“Показать заказы, содержащие шлем”.)

Преобразование: Select the distinct order numbers from the order details table where the product number is in the list of helmet product numbers
(Выбрать неповторяющиеся номера заказов из таблицы “Детали заказов”, где номер товара находится в списке номеров товаров “шлем”)

Уточнение: Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where the product number ~~is~~ in ~~the list of~~ helmet product numbers
(Выбрать неповторяющиеся номера заказов из "Детали заказов", где номер товара в номерах товаров "шлем")

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

Мы получим окончательное решение, если используем *пересечение* этих двух множеств. На рис. 7.8 представлена синтаксическая диаграмма SQL, которая решает эту задачу. (Обратите внимание: INTERSECT можно использовать несколько раз для объединения нескольких операторов SELECT.)

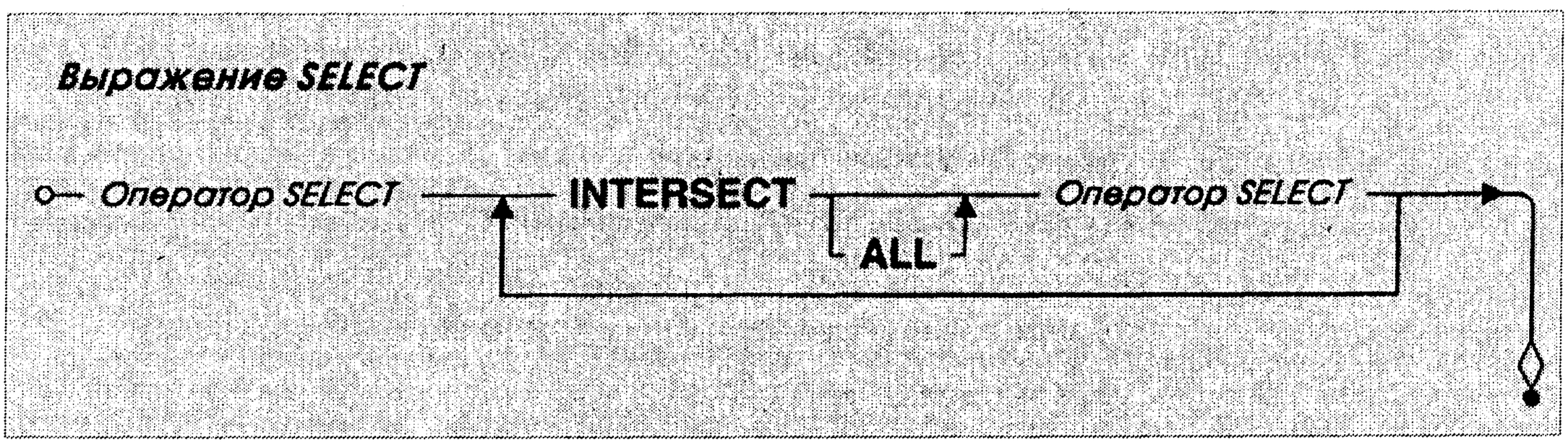


Рис. 7.8. *Связывание двух операторов SELECT в INTERSECT*

Можно взять две части нашего запроса и связать их вместе в операторе INTERSECT, чтобы получить правильный ответ:

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
INTERSECT
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

Жаль, что мало коммерческих реализаций SQL пока еще поддерживают оператор INTERSECT. Однако не все потеряно! Вспомните, что первичный ключ таблицы уникальным образом идентифицирует каждую строку (не требуется совпадения по всем полям в строке — только по первичному ключу — для поиска уникальных строк, которые "пересекаются".) В главе 8 мы покажем альтернативный метод (JOIN), который может решать этот тип проблем другим способом. Большинство коммерческих реализаций SQL *поддерживают* JOIN.

Поиск пропущенных значений — EXCEPT (Разность)

Вернемся снова к задаче с велосипедами и шлемами. Предположим, что вы пытаетесь решить этот, на первый взгляд простой, запрос следующим образом:

"Show me the orders that contain a bike but not a helmet".
("Показать заказы, содержащие велосипед, но не содержащие шлем".)

Преобразование: Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers and product number is not in the list of helmet product numbers
(Выбрать неповторяющиеся номера заказов из таблицы "Детали заказов", где номер товара находится в списке номеров товаров "велосипед" и не находится в списке номеров товаров "шлем".)

Уточнение: Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers and product number ~~is~~ not in ~~the list of~~ helmet product numbers
(Выбрать неповторяющиеся номера заказов из "Детали заказов", где номер товара в номерах товаров "велосипед и не в номерах товаров "шлем")

SQL

```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
    AND ProductNumber NOT IN (10, 25, 26)
```

К сожалению, ответ покажет только заказы, содержащие велосипед! Проблема состоит в том, что первое IN условия находит строки, содержащие велосипед, но второе условие IN просто исключает строки со шлемом. Если графически представить "Заказы на велосипеды" и "Заказы на шлемы" как два отдельных множества, это будет легче понять. На рис. 7.9 показана возможная связь между этими двумя множествами заказов.

Если в запросе встречается "except" (за исключением) или "but not" (кроме), это предполагает, что решение требуется разделить на два отдельных множества данных, а затем каким-либо образом связать их. (Ваш запрос также необходимо разделить на две части.)

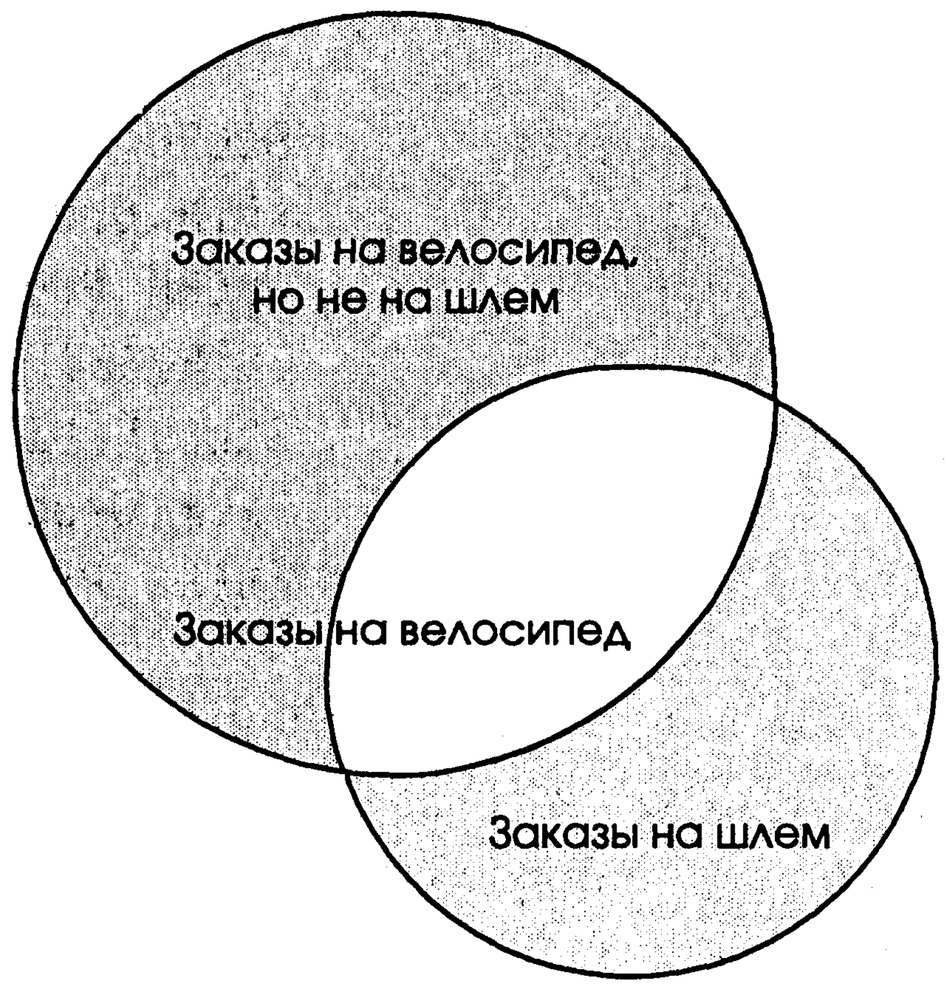


Рис. 7.9. *Заказы на велосипед, не содержащие заказа на шлем*

"Show me the orders that contain a bike".
("Показать заказы, содержащие велосипед".)

Преобразование: Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers
(Выбрать неповторяющиеся номера заказов из таблицы "Детали заказов", где номер товара находится в списке номеров товаров "велосипед")

Уточнение: Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers
(Выбрать неповторяющиеся номера заказов из "Детали заказов", где номер товара среди номеров товаров "велосипед")

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
```

"Show me the orders that contain a helmet".
("Показать заказы, содержащие шлем".)

Преобразование: Select the distinct order numbers from the order details table where the product number is in the list of helmet product numbers
(Выбрать неповторяющиеся номера заказов из таблицы "Детали заказов", где номер товара находится в списке номеров товаров "шлем")

Уточнение: Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ helmet product numbers
(Выбрать неповторяющиеся номера заказов из "Детали заказов", где номер товара среди номеров товаров "шлем")

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

Теперь можно получить окончательное решение, используя *разность* двух множеств. В SQL для обозначения операции разности используется ключевое слово EXCEPT. На рис. 7.10 представлена синтаксическая диаграмма SQL, которая решает эту проблему.

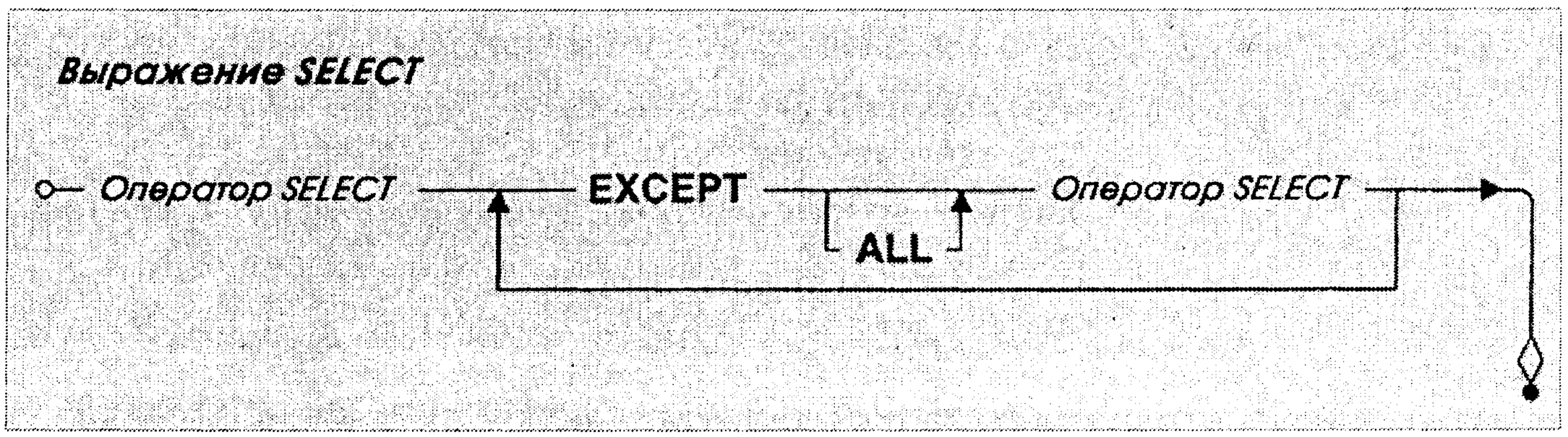


Рис. 7.10. *Связывание двух операторов SELECT в EXCEPT*

Теперь можно объединить две части запроса в операторе EXCEPT, чтобы получить правильный ответ:

```
SQL                 SELECT DISTINCT OrderNumber FROM Order_Details
                    WHERE ProductNumber IN (1, 2. 6, 11)
                    EXCEPT
                    SELECT DISTINCT OrderNumber FROM Order_Details
                    WHERE ProductNumber IN (10, 25, 26)
```

Вспомните, что последовательность множеств имеет значение. В нашем случае мы обращаемся с запросом о велосипедах "за исключением" шлемов. Если нужно узнать иное — заказы на шлемы, которые не содержат заказов на велосипеды,— можно выполнить преобразование следующим образом:

```
SQL                 SELECT DISTINCT OrderNumber FROM Order_Details
                    WHERE ProductNumber IN (10, 25, 26)
                    EXCEPT
                    SELECT DISTINCT OrderNumber FROM Order_Details
                    WHERE ProductNumber IN (1, 2, 6. 11)
```

Пока еще мало коммерческих реализаций SQL поддерживают оператор EXCEPT. Крепче держите свой шлем! Однако вспомните, что первичный ключ таблицы уникальным образом идентифицирует каждую строку (не требуется совпадения по всем полям в строке — только по первичному ключу — для поиска уникальных строк, которые "различаются"). В главе 9 мы покажем альтернативный метод OUTER JOIN, который может решать этот тип проблем другим способом. Большинство коммерческих реализаций SQL *поддерживают* OUTER JOIN.

Объединение множеств — UNION

Еще один пример с велосипедами и шлемами. Допустим, нужно решить следующий запрос, который при поверхностном взгляде кажется достаточно простым:

> *"Show me the orders that contain either a bike or a helmet".*
> *("Показать заказы, содержащие либо велосипед, либо шлем".)*

Преобразование: Select the distinct order numbers from the order details table where the product number is in the list of bike and helmet product numbers
(Выбрать неповторяющиеся номера заказов из таблицы "Детали заказов", где номер товара находится в списке номеров товаров "велосипед" и "шлем")

Уточнение: Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike and helmet product numbers
(Выбрать неповторяющиеся номера заказов из "Детали заказов", где номер товара в номерах товаров "велосипед и "шлем")

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 10, 11, 25, 26)
```

Фактически это работает просто прекрасно! Тогда зачем для решения такой задачи использовать UNION? Правда в том, что можно и не использовать. Однако, если задачу немного усложнить:

"List the customers who ordered a bicycle together with the vendors who provide bicycles".
("Предоставить список клиентов, заказавших велосипед, вместе с поставщиками, поставляющими велосипеды".)

— UNION будет полезна.

Рис. 7.11. *Заказы на велосипеды или шлемы*

К сожалению, ответ на этот вопрос включает создание пары запросов с использованием операций JOIN, а затем использование UNION для получения окончательного результата. Поскольку мы еще не обсуждали выполнение JOIN, сохраним решение этой задачи до главы 10. Это позволяет заглянуть немного вперед, не так ли?

Вернемся обратно к задаче с "велосипедами или шлемами" и решим ее с помощью UNION. Если графически представить "Заказы на велосипеды" и "Заказы на шлемы" как два отдельных множества, задачу будет легче понять. На рис. 7.11 показано возможное отношение между этими двумя множествами.

Наличие слов “either” (любой), “or” (или) или “together” (вместе) в запросе наводит на мысль, что потребуется разделить решение на два отдельных множества данных, а затем связать их в UNION. Этот конкретный запрос можно разделить на две части.

“Show me the orders that contain a bike”.
(“Показать заказы, содержащие велосипед”.)

Преобразование: Select the distinct order numbers from the order details table where the product number is in the list of bike product numbers
(Выбрать неповторяющиеся номера заказов из таблицы “Детали заказов”, где номер товара находится в списке номеров товаров “велосипед”)

Уточнение: Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ bike product numbers
(Выбрать неповторяющиеся номера заказов из “Детали заказов”, где номер товара среди номеров товаров “велосипед”)

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (1, 2, 6, 11)
```

“Show me the orders that contain a helmet”.
(“Показать заказы, содержащие шлем”.)

Преобразование: Select the distinct order numbers from the order details table where the product number is in the list of helmet product numbers
(Выбрать неповторяющиеся номера заказов из таблицы “Детали заказов”, где номер товара находится в списке номеров товаров “шлем”)

Уточнение: Select ~~the~~ distinct order numbers from ~~the~~ order details ~~table~~ where ~~the~~ product number ~~is~~ in ~~the list of~~ helmet product numbers
(Выбрать неповторяющиеся номера заказов из “Детали заказов”, где номер товара среди номеров товаров “шлем”)

SQL
```
SELECT DISTINCT OrderNumber
FROM Order_Details
WHERE ProductNumber IN (10, 25, 26)
```

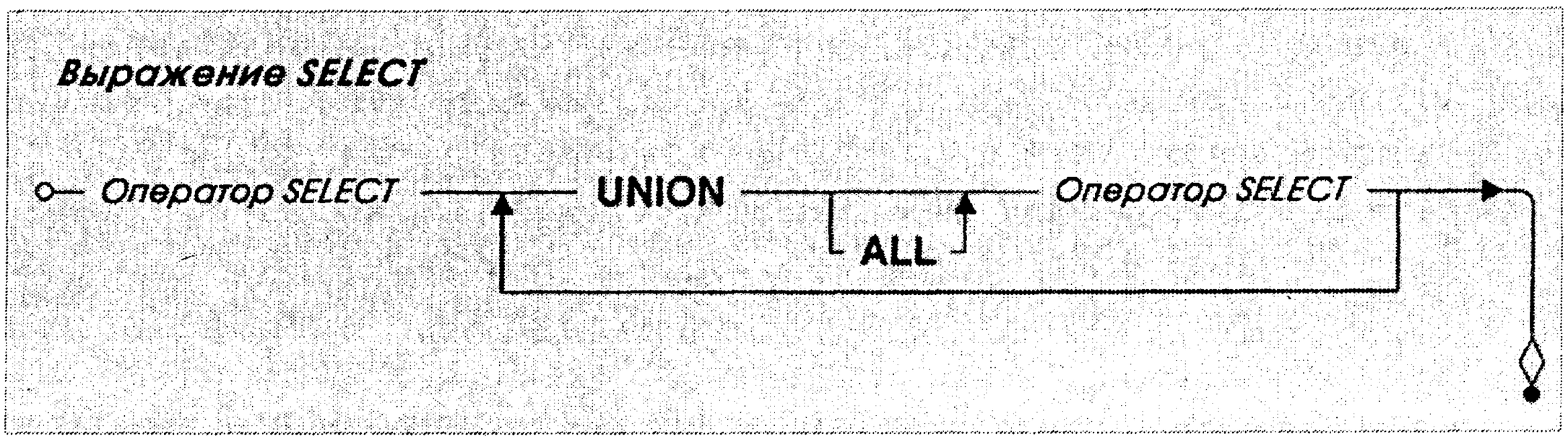


Рис. 7.12. *Связывание двух операторов SELECT в UNION*

Теперь можно получить окончательное решение, используя *объединение* двух множеств. На рис. 7.12 представлена синтаксическая диаграмма SQL, которая решает эту задачу.

Возьмем две части запроса и объединим их в операторе UNION, чтобы получить правильный ответ:

```
SQL                SELECT DISTINCT OrderNumber
                   FROM Order_Details
                   WHERE ProductNumber IN (1, 2. 6, 11)
                   UNION
                   SELECT DISTINCT OrderNumber
                   FROM Order_Details
                   WHERE ProductNumber IN (10, 25, 26)
```

Хорошо то, что большинство коммерческих реализаций SQL поддерживают оператор UNION. Как очевидно из приведенных выше примеров, UNION достигает своих целей сложным способом, когда желательно получить результат типа “включая/или” из одной таблицы. Для составления списка из нескольких имеющих одинаковую структуру, но разных таблиц, наиболее полезной операцией является UNION (подробнее см. в главе 10).

Итоги

В данной главе мы обсудили концепцию множества и каждую из основных операций над множествами, реализованную в SQL,— пересечение, разность и объединение. Для графического представления решаемой задачи можно использовать диаграммы множеств. Здесь были представлены основные синтаксические конструкции и ключевые слова SQL (IN, INTERSECT, EXCEPT и UNION) для всех трех операций.

Если вы задумались о том, зачем вам показали три вида операций над множествами, когда два из них, вероятно, невозможно использовать, то вспомните заголовок этой главы: “Мышление множествами”. Если вы собираетесь достичь полного успеха в решении сложных проблем, вам потребуется разделить свою задачу на наборы результатов с информацией, которые затем опять связываются вместе.

Поэтому, если задача содержит слова "должно быть это *и* должно быть то", значит, необходимо получить решение для "это", а затем — для "то" и после этого связать все вместе для получения окончательного решения. Стандарт SQL определяет удобную операцию INTERSECT, но JOIN может работать также хорошо (см. главу 8).

Более того, если задача включает условие "*должно быть это, но не должно быть* то", необходимо вначале решить "это", а затем "то" и после вычесть "то" из "это", чтобы получить ответ. Мы показали вам операцию EXCEPT из стандарта SQL, но, используя OUTER JOIN, также может выполнить этот трюк (см. главу 9).

В конце главы мы показали, как "складывать" множества информации с помощью UNION.

8

Внутренние соединения

"Не сдерживайте свое вдохновение и воображение,
не становитесь рабом натурщика".

— Винсент Ван Гог

Вопросы, рассматриваемые в данной главе:

- Что такое JOIN
- INNER JOIN
- Использование операций INNER JOIN
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

До этого момента наше внимание в основном было сосредоточено на решении задач с использованием взятых в отдельности таблиц. Теперь вы знаете, как получать простые ответы из одной таблицы, как получить немного более сложные ответы, используя выражения или сортировку набора результатов. Образно говоря, вам теперь известно, как безукоризненно нарисовать глаза, подбородок, рот или нос. В данной главе показано, как связать все это вместе, как "соединить" несколько частей лица, чтобы получить портрет.

Что такое JOIN

Мы уже подчеркивали важность разделения данных в таблицах по отдельным предметам. Однако большинство задач, которые необходимо решать в реальной жизни, требуют связывания в одно целое данных из нескольких таблиц — Customers (Клиенты) и Orders (Заказы), Customers и Entertainers (Эстрадные артисты), Bowlers (Игроки в боулинг) и Scores (Очки), Students (Студенты) и посещаемые ими Classes (Курсы лекций) или Recipes (Рецепты) и необходимые Ingredients (Компоненты). Для решения таких более сложных задач необходимо связать их вместе, или "соединить" несколько таблиц. Для этого используется ключевое слово *JOIN*.

В предыдущей главе показано, насколько полезно получить пересечение двух множеств данных для решения задач. Однако для получения ответа с использованием

INTERSECT требуется совпадение всех столбцов в обоих наборах результатов. JOIN также является операцией пересечения, но она отличается от остальных, потому что при ее выполнении от системы базы данных требуется присоединения только указанных столбцов. Таким образом, JOIN позволяет получить пересечение двух очень разнородных таблиц на основе совпадения значений столбцов. Например, можно воспользоваться JOIN для связывания Customers с их Orders путем сопоставления CustomerID из таблиц в Customers с CustomerID в таблице Orders.

JOIN рассматривается как часть условия FROM в операторе SQL. JOIN определяет "логическую таблицу", которая является результатом связывания двух таблиц или наборов результатов. Помещение JOIN в условие FROM определяет порядок связывания таблиц, из которых запрос извлекает окончательный набор результатов. Другими словами, JOIN заменяет имя отдельной таблицы. Можно также определить несколько операций JOIN для создания сложного набора результатов на основе более чем двух таблиц.

INNER JOIN

Стандарт SQL определяет несколько способов выполнения операций JOIN, наиболее обычным из которых является *INNER JOIN*. Предположим, вы связываете студентов и курсы лекций, на которые они записались. Может случиться так, что некоторые студенты, которые уже были приняты, еще не записались на какие-либо лекции. А может быть и так, что на некоторые курсы лекций, которые имеются в расписании, еще не записался ни один студент.

Операция INNER JOIN между таблицей Students и таблицей Classes возвращает строки Student, связанные с соответствующими строками Classes (через таблицу Student_Schedules) — но она не возвращает ни студентов, которые еще не зарегистрировались на какой-либо курс лекций, ни курсы лекций, на которые не записался ни один студент. INNER JOIN возвращает только те строки, где связывающие значения совпадают в обеих таблицах или в обоих наборах результатов.

Что "разрешено" в JOIN

Чаще всего связью, которую использует JOIN, является первичный ключ из одной таблицы и связанный внешний ключ из второй таблицы. Внешний ключ должен иметь тот же тип данных, что и связанный первичный ключ. Однако в JOIN также разрешается соединить две таблицы или два набора результатов по любым столбцам, которые имеют, как это называется в стандарте SQL, типы данных, "пригодные для соединения".

В общем случае можно соединить один символьный столбец с другим символьным столбцом или выражением, столбец любого числового типа (например, целый) с любым столбцом другого числового типа (возможно, со значениями типа "с плавающей точкой") и любой столбец типа "дата" с другим столбцом типа "дата". Это позволит, например, соединять в JOIN строки из таблицы Customers со строками из таблицы Employees по столбцам City (Город) или Zip Code (Почтовый индекс) —

возможно, чтобы узнать, кто из клиентов и сотрудников живут в одном и том же городе или в регионе с одинаковым почтовым индексом.

Внимание! *Можно* применить JOIN к любым столбцам, пригодным для соединения, в двух таблицах, но это не означает, что это обязательно *следует* делать. Связываемые столбцы должны иметь одинаковые значения данных, чтобы операция JOIN имела смысл. Например, не стоит выполнять ее для Customer Name (Имя_клиента) с Employee Address (Адрес_сотрудника), хотя оба столбца имеют символьный тип данных. В наборе результата не будет получено ни одного столбца, если только кто-нибудь не занес по ошибке имя в столбец Employee Address. Также не имеет смысла выполнять JOIN Student ID (Идентификатор студента) с Class ID (Идентификатором курса лекций), хотя они оба являются числами. Возможно, и будут получены какие-либо строки в наборе результатов, но они не будут иметь смысла.

Даже когда есть резон соединить в JOIN связывающие столбцы, это может закончиться построением запроса, выполнение которого потребует много времени. Например, запросив операцию JOIN столбцов, для которых администратор БД не определил индекс, СУБД должна будет проделать большой объем дополнительной работы. Запрашивая JOIN для выражений — например конкатенацию имени и фамилии из двух таблиц,— СУБД должна будет не только сформировать столбец результата из выражения для всех строк, но также и выполнить несколько просмотров всех данных в обеих таблицах, чтобы вернуть правильный результат.

Синтаксис

Теперь достанем наши палитры, смешаем краски и начнем изучать синтаксис INNER JOIN.

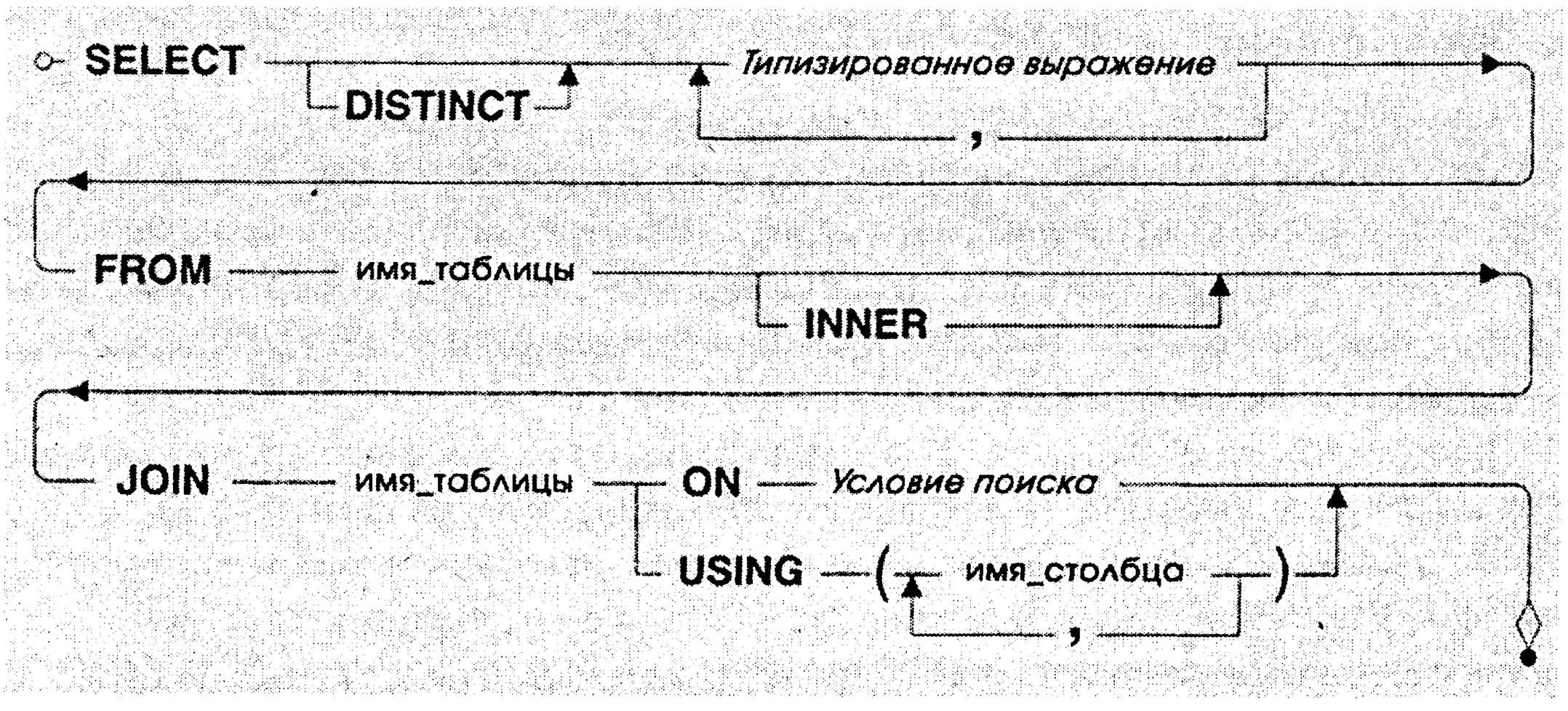


Рис. 8.1. *Диаграмма запроса с использованием INNER JOIN двух таблиц*

Использование таблиц

Начнем с простого — операции INNER JOIN двух таблиц. На рис. 8.1 представлена синтаксическая конструкция для создания запроса.

Как можно видеть, условие FROM теперь немного более сложное (условия WHERE и ORDER BY пока исключены для простоты). Вместо одного имени таблицы определяются имена двух таблиц, которые связываются ключевым словом JOIN. Необязательное ключевое слово INNER определяет тип соединения. Если тип нужного соединения не указан явно, то по умолчанию используется тип INNER. Рекомендуем всегда явно указывать тип нужного соединения, чтобы был понятен характер запроса.

Внимание! Воспользовавшиеся полной диаграммой синтаксической структуры из приложения А обнаружат, что операция `table_name JOIN table_name` описана как часть определенного термина "*соединенные таблицы*". *Ссылка на таблицы* включает *соединенные таблицы*, а условие FROM оператора SELECT использует *ссылку на таблицу*. Эти сложные определения были "свернуты" в одну диаграмму, чтобы облегчить изучение простого соединения двух таблиц. Такой же метод упрощения используется в синтаксических диаграммах остальной части данной главы.

Необходимой частью INNER JOIN является условие ON или USING, расположенное после имени второй таблицы и указывающее системе базы данных, как выполнять соединение. Для разрешения этого соединения СУБД логически объединяет каждую строку в первой таблице с каждой строкой во второй таблице (это называется *декартовым произведением*). Затем к ним применяется критерий, указанный в условии ON или USING, чтобы отфильтровать строки, которые должны быть возвращены.

В условии ON *условие поиска* может использоваться внутри JOIN для определения логической проверки, результат которой должен иметь значение True (Истина) для возвращения любых двух связанных строк. Помните, что имеет смысл указывать условие поиска только в том случае, когда по крайней мере один столбец из первой таблицы сравнивается по крайней мере с одним столбцом второй таблицы. Хотя можно записать очень сложное условие поиска, обычно выполняется проверка простого условия равенства по первичному ключу одной таблицы со столбцами внешних ключей второй таблицы.

Рассмотрим простой пример. В хорошо спроектированной базе данных следует выделить сложные квалификационные имена во вторую таблицу, а затем связать имена с исходной таблицей предмета через простое значение ключа. Это делается с целью предотвращения ошибок ввода данных. Кто-либо, использующий вашу БД, скорее выберет из списка квалификационных имен, чем введет имя с клавиатуры (и, возможно, с ошибками) в каждой строке. Например, в примере базы данных "Рецепты" Recipe_Classes (Виды_рецептов) заносятся в таблицу отдельно от Recipes (Рецепты). На рис. 8.2 показано отношение между Recipe_Classes и Recipes.

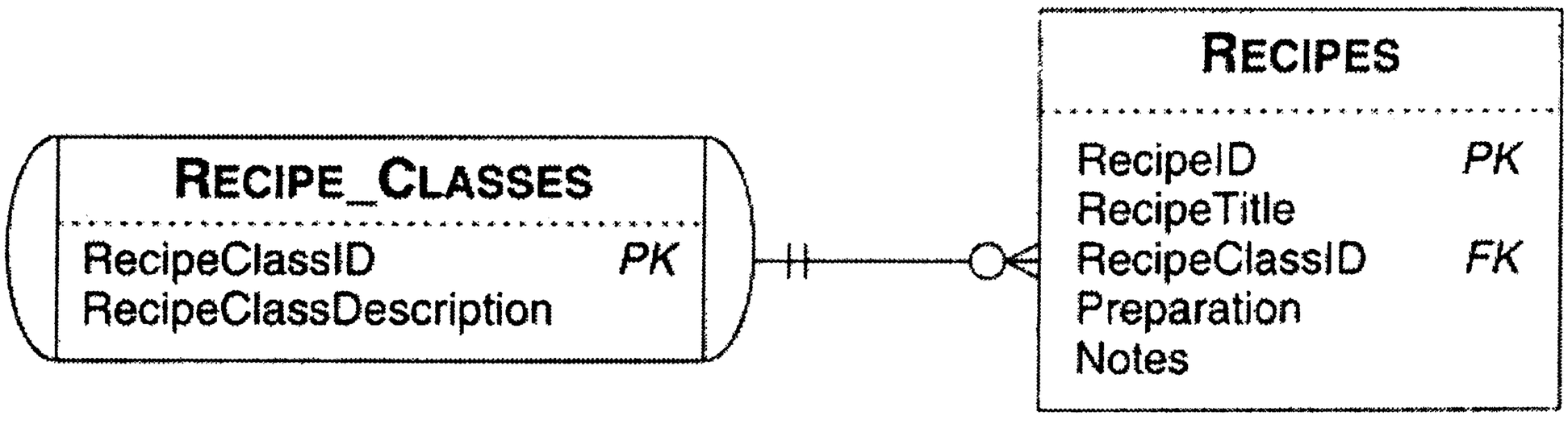


Рис. 8.2. *Описание видов рецептов и сами рецепты находятся в разных таблицах*

Когда из БД нужно извлечь информацию о рецептах и соответствующее Recipe Class Description (Описание вида рецепта), не требуется просматривать коды Recipe Class ID (Идентификатор вида рецепта) из таблицы Recipes. Для этого используется JOIN.

> **Внимание!** В данной главе используется метод "Запрос/Преобразование/Уточнение/SQL", введенный в главе 4.

"Show me the recipe title, preparation, and recipe class description of all recipes in my database."
("Показать название рецепта, приготовление и описание вида рецепта для всех рецептов в моей базе данных".)

Преобразование: Select recipe title, preparation, and recipe class description from the recipe classes table joined with the recipes table on recipe class ID in the recipe classes table matching recipe class ID in the recipes table
(Выбрать название рецепта, приготовление и описание вида рецепта из таблицы "Виды рецептов", соединенной с таблицей "Рецепты" по идентификатору вида рецепта в таблице "Виды рецептов", совпадающему с идентификатором вида рецепта в таблице "Рецепты")

Уточнение: Select recipe title, preparation, ~~and~~ recipe class description from ~~the~~ recipe classes ~~table~~ join~~ed with the~~ recipes ~~table~~ on recipe class ID ~~in the recipe classes table matching~~ = recipe class ID ~~in the recipes table~~
(Выбрать название рецепта, приготовление, описание вида рецепта из "Виды рецептов", соединенной с "Рецепты" по идентификатору вида рецепта = идентификатору вида рецепта)

SQL
```
SELECT RecipeTitle, Preparation,
    RecipeClassDescription
FROM Recipe_Classes
INNER JOIN Recipes
```

```
ON  Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID
```

Используя условие FROM нескольких таблиц, следует всегда полностью сверять имя каждого столбца с именем таблицы, когда бы он ни использовался, чтобы было абсолютно понятно, какой столбец из какой таблицы нужен. Здесь нам требуется уточнить имя RecipeClassID в условии ON , потому что существуют два столбца с именами RecipeClassID — один в таблице Recipes и второй в таблице Recipe_Classes. Имена столбцов RecipeTitle, Preparation или RecipeClassDescription в условии SELECT уточнять не требуется, потому что каждое из них появляется только один раз во всех таблицах. Если RecipeClassID нужно включить в вывод, то системе базы данных нужно указать, *какой именно* RecipeClassID нужен — из таблицы Recipe_Classes или из Recipes. Чтобы записать этот запрос со всеми полностью уточненными именами, укажите их следующим образом:

SQL

```
SELECT Recipes.RecipeTitle,
    Recipes.Preparation,
    Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
INNER JOIN Recipes
ON  Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID
```

Внимание! Хотя ключевое слово JOIN поддерживается большинством коммерческих реализаций SQL, некоторые реализации все же его не поддерживают. Если ваша база данных не поддерживает JOIN, то предыдущую задачу можно решить путем перечисления всех нужных таблиц в условии FROM, а затем перемещением *условия поиска* из условия ON в условие WHERE. В базах данных, которые не поддерживают JOIN, приведенная выше задача решается следующим способом:

```
SELECT  Recipes.RecipeTitle,  Recipes.Preparation,
     Recipe_Classes.RecipeClassDescription
FROM  Recipe_Classes,  Recipes
WHERE  Recipe_Classes.RecipeClassID  =
     Recipes.RecipeClassID
```

Для начинающего этот синтаксис для простых запросов, вероятно, интуитивно намного более понятен. Тем не менее синтаксис стандарта SQL позволяет полностью определить источник для конечного набора результатов целиком внутри условия FROM. Представьте себе условие FROM как полное определение связанного набора результатов, из которого СУБД получает ответ. В стандарте SQL условие WHERE используется только для фильтрации строк набора результатов, определенного условием FROM.

Не слишком сложно, не так ли? А как насчет условия USING, показанного на рис. 8.1? Если совпадающие столбцы в двух таблицах имеют одинаковые имена и требуется только соединить равные значения, воспользуйтесь условием USING и перечислите имена столбцов. Решим предыдущую задачу снова, на этот раз с использованием USING:

"Show me the recipe title, preparation, and recipe class description of all recipes in my database".
("Показать название рецепта, способ приготовления и описание вида рецепта для всех рецептов в моей базе данных".)

Преобразование: Select recipe title, preparation, and recipe class description from the recipe classes table joined with the recipes table using recipe class ID
(Выбрать название рецепта, способ приготовления и описание вида рецепта из таблицы "Виды рецептов", соединенной с таблицей "Рецепты", используя идентификатор вида рецепта)

Уточнение: Select recipe title, preparation, ~~and~~ recipe class description from ~~the~~ recipe classes ~~table~~ join~~ed with the~~ recipes ~~table~~ using recipe class ID
(Выбрать название рецепта, способ приготовления, описание вида рецепта из "Виды рецептов", соединенной с "Рецепты" по идентификатору вида рецепта)

SQL

```
SELECT Recipes.RecipeTitte, Recipes.Preparation,
    Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
INNER JOIN Recipes
USING (RecipeClassID)
```

Внимание! Стандарт SQL также определяет операцию NATURAL JOIN, которая связывает две определенные таблицы по совпадению всех столбцов с одинаковыми именами. Если общими столбцами являются только связывающие столбцы и база данных поддерживает NATURAL JOIN, то приведенную выше задачу можно решить таким образом:

```
SELECT Recipes.ReciepeTitle, Recipes.Preparation,
     Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
ATURAL INNER JOIN Recipes
```

Не определяйте условие ON или USING при использовании ключевого слова NATURAL.

Некоторые системы баз данных пока еще не поддерживают использование USING. Однако всегда можно получить тот же результат в условии ON и условии сравнения на равенство.

СУБД логически создает комбинацию каждой строки из первой таблицы с каждой строкой из второй таблицы, а затем применяет условие ON или USING. Это воспринимается как большой объем дополнительной работы для базы данных: вначале построить все комбинации, а затем отфильтровать потенциально несколько строк, удовлетворяющих условиям.

В остальном все современные системы баз данных оценивают условие JOIN целиком, прежде чем начать извлечение строк. Приведенный выше пример многие системы базы данных начинают решать с извлечения строки из Recipe_Classes. Затем база данных использует внутреннюю связь — индекс (если он был определен разработчиком таблиц) — для быстрого поиска всех строк из Recipe, удовлетворяющих условиям для первого Recipe_Classes, прежде чем перейти к следующей строке в Recipe_Classes. Другими словами, база данных использует "интеллектуальный" или "оптимизированный" план извлечения только тех строк, которые удовлетворяют условиям. Это кажется неважным, когда таблицы базы данных содержат всего несколько сотен строк, но приводит к серьезным отличиям, когда база данных должна распоряжаться сотнями тысяч строк!

Назначение корреляционных имен (псевдонимов) таблицам

Стандарт SQL определяет способ назначения псевдонима (алиаса) — называемого в стандарте *корреляционным именем* — любой таблице, перечисленной в условии FROM. Эта возможность может быть очень удобной при построении сложных запросов с использованием таблиц, имеющих длинные описательные имена. Таблице можно назначить короткое корреляционное имя, чтобы облегчить явное указание столбцов в таблице с длинным именем.

На рис. 8.3 показано, как назначить корреляционное имя таблице в условии FROM.

Для того чтобы назначить корреляционное имя таблице, укажите после ее имени необязательное ключевое слово AS, а затем корреляционное имя, которое нужно присвоить. (Как и для всех необязательных ключевых слов, AS рекомендуется использовать для того, чтобы облегчить чтение и понимание запроса.) После назначения таблице корреляционного имени это имя используется вместо ее исходного имени во всех других условиях, включая условие SELECT, условия поиска в условиях ON и WHERE и условие ORDER BY. Это может немного сбить с толку, потому что обычно имеется склонность записывать условие SELECT до того, как записывается условие FROM. Если планируется назначить таблице алиас в условии FROM, то этот алиас следует использовать при уточнении имен столбцов в условии SELECT.

Переформулируем запрос из нашего примера и используем корреляционные имена, просто чтобы увидеть, как это выглядит. "R" будет корреляционным именем для таблицы Recipes, а "RC" — для таблицы Recipe_Classes.

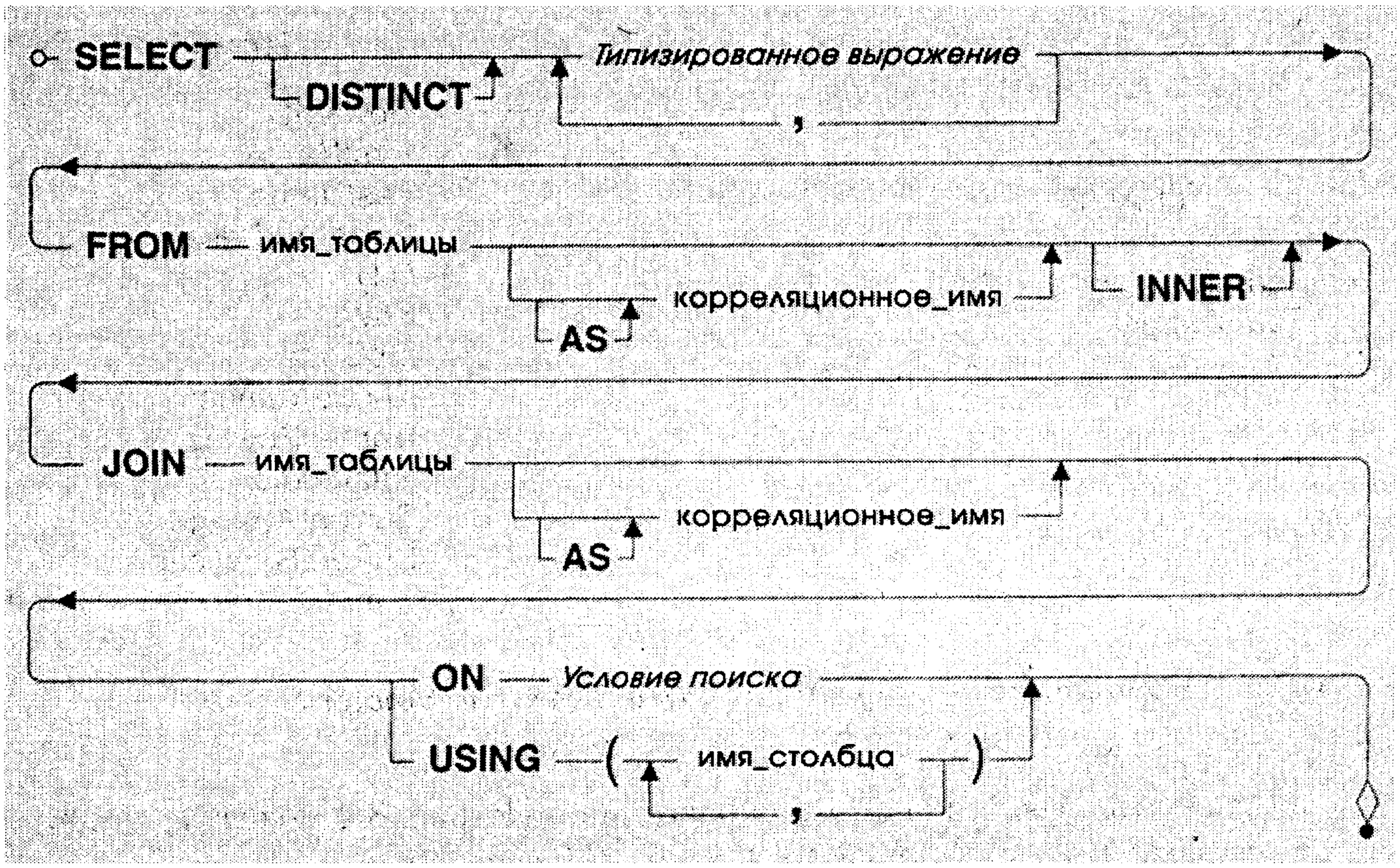


Рис. 8.3. *Назначение корреляционного имени таблице в условии FROM*

```
SQL                SELECT R.RecipeTitle, R.Preparation,
                      RC.RecipeClassDescription
                   FROM Recipe_Classes AS RC
                   INNER JOIN Recipes AS R
                   ON RC.RecipeClassID = R.RecipeClassID
```

Предположим, что нужно добавить фильтр, чтобы получить только рецепты вида "Главное блюдо" или "Десерт" (подробнее о фильтрах см. в главе 6). Раз назначено корреляционное имя, то следует продолжить использование этого имени во всех ссылках на таблицу. SQL будет следующим:

```
SQL                SELECT R.RecipeTitle, R.Preparation,
                      RC.RecipeClassDescription
                   FROM Recipe_Classes AS RC
                   INNER JOIN Recipes AS R
                   ON RC.RecipeClassID = R.RecipeClassID
                   WHERE RC.RecipeClassDescription = 'Main course'
                   OR RC.RecipeClassDescription = 'Dessert'
```

Совсем не обязательно назначать корреляционные имена всем таблицам. В предыдущем примере можно назначить корреляционное имя только для Recipes или Recipe_Classes, но не для обоих.

В некоторых случаях *необходимо* назначить корреляционное имя таблице в сложном соединении. Воспользуемся учебной базой данных "Лига игры в боулинг",

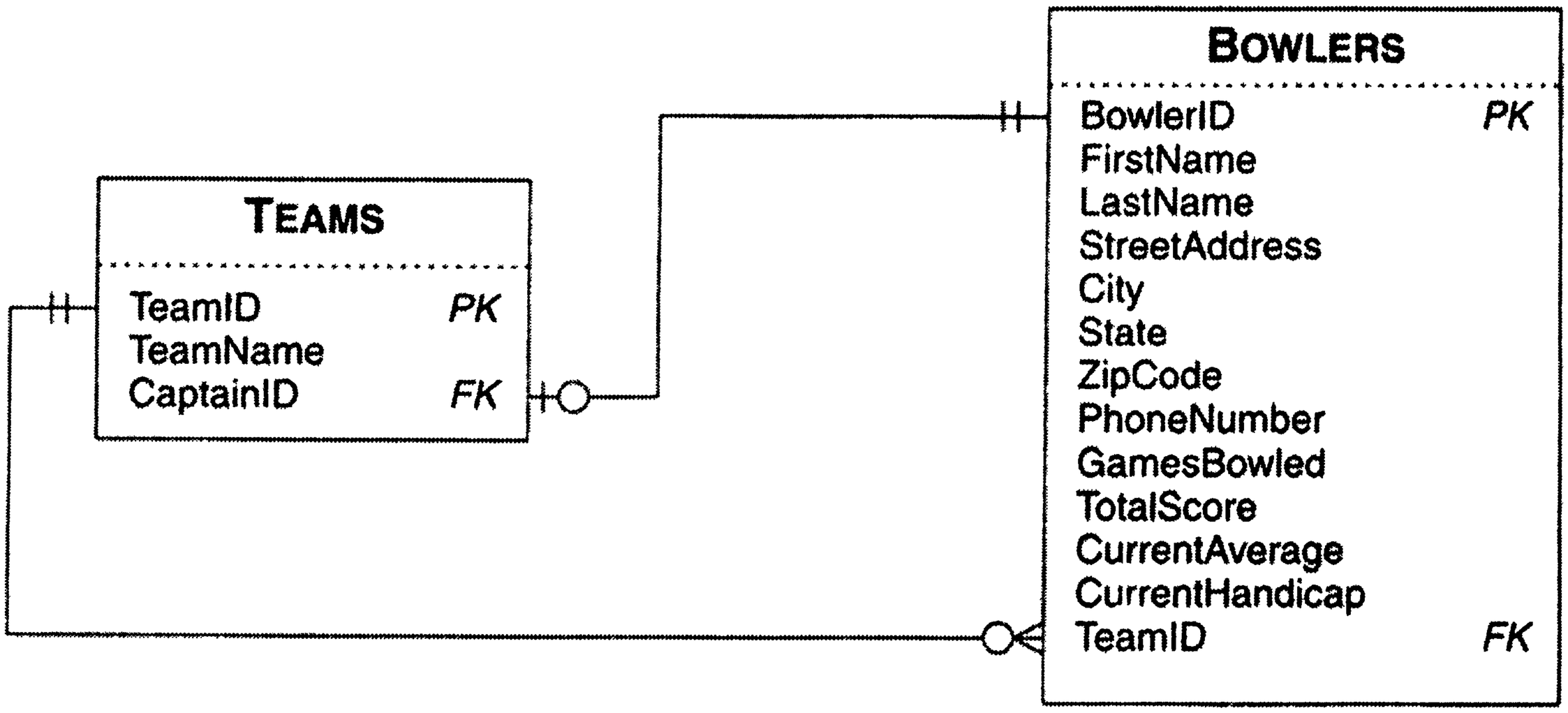


Рис. 8.4. *Отношение между таблицами Teams и Bowlers*

чтобы исследовать случай, когда это справедливо. На рис. 8.4 показано отношение между таблицами Teams (Команды) и Bowlers (Игроки в боулинг).

Как можно видеть, TeamID является внешним ключом в таблице Bowlers, который позволяет отыскивать информацию обо всех игроках команды. Один из игроков является капитаном, поэтому также имеется связь от BowlerID в таблице Bowlers к CaptainID в таблице Teams.

Если нужно получить список с названием команды, именем капитана команды и именами всех игроков в боулинг в одном запросе, то потребуется включить в запрос *две* копии таблицы Bowlers — одну для связывания с CaptainID для излечения имени капитана команды и другую для связывания по TeamID, чтобы получить список всех участников. В этом случае *следует* назначить имя-псевдоним одной или обеим копиям таблицы Bowlers, чтобы СУБД могла отличить копию, которая связана с именем капитана, и копию, которая предоставляет список всех участников команды. Позднее в этой главе мы покажем пример, который требует включения нескольких копий одной таблицы и назначения им имен-псевдонимов.

Вложение оператора SELECT

Сделаем задачу немного более интересной. Вместо того чтобы нанести немного голубого, а затем немного желтого на картину, чтобы получить зеленый, вначале смешаем нужный оттенок зеленого прямо на палитре.

В большинстве реализаций SQL можно заменять весь оператор SELECT на любое имя таблицы в условии FROM. Конечно, нужно назначить корреляционное имя, чтобы результат оценки вложенного запроса имел имя. На рис. 8.5 показано, как составить условие JOIN, используя вложенные операторы SELECT.

На рисунке оператор SELECT может включать все условия запроса, *кроме* условия ORDER BY. Также можно смешивать и согласовывать операторы SELECT с именами таблиц по любую сторону ключевых слов INNER JOIN.

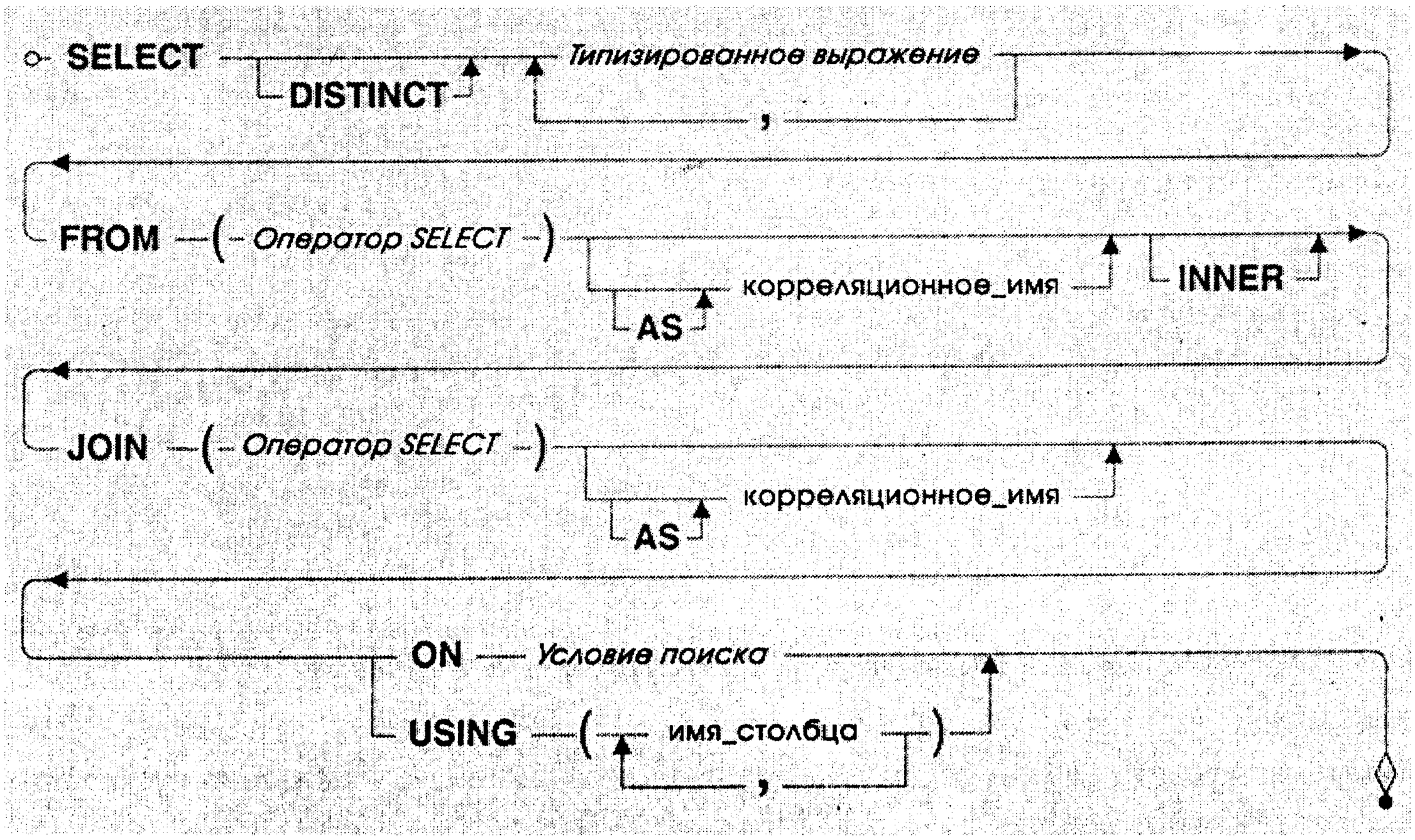


Рис. 8.5. *Замена имен таблиц в операторе SELECT на JOIN*

Еще раз рассмотрим Recipes и Recipe Classes. Наш запрос все еще требует только "Главное блюдо" и "Десерты". Вот снова запрос с таблицей Recipe_Classes, отфильтрованной в операторе SELECT, который является частью INNER JOIN:

SQL

```
SELECT R.RecipeTitle, R.Preparation,
    RCFiltered.ClassName
FROM (SELECT RecipeClassID,
    RecipeClassDescription AS ClassName
FROM Recipe_Classes AS RC
WHERE RC.ClassName = 'Main course' OR
    RC.ClassName = 'Dessert' AS RCFiltered
INNER JOIN Recipes AS R
ON RCFiltered.RecipeClassID = R.RecipeCIassID
```

Но осторожно! Имеется парочка "набрызганных пятен". Вначале, когда принимается решение заменить оператор SELECT на имя таблицы, не забудьте включить не только столбцы, которые желательно увидеть в окончательном результате, но также все связывающие столбцы, необходимые для выполнения JOIN. Именно поэтому во вложенном операторе присутствует как RecipeClassID, так и RecipeClassDescription. Просто ради шутки во вложенном операторе RecipeClassDescription мы присвоили имя-алиас ClassName. В результате условие SELECT запрашивает ClassName вместо RecipeClassDescription. Условие ON теперь ссылается на корреляционное имя вложенного оператора SELECT — RCFiltered — вместо исходного имени таблицы или корреляционного имени, назначенного таблице внутри вложенного оператора SELECT.

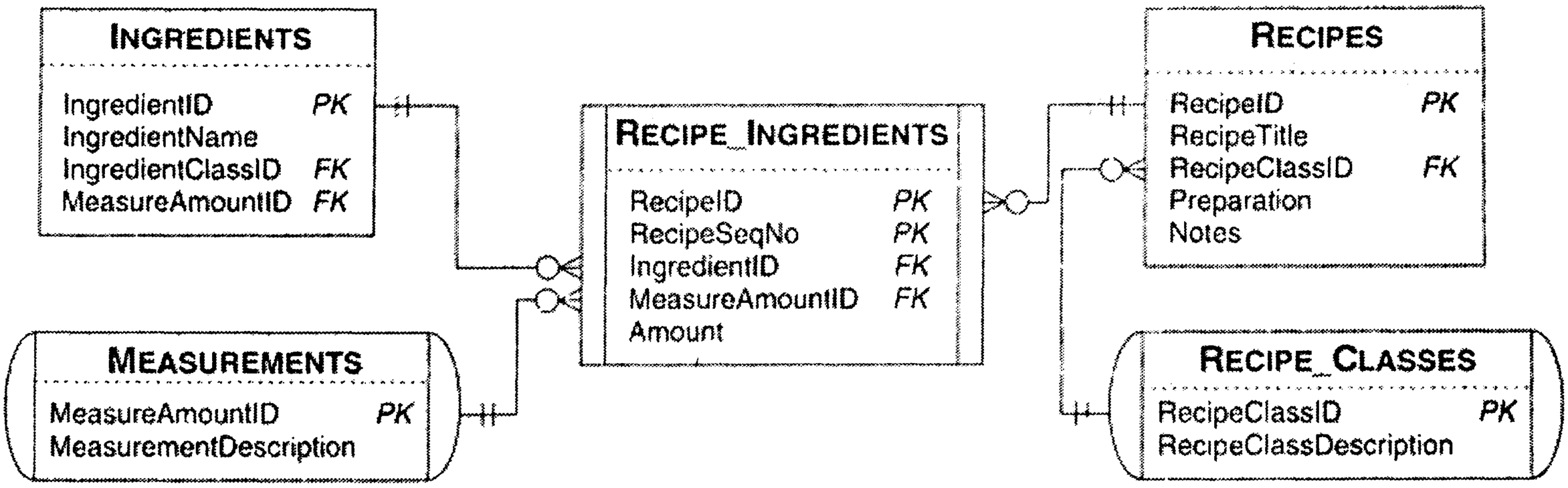


Рис. 8.6. *Таблицы из учебной базы данных "Рецепты", необходимые для извлечения всей информации о рецептах*

Если СУБД обладает очень "интеллектуальным" оптимизатором, то запрос, определенный таким способом, должен выполняться настолько же быстро, насколько предыдущий пример, где фильтр на Class Description (Описание вида) был наложен посредством условия WHERE после JOIN. Можно подумать, что СУБД вначале должна отфильтровать строки из Recipe_Classes, прежде чем пытаться искать какие-либо строки, удовлетворяющие условиям, в Recipes. Процесс, при котором вначале соединяются все строки из Recipe_Classes со строками из Recipes, удовлетворяющими условиям, а затем применяется фильтр, может выполняться намного медленнее. Если выполнение этого запроса займет намного больше времени, чем следует, перемещение условия WHERE в оператор SELECT в пределах JOIN может заставить систему БД выполнить вначале фильтрацию на Recipe_Classes.

Вложение операций JOIN в операции JOIN

Множество задач можно решить, просто связывая две таблицы, однако часто требуется связать три, четыре или более таблиц, чтобы получить все необходимые данные. Например, может потребоваться извлечь всю существенную информацию о рецептах — тип рецепта, имя рецепта и все ингредиенты этого рецепта — в одном запросе. На рис. 8.6 представлены таблицы, необходимые для ответа на этот запрос.

Похоже, что требуется извлечь данные из *пяти* различных таблиц! Отбросьте страх — это можно сделать, построив более сложное условие FROM, вкладывая условия JOIN в условия JOIN. Фокус вот в чем: везде, где можно определить имя таблицы, можно определить полное условие JOIN, заключенное в скобки. Рис. 8.7 является упрощенным вариантом рис. 8.3 (для организации простого соединения двух таблиц вместо условий с корреляционными именами было выбрано условие ON).

Для добавления третьей таблицы к соединению просто поместите открывающую скобку перед именем первой таблицы, добавьте закрывающую скобку после условия поиска и вставьте INNER JOIN, имя таблицы, ключевое слово ON и еще одно условие поиска. На рис. 8.8 показано, как это сделать.

INNER JOIN двух таблиц, заключенное в скобки, образует "логическую" таблицу или внутренний набор результатов. Этот набор теперь занимает место имени

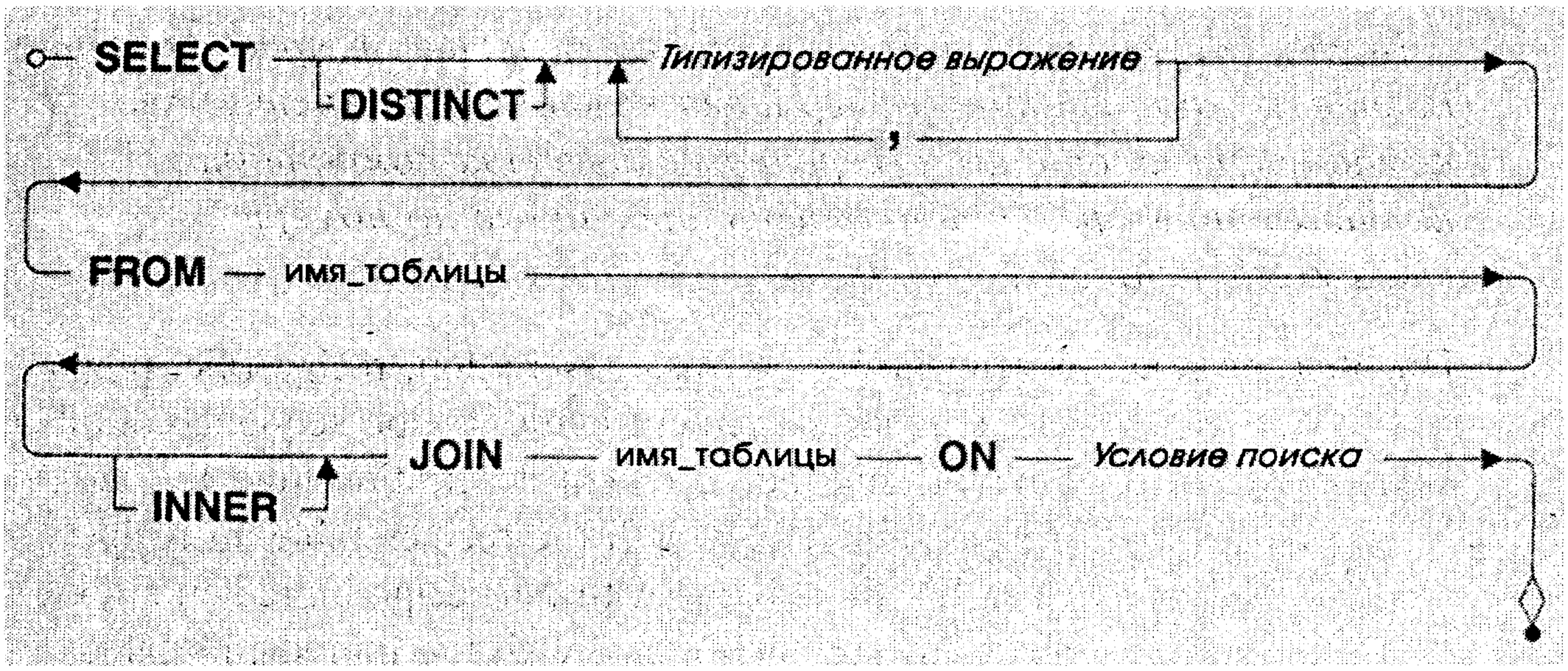


Рис. 8.7. *Простое условие INNER JOIN двух таблиц*

первой простой таблицы на рис. 8.7. Можно продолжить этот процесс заключения полного условия JOIN в скобки, а затем добавления другого ключевого условия JOIN, имени таблицы, ключевого слова ON и условия поиска до тех пор, пока не будут получены все необходимые наборы результатов. Составим запрос, для которого необходимы данные из всех таблиц, представленных на рис. 8.6, и посмотрим, какой будет результат.

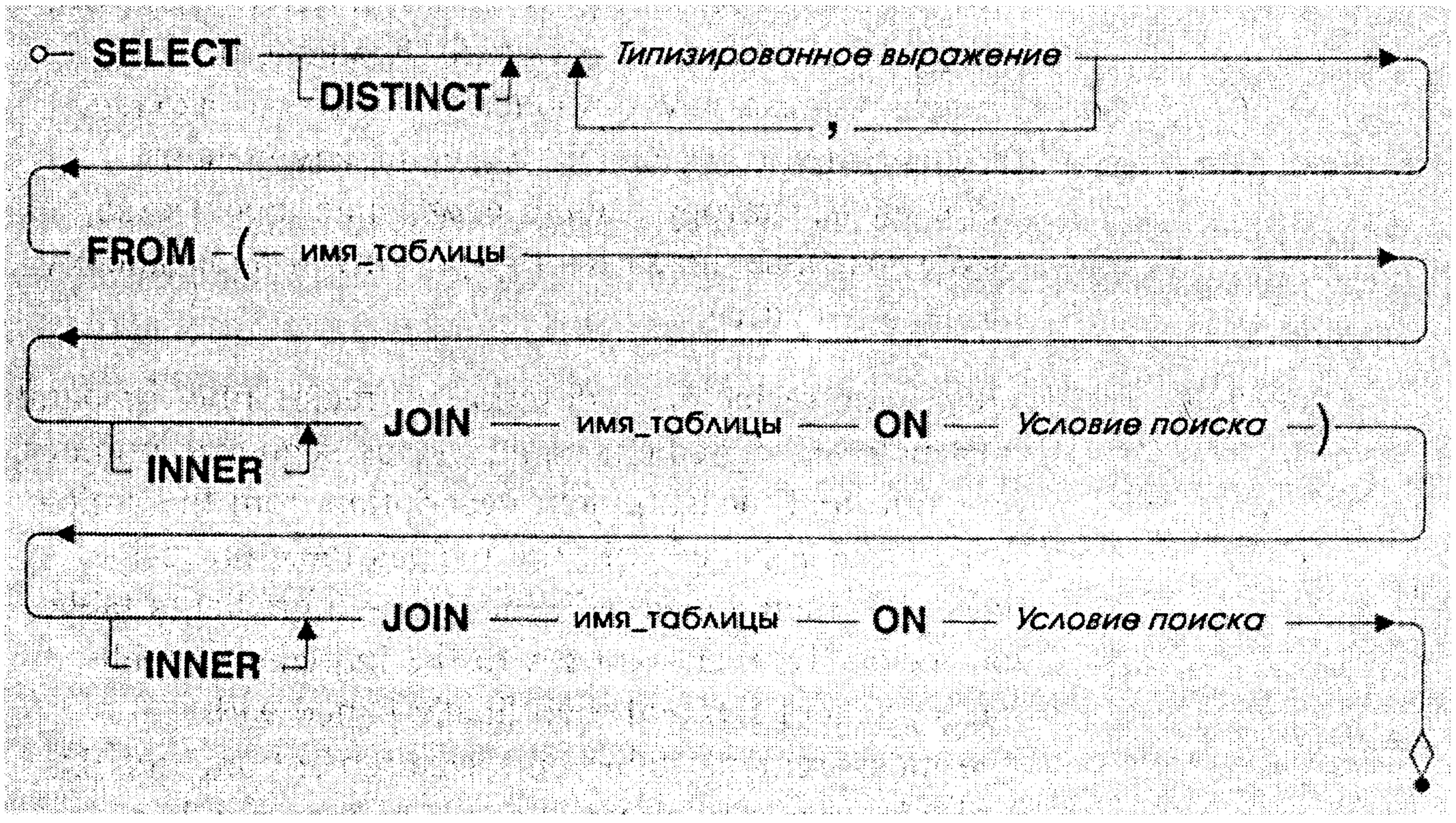


Рис. 8.8. *Простое условие INNER JOIN трех таблиц*

"I need the recipe type, recipe name, preparation instructions, ingredient names, ingredient step number, ingredient quantities, and ingredient measurements from my Recipes database, sorted in step number sequence".

("Нужен тип рецепта, имя рецепта, инструкции по приготовлению, названия компонентов, номер шага компонента, количество компонента и единицы измерения количества компонента из базы данных рецептов, отсортированные по номеру шага компонента".)

Преобразование: Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table joined with the recipes table on recipe class ID, then joined with the recipe ingredients table on recipe ID, then joined with the ingredients table on ingredient ID, and then finally joined with the measurements table on measurement amount ID, order by recipe title and recipe sequence number
(Выбрать описание вида рецепта, заголовок рецепта, инструкции по приготовлению, название компонента, порядковый номер рецепта, количество и описание единиц измерения из таблицы "Виды рецептов", соединенной с таблицей "Рецепты" по идентификатору вида рецепта, затем соединенной с таблицей "Компоненты рецепта" по идентификатору рецепта, затем соединенной с таблицей "Компоненты" по идентификатору компонента, и затем, наконец, соединенной с таблицей "Единицы измерения" по идентификатору единиц измерения количества, упорядоченные по заголовку рецепта и порядковому номеру рецепта)

Уточнение: Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description from ~~the~~ recipe classes ~~table~~ join~~ed with the~~ recipes ~~table~~ on recipe class ID, ~~then~~ join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID, ~~then~~ join~~ed with the~~ ingredients ~~table~~ on ingredient ID, ~~and then finally~~ join~~ed with the~~ measurements ~~table~~ on measurement amount ID, order by recipe title ~~and~~ recipe sequence number
(Выбрать описание вида рецепта, заголовок рецепта, приготовление, название компонента, порядковый номер рецепта, количество, описание единиц измерения из "Виды рецептов", соединенной с "Рецепты" по идентификатору вида рецепта, соединенной

с “Компоненты рецепта” по идентификатору рецепта,
соединенной с “Компоненты” по идентификатору
компонента, соединенной с “Единицы измерения”
по идентификатору единиц измерения количества,
упорядоченные по заголовку рецепта,
порядковому номеру рецепта)

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
    Recipes.RecipeTitle,  Recipes.Preparation,
    Ingredients.IngredientName,
    Recipe_Ingredients.RecipeSeqNo,
    Recipe_Ingredients.Amount,
    Measurements.MeasurementDescription
FROM (((Recipe_Classes
INNER JOIN Recipes
ON  Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID)
INNER JOIN Recipe_Ingredients
ON  Recipes.RecipeID =
    Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON  Measurements.MeasureAmountID =
    Recipe_Ingredients.MeasureAmountID
ORDER BY RecipeTitle.RecipeSeqNo
```

Здорово! Кто-нибудь испытывает желание влезть в это и добавить фильтр для вида рецептов “Основные блюда”? Если вы скажете, что необходимо добавить условие WHERE непосредственно перед условием ORDER BY, вы отгадали самый легкий способ сделать это.

Действительно, можно заменить взятый в целом JOIN двух таблиц в любом месте, где можно поместить просто имя таблицы. На рис. 8.8 подразумевалось, что вначале нужно соединить первую таблицу со второй, а затем полученный результат соединить с третьей таблицей. Также можно вначале соединить вторую и третью таблицы (если третья таблица фактически не связана ни со второй, ни с первой), а затем выполнить окончательное соединение с первой таблицей. На рис. 8.9 представлен этот альтернативный метод.

Давайте рассмотрим задачу с точки зрения живописи. При попытке получить нежно-зеленый цвет последовательность смешивания не имеет значения. Можно смешать белый цвет с синим, чтобы получить голубой, а затем добавить немного желтого. Или можно смешать синий цвет с желтым, чтобы получить зеленый, а затем добавить немного белого.

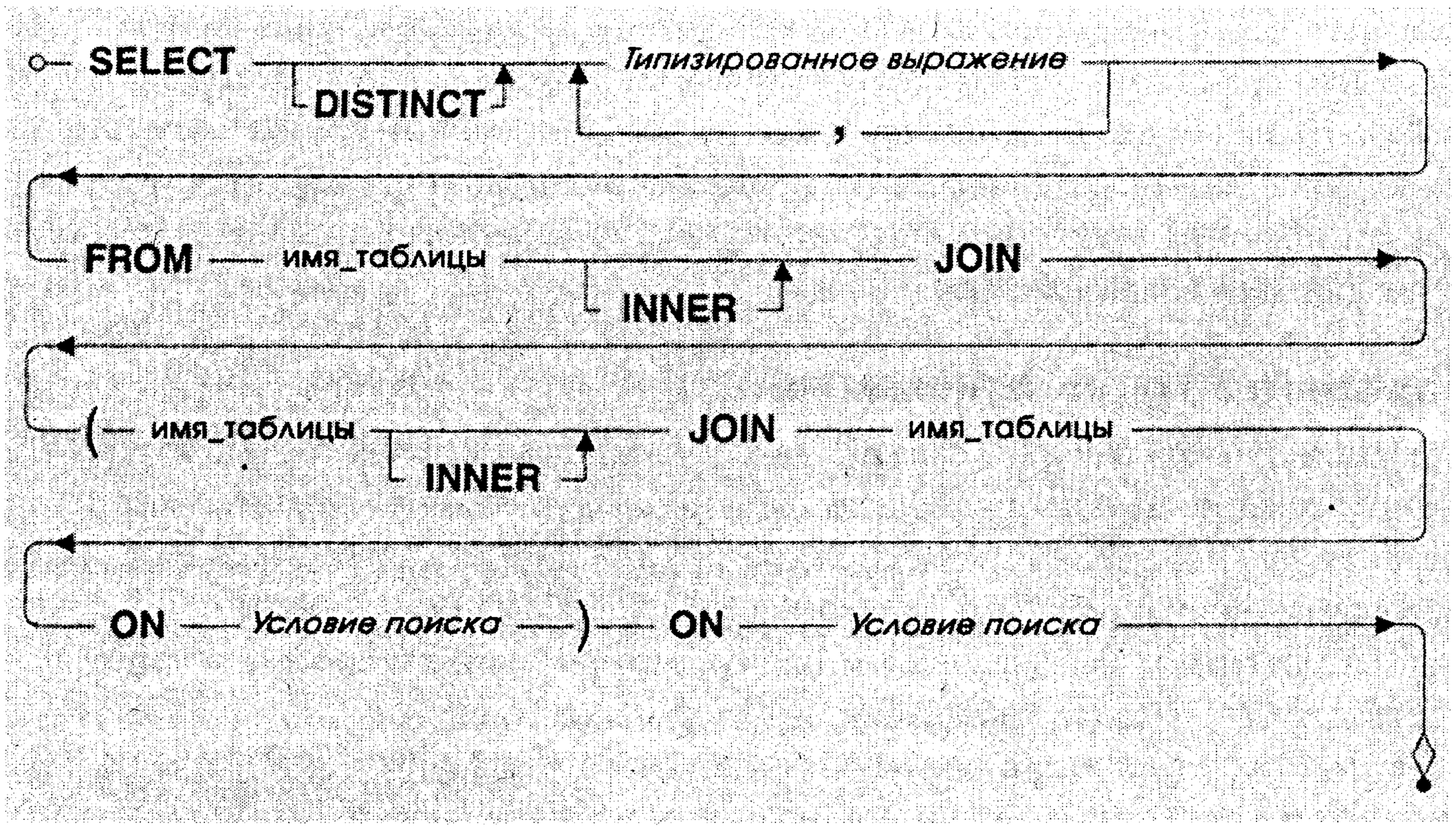


Рис. 8.9. *Соединение более чем двух таблиц в альтернативной последовательности*

Для составления только что представленного запроса с использованием пяти таблиц можно также записать в SQL следующее:

```
SQL                SELECT Recipe_Classes.RecipeClassDescription,
                      Recipes.RecipeTitle,  Recipes.Preparation,
                      Ingredients.IngredientName,
                      Recipe_Ingredients.RecipeSeqNo,
                      Recipe_Ingredients.Amount,
                      Measurements.MeasurementDescription
                   FROM Recipe_Classes
                   INNER JOIN (((Recipes
                   INNER JOIN Recipe_Ingredients
                   ON  Recipes.RecipeID =
                      Recipe_Ingredients.RecipeID)
                   INNER JOIN Ingredients
                   ON  Ingredients.  IngredientID =
                      Recipe_Ingredients.IngredientID)
                   INNER JOIN Measurements
                   ON  Measurements.MeasureAmountID =
                      Recipe_Ingredients.MeasureAmountID)
                   ON  Recipe_Classes.RecipeClassID =
                      Recipes.RecipeClassID
                   ORDER BY RecipeTitle,  RecipeSeqNo
```

Необходимо учитывать эту возможность, потому что можно встретить такой тип конструкции либо в запросах, написанных другими, либо в SQL, сконструированных в программном обеспечении построения запросов по образцу. Также оптимизаторы

некоторых БД чувствительны к последовательности определения условий JOIN. Если выполнение запроса с использованием многих соединений окажется слишком продолжительным для больших баз данных, можно заставить его выполняться быстрее, изменив последовательность операций JOIN в операторе SQL. Позже в данной главе мы покажем такие примеры с использованием непосредственного расположения соединений слева направо.

Проверяйте такие отношения!

Понимание отношений (связей) между таблицами исключительно важно. Когда столбцы нужных данных размещаются в разных таблицах, вероятно потребуется построить условие FROM такой сложности, как только что представленное, обеспечивая возможность сбора вместе всех частей способом, имеющим смысл. Если связь между таблицами и связывающие столбцы, которые устанавливают отношение, неизвестны, то вы ставите себя в безвыходное положение!

Во многих случаях для получения нужных данных необходимо последовательно "пройти" несколько отношений. Для примера упростим предыдущий запрос и попробуем получить только наименование рецепта и названия компонентов:

> *"Show me the names of all my recipes and the names of all the ingredients for each of those recipes".*
> *("Показать названия всех рецептов и названия всех компонентов для каждого из этих рецептов".)*
>
> Преобразование: Select the recipe title and the ingredient name from the recipes table joined with the recipe ingredients table on recipe ID, and then joined with the ingredients table on ingredient ID
> (Выбрать заголовок рецепта и название компонента из таблицы "Рецепты", соединенной с таблицей "Компоненты рецепта" по идентификатору рецепта, а затем соединенной с таблицей "Компоненты" по идентификатору компонента)
>
> Уточнение: Select ~~the~~ recipe title ~~and the~~ ingredient name from ~~the~~ recipes ~~table~~ join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID, ~~and then~~ join~~ed with the~~ ingredients ~~table~~ on ingredient ID
> (Выбрать заголовок рецепта, название компонента из "Рецепты", соединенной с "Компоненты рецепта" по идентификатору рецепта, соединенной с "Компоненты" по идентификатору компонента)
>
> SQL
>
```
SELECT Recipes.RecipeTitle,
    Ingredients.IngredientName
FROM (Recipes
INNER JOIN Recipe_Ingredients
```

```
ON  Recipes.RecipeID =
    Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID
```

Хотя столбцы из таблицы Recipe_Ingredients и не нужны, ее все равно нужно включить в запрос, поскольку таблицы Recipes и Ingredients связаны только *через* таблицу Recipe_Ingredients.

Применения условий INNER JOIN

Рассмотрим некоторые типы задач, которые можно решить с использованием INNER JOIN.

Поиск связанных строк

Наиболее обычным способом использования INNER JOIN является связывание таблиц вместе, чтобы можно было извлекать столбцы из различных таблиц. Ниже приводится список различных видов запросов из учебных баз данных, которые можно решить, используя INNER JOIN.

База данных заказов на закупку

"Показать поставщиков и товары, поставляемые ими нам".

"Привести список сотрудников и клиентов, для которых они зарегистрировали заказ".

База данных эстрадных мероприятий

"Показать на экране агентов и даты заключенных ими ангажементов".

"Привести список клиентов и эстрадных артистов, на которых они подали заявки".

"Найти эстрадных артистов, которые отыграли ангажементы для клиентов Бонниксен или Росалес".

База данных расписания занятий

"Вывести на экран здания и аудитории в каждом здании".

"Вывести список персонала факультета и предмет, который ведет каждый преподаватель".

База данных лиги игроков в боулинг

"Отобразить на экране команды игроков в боулинг и имя капитана каждой команды".

"Привести список команд игроков в боулинг и участников команды".

База данных рецептов

"Показать рецепты, содержащие говядину и чеснок".

"Вывести на экран компоненты рецептов, содержащих морковь".

Поиск значений, удовлетворяющих условиям

Немного более скрытое использование INNER JOIN состоит в поиске строк из двух и более таблиц или наборов результатов, совпадающих по одному или нескольким значениям, которые *не являются соответствующими* значениями ключей. В главе 7 мы обещали показать, как создать запрос, эквивалентный INTERSECT, с использованием INNER JOIN. Ниже приведен небольшой пример именно таких запросов, которые можно решить с помощью этого метода.

База данных заказов на закупку

"Показать клиентов и сотрудников с одинаковыми именами".

"Показать клиентов и сотрудников, проживающих в одном городе".

"Найти всех клиентов, заказавших мотоцикл, которые также заказали шлем".

База данных агентства эстрадных мероприятий

"Найти агентов и эстрадных артистов с одинаковым почтовым индексом".

"Привести список эстрадных артистов, отыгравших ангажементы для клиентов Бонниксен и Росалес".

База данных расписания занятий

"Показать студентов и их преподавателей, имеющих одинаковые имена".

"Показать студентов со средним баллом 85 или выше по курсу "Искусство" и средним баллом 85 или выше по курсу "Вычислительная техника".

База данных лиги игроков в боулинг

"Найти игроков в боулинг с одинаковым средним количеством очков".

"Найти игроков в боулинг с количеством предварительных очков 155 или выше как на Зандербирд Лэйнс, так и на Болеро Лэйнс.

База данных рецептов

"Найти компоненты, для которых по умолчанию используются одни и те же единицы измерения количества".

"Показать рецепты, содержащие говядину и чеснок".

В следующем разделе показано, как решить некоторые из перечисленных задач.

Примеры операторов

Ознакомимся с довольно устойчивым множеством примеров, в которых используется INNER JOIN. Они взяты из учебных баз данных и иллюстрируют использование INNER JOIN на двух таблицах, на нескольких таблицах и на соединениях по совпадающим значениям.

Внимание! Поскольку многие их приведенных примеров используют сложные соединения, СУБД может выбрать другой способ решения этих запросов. Поэтому несколько первых строк, которые показаны здесь, могут не совпадать точно с результатом, полученным вами, но общее количество строк должно быть одинаково.

Для упрощения процесса этапы преобразования и уточнения для всех последующих примеров объединены.

Две таблицы

Начнем с простых основных цветов и покажем примеры запросов, которые требуют INNER JOIN только по двум таблицам.

База данных заказов на закупку

"Display all products and their categories".
("Показать все товары и их категории".)

Преобразование/ Уточнение:

Select category description ~~and~~ product name from ~~the~~ categories ~~table~~ join~~ed with the~~ products ~~table~~ on category ID
(Выбрать описание категории, наименование товара из "Категории", соединенной с "Товары" по идентификатору категории)

SQL

```
SELECT Categones.CategoryDescription,
    Products.ProductName
FROM Categories
INNER JOIN Products
ON  Categories.CategoryID = Products.CategoryID
```

Products_And_Categories (40 строк)

CategoryDescription	ProductName
Accessories	Dog Ear Cyclecomputer
Accessories	Dog Ear Helmet Mount Mirrors
Accessories	Viscount C-500 Wireless Bike Computer
Accessories	Kryptonite Advanced 2000 U-Lock
Accessories	Nikoma Lok-Tight U-Lock
Accessories	Viscount Microshell Helmet
<< остальные строки >>	

База данных агентства эстрадных мероприятий

"Show me entertainers, start and end dates of their contracts, and the contract price".
("Показать эстрадных артистов, даты начала и окончания их контрактов и цену контракта".)

Преобразование/ Уточнение:

Select entertainer stage name, start date, end date ~~and~~ contract price from ~~the~~ entertainers ~~table~~ join~~ed with the~~ engagements ~~table~~ on entertainer ID
(Выбрать псевдоним эстрадного артиста, дату начала, дату окончания, цену контракта из "Эстрадные артисты", соединенной с "Ангажементы" по идентификатору эстрадного артиста)

SQL

```
SELECT Entertainers.EntStageName,
    Engagements.StartDate, Engagements.EndDate,
    Engagements.ContractPrice
FROM Entertainers
INNER JOIN Engagements
ON Entertainers.EntertainerID =
    Engagements.EntertainerID
```

Entertainers_And_Contracts (131 строка)

EntStageName	StartDate	EndDate	ContractPrice
Carol Peacock Trio	1999-07-18	1999-07-26	$1,670.00
Carol Peacock Trio	1999-07-31	1999-08-06	$1,940.00
Carol Peacock Trio	1999-08-13	1999-08-14	$410.00
Carol Peacock Trio	1999-08-20	1999-08-20	$140.00
Carol Peacock Trio	1999-09-12	1999-09-18	$680.00
Carol Peacock Trio	1999-10-22	1999-10-25	$410.00
Carol Peacock Trio	1999-10-28	1999-11-06	$1,400.00
Carol Peacock Trio	1999-11-07	1999-11-07	$320.00
<< остальные строки >>			

База данных расписания занятий

"List the subjects taught on Wednesday".
("Привести список предметов, преподаваемых в среду".)

Преобразование/ Уточнение:
Select subject name from ~~the~~ subject ~~table~~ join~~ed with the~~ classes ~~table~~ on subject ID where Wednesday schedule is = true
(Выбрать название предмета из "Предметы", соединенной с "Классы" по идентификатору предмета, где расписание на среду = истина)

SQL

```
SELECT DISTINCT Subjects.SubjectName
FROM Subjects
INNER JOIN Classes
ON Subjects.SubjectID = Classess.SubjectID
WHERE Classess.WednesdaySchedule = -1
```

Внимание! Поскольку в расписании на один и тот же день недели может быть указано несколько групп из одного класса, то включено ключевое слово DISTINCT для исключения повторов. Некоторые БД поддерживают ключевое слово "true", но мы предпочли более универсальный вариант: значение целого типа со всеми битами, установленными в единицу. Это значение равно –1. Если база данных системы сохраняет значение "истина/ложь" как один бит, то значение результата "истина" также можно проверять на значение 1. Значение для "ложь" — всегда число ноль (0).

Subjects_On_Wednesday (45 строк)

SubjectName
Advanced English Grammar
Art History
Biological Principles
Business Tax Accounting
Chemistry
Composition—Fundamentals
Composition—Intermediate
Computer Art
Database Management
<<остальные строки>>

База данных лиги игроков в боулинг

"Display bowling teams and the name of each team captain".
("Вывести на экран команды игры в боулинг и имя капитана каждой команды".)

Преобразование/ Уточнение:
Select team name ~~and~~ captain full name from ~~the~~ teams ~~table~~ join~~ed with the~~ bowlers ~~table~~ on team captain ID ~~equals~~ = bowler ID

(Выбрать название команды, полное имя капитана
из “Команды”, соединенной с “Игроки в боулинг”
по идентификатору капитана команды = идентификатору
игрока в боулинг)

SQL

```
SELECT Teams.TeamName, (Bowlers.BowlerLastName
    || ', ' || Bowlers.BowlerFirstName)
    AS CaptainName
FROM Teams
INNER JOIN Bowlers
ON Teams.CaptainID = Bowlers.BovlerID
```

Teams_And_Captains (8 строк)

TeamName	CaptainName
Marlins	Fournier, David
Sharks	Patterson, Ann
Terrapins	Morgenstern, Iris
Barracudas	Sheskey, Richard
Dolphins	Viescas, Suzanne
Orcas	Thompson, Sarah
Manatees	Viescas, Michael
Swordfish	Rosales, Joe

База данных рецептов

“Show me the recipes that have beef or garlic”.
(“Показать рецепты, в состав которых входит говядина или чеснок”.)

Преобразование/ Уточнение:

Select ~~unique~~ distinct recipe title from ~~the~~ recipes ~~table~~ join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID where ingredient ID ~~is~~ in ~~the list of beef and garlic IDs~~ (1,9)
(Выбрать неповторяющееся название рецепта
из “Рецепты”, соединенной с “Компоненты рецепта”
по идентификатору рецепта, где идентификатор
компонента в (1,9))

SQL

```
SELECT DISTINCT Recipes.RecipeTitle
FROM Recipes
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID = Recipe_Ingredients.RecipeID
WHERE Recipe_Ingredients.IngredientID IN (1, 9)
```

Внимание! Поскольку некоторые рецепты могут включать как говядину, так и чеснок, то для исключения возможных повторяющихся строк добавлено ключевое слово DISTINCT.

Beef_Or_Garlic_Recipes (5 строк)

RecipeTitle
Asparagus
Garlic Green Beans
Irish Stew
Pollo Picoso
Roast Beef

Более двух таблиц

В этот раз добавим какую-нибудь краску к палитре и составим запросы, для которых требуется JOIN более чем по двум таблицам.

База данных заказов на закупку

"Find all the customers who ever ordered a bicycle helmet".
("Найти всех клиентов, которые когда-либо заказывали шлем".)

Преобразование/ Уточнение:

Select customer first name, customer last name from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~then~~ join~~ed with the~~ order details ~~table~~ on order number, ~~then~~ join~~ed with the~~ products ~~table~~ on product number where product name ~~contains~~ LIKE '%Helmet%'
(Выбрать имя клиента, фамилию клиента из "Клиенты", соединенной с "Заказы" по идентификатору клиента, соединенной с "Детали заказа по номеру заказа, соединенной с "Товары" по номеру товара, где наименование товара соответствует шаблону '%Helmet%')

SQL

```
SELECT DISTINCT Customers.CustFirstName,
    Customers.CustLastName
FROM ((Customers
INNER JOIN Orders
ON  Customers.CustomerID = Orders .CustomerID)
INNER JOIN Order_Details
ON  Orders.OrderNumber =
    Order_Details.OrderNumber)
INNER JOIN Products
ON  Products.ProductNumber =
    Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%Helmet%'
```

Осторожно! Если СУБД чувствительна к регистру при выполнении поиска по символьным полям, следует быть внимательными при вводе критерия поиска и использовать правильный регистр для букв. Например, во многих системах баз данных 'helmet' — это не одно и то же, что 'Helmet'.

Внимание! Поскольку клиент может заказывать шлем несколько раз, то для исключения повторяющихся строк использовано ключевое слово DISTINCT.

Customers_Who_Ordered_Helmets (24 строки)

CustFirstName	CustLastName
Alaina	Hallmark
Allan	Davis
Amelia	Buchanan
Consuelo	Maynez
David	Callahan
David	Smith
Estella	Pundt
Gary	Hallmark
Gregory	Piercy
John	Viescas
<< остальные строки >>	

База данных агентства эстрадных мероприятий

"Find the entertainers who played engagements for customers Bonnicksen or Rosales".
("Найти эстрадных артистов, отыгравших ангажементы для клиентов Бонниксен или Росалес".)

Преобразование/ Уточнение:

Select ~~unique~~ distinct entertainer stage name from ~~the~~ entertainers ~~table~~ join~~ed with the~~ engagements ~~table~~ on entertainer ID, ~~then~~ join~~ed with the~~ customers ~~table~~ on customer ID where ~~the~~ customer last name ~~is~~ = 'Bonnicksen' or ~~the~~ customer last name ~~is~~ = 'Rosales'
(Выбрать неповторяющиеся псевдонимы эстрадного артиста из "Эстрадные артисты", соединенной с "Ангажементы" по идентификатору эстрадного артиста, соединенной с "Клиенты" по идентификатору клиента, где фамилия клиента = 'Bonnicksen' или фамилия клиента = 'Rosales')

SQL

```
SELECT DISTINCT
    Entertainers.EntStageName
FROM (Entertainers
INNER JOIN Engagements
ON  Entertainers.EntertainerID =
    Engagements.EntertainerID)
INNER JOIN Customers
ON  Customers.CustomerID =
    Engagements.CustomerID
WHERE Customers.CustLastName =
    'Bonnicksen'
OR  Customers.CustLastName =
    'Rosales'
```

Entertainers_For_Bonnicksen_Or_Rosales (9 строк)

EntStageName
Carol Peacock Trio
Country Feeling
Julia Schnebly
JV & the Deep Six
Katherine Ehrlich
Modern Dance
Saturday Revue
Susan McLain
Topazz

База данных лиги игроков в боулинг

"List all the tournaments, the tournaments matches, and the game results".
("Привести список всех турниров, матчи турниров и результаты игр".)

Преобразование/ Уточнение:

Select tourney ID, tourney location, match ID, lanes, odd lane team, even lane team, game number, game winner from ~~the~~ tournaments ~~table~~ join~~ed with the~~ tourney matches ~~table~~ on tourney ID, ~~then~~ join~~ed with the~~ teams ~~table aliased~~ as odd lane team on odd lane team ID ~~equals~~ = team ID, ~~then~~ join~~ed with the~~ teams ~~table aliased~~ as even lane team on even lane team ID ~~equals~~ = team ID, ~~then~~ join~~ed with the~~ match games table on match ID, ~~then~~ join~~ed with the~~ teams ~~table aliased~~ as winner on winning team ID ~~equals~~ = team ID
(Выбрать идентификатор турнира, место проведения турнира, идентификатор матча, дорожки, команду нечетных дорожек, команду четных дорожек, номер игры, победителя игры из "Турниры", соединенной с "Матчи турнира" по идентификатору турнира, соединенной с "Команды" как команда нечетных дорожек по идентификатору команды нечетных дорожек = идентификатору команды, соединенной с "Команды" как команда четных дорожек по идентификатору команды четных дорожек = идентификатору команды, соединенной с "Игры матча" по идентификатору матча, соединенной с "Команды" как победитель по идентификатору выигравшей команды = идентификатору команды)

SQL

```
SELECT Tournaments.TourneyID AS Tourney,
    Tournaments.TourneyLocation AS Location,
    Tourney_Matches.MatchID,
    Tourney_Matches.Lanes,
    OddTeam.TeamName AS OddLaneTeam,
    EvenTeam.TeamName AS EvenLaneTeam,
    Match_Games.GameNumber AS GameNo,
    Winner.TeamName AS Winner
FROM Teams AS Winner
INNER JOIN (Teams AS EvenTeam
INNER JOIN (Teams AS OddTeam
INNER JOIN ((Tournaments
INNER JOIN Tourney_Matches
ON Tournaments.TourneyID =
    Tourney_Matches.Tourney ID)
INNER JOIN Match_Games
ON Tourney_Matches.MatchID =
    Match_Games.MatchID)
```

```
ON  OddTeam.TeamID =
    Tourney_Matches.OddLaneTeamID)
ON  EvenTeam.TeamID =
    Tourney_Matches.EvenLaneTeamID)
ON  Winner.TeamID = Match_Games .WinningTeamID
```

Внимание! Это действительно забавный запрос, потому что для его выполнения требуется *три* копии одной таблицы (Teams). Необходимо, по крайней мере, присвоить корреляционные имена двум таблицам, чтобы все было законно, но мы пошли еще дальше и присвоили им всем имена-алиасы, чтобы отразить их специальные роли в запросе. Также при построении SQL структура, полученная на этапе Преобразование/Уточнение, соблюдалась не абсолютно точно. Это было сделано для того, чтобы продемонстрировать, что можно определить вложенные условия JOIN любым понравившимся способом до тех пор, пока не нарушаются отношения.

Tournament_Match_Game_Results (168 строк)

Tourney	Location	MatchID	Lanes	OddLane Team	EvenLane Team	GameNo	Winner
1	Red Rooster Lanes	1	01-02	Marlins	Sharks	1	Marlins
1	Red Rooster Lanes	1	01-02	Marlins	Sharks	2	Sharks
1	Red Rooster Lanes	1	01-02	Marlins	Sharks	3	Marlins
1	Red Rooster Lanes	2	03-04	Terrapins	Barracudas	1	Terrapins
1	Red Rooster Lanes	2	03-04	Terrapins	Barracudas	2	Barracudas
1	Red Rooster Lanes	2	03-04	Terrapins	Barracudas	3	Terrapins
1	Red Rooster Lanes	3	05-06	Dolphins	Orcas	1	Dolphins
1	Red Rooster Lanes	3	05-06	Dolphins	Orcas	2	Orcas
1	Red Rooster Lanes	3	05-06	Dolphins	Orcas	3	Dolphins
<<остальные строки >>							

База данных рецептов

"Show me the main course recipes and list all the ingredients".
("Показать рецепты основного блюда и список всех компонентов".)

Преобразование/ Уточнение:

Select recipe title, ingredient name, measurement description, ~~and~~ amount from ~~the~~ recipe classes ~~table~~ join~~ed with the~~ recipes ~~table~~ on recipe class ID, ~~then~~ join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID, ~~then~~ join~~ed with the~~ ingredients ~~table~~ on ingredient ID, ~~and finally~~ join~~ed with the~~ measurements ~~table~~ on measure amount ID, where recipe class description ~~is~~ = 'main course'
(Выбрать заголовок рецепта, название компонента, описание единиц измерения, количество из "Виды рецептов", соединенной с "Рецепты" по идентификатору вида рецепта, соединенной с "Компоненты рецепта" по идентификатору рецепта, соединенной с "Компоненты" по идентификатору компонента, соединенной с "Единицы измерения" по идентификатору количества в единицах измерений, где описание вида рецепта = 'main course')

SQL

```
SELECT Recipes.RecipeTitle, Ingredients.IngredientName,
    Measurements.MeasurementDescription,
    Recipe.Ingredients.Amount
FROM (((Recipe_Classes
INNER JOIN Recipes
ON  Recipes.RecipeClassID =
    Recipe_Classes.RecipeClassID)
INNER ]OIN Recipe_Ingredients
ON  Recipes.RecipeID =
    Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON  Measurements.MeasureAmountID =
    Recipe_Ingredients.MeasureAmountID
WHERE Recipe_Classes.RecipeClassDescription =
    'Main Course'
```

Осторожно! MeasureAmountID можно найти как в таблице Ingredients (Компоненты), так и в Recipe_Ingrediems (Компоненты рецепта). Если определить последнее соединение по MeasureAmountID, используя таблицу Ingredients вместо таблицы Recipe_Ingredients, то для компонента вместо единицы измерения количества, указанной для компонента в рецепте, будет получена единица измерения по умолчанию.

Main_Course_Ingredients (53 строки)

RecipeTitle	IngredientName	MeasurementDescription	Amount
Irish Stew	Beef	Pound	1
Irish Stew	Onion	Whole	2
Irish Stew	Potato	Whole	4
Irish Stew	Carrot	Whole	6
Irish Stew	Water	Quarts	4
Irish Stew	Guinness Beer	Ounce	12
Fettuccini Alfredo	Fettuccini Pasta	Ounce	16
Fettuccini Alfredo	Vegetable Oil	Tablespoon	1
Fettuccini Alfredo	Salt	Teaspoon	3
<< остальные строки >>			

Поиск совпадающих значений

Наконец добавим третье измерение в картину. В этом последнем наборе примеров показаны запросы, которые используют соединение по общим значениям из двух или более наборов результатов или таблиц. Если база данных поддерживает ключевое слово INTERSECT, то многие из таких задач можно решить с помощью пересечения наборов результатов.

База данных заказов на закупку

> *"Find all the customers who ordered a bicycle and who also ordered a helmet".*
> *("Найти всех клиентов, заказавших велосипед, а также заказавших шлем".)*

Этот запрос кажется достаточно простым, возможно, слишком простым. Поэтому зададим его другим образом, чтобы было более понятно, что требуется сделать базе данных.

> *"Find all the customers who ordered a bicycle, then find all the customers who ordered a helmet, and finally list the common customers so we know who ordered both a bicycle and a helmet".*
> *("Найти всех клиентов, заказавших велосипед, затем найти всех клиентов, заказавших шлем, и, наконец, привести список общих клиентов, чтобы знать, кто заказал как велосипед, так и шлем".)*

Преобразование 1: Select customer first name and customer last name from those common to the set of customers who ordered bicycles and the set of customers who ordered helmets

(Выбрать имена и фамилии клиентов из тех, которые являются общими для множества клиентов, заказавших велосипеды, и множества клиентов, заказавших шлемы)

Преобразование 2/ Уточнение:

Select customer first name ~~and~~ customer last name from (Select unique customer names from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~then~~ join~~ed with the~~ order details ~~table~~ on order number, ~~then~~ join~~ed with the~~ products ~~table~~ on product number where product name ~~contains~~ LIKE '%Bike') join~~ed with~~ (Select unique customer names from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~then~~ join~~ed with the~~ order details ~~table~~ on order number, ~~then~~ join~~ed with the~~ products ~~table~~ on product number where product name ~~contains~~ LIKE '%Helmet') on customer ID
(Выбрать имена и фамилии клиентов из
(Выбрать уникальные имена клиентов из "Клиенты", соединенной с "Заказы" по идентификатору клиента, соединенной с "Детали заказа" по номеру заказа, соединенной с "Товары" по номеру товара, где наименование товара соответствует шаблону '%Bike'), соединенной с (Выбрать уникальные имена клиентов из "Клиенты", соединенной с "Заказы" по идентификатору клиента, соединенной с "Детали заказа" по номеру заказа, соединенной с "Товары" по номеру товара, где наименование товара соответствует шаблону '%Helmet') по идентификатору клиента)

SQL

```
SELECT CustBikes.CustFirstName,
    CustBikes.CustLastName
FROM
(SELECT DISTINCT Customers.CustomerID,
    Customers.CustFirstName,
    Customers.CustLastName
FROM ((Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
ON Products.ProductNumber =
    Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%Bike')
```

```
AS CustBikes
INNER JOIN
(SELECT DISTINCT Customers.CustomerID
FROM ((Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
ON Products.ProductNumber =
   Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%Helmet')
AS CustHelmets
ON CustBikes.CustomerID = CustHelmets.CustomerID
```

Внимание! Второй вложенный оператор SELECT был упрощен и извлекал только идентификатор клиента, потому что только этот столбец необходим для работы INNER JOIN по двум множествам. Фактически можно было исключить JOIN с таблицей Customers (Клиенты) и извлечь CustomerID из таблицы Orders (Заказы). Эту проблему можно было также решить как INTERSECT двух множеств, но при этом потребовалось бы включить все выводимые столбцы в оба набора результатов, которые использовались при пересечении.

Customers_Both_Bikes_And_Helmets (24 строки)

CustFirstName	CustLastName
Suzanne	Viescas
Will	Thompson
Gary	Hallmark
Michael	Davolio
Kenneth	Peacock
John	Viescas
Laura	Callahan
Neil	Patterson
Margaret	Peacock
<< остальные строки >>	

База данных агентства эстрадных мероприятий

"List the entertainers who played engagements for both customers Bonnicksen and Rosales".
("Привести список эстрадных артистов, отыгравших ангажементы как для клиента Бонниксен, так и Росалес".)

Получить решение для Бонниксен *или* Росалес легко. Зададим запрос другим образом, чтобы было более понятно, что требуется сделать базе данных.

"Find all the entertainers who played an engagement for Bonnicksen, then find all the entertainers who played an engagement for Rosales, and finally list the common entertainers so we know who played an engagement for both".
("Найти всех эстрадных артистов, отыгравших ангажемент для Бонниксен, затем всех эстрадных артистов, отыгравших ангажемент для Росалес, и, наконец, привести список общих эстрадных артистов, чтобы узнать, кто отыграл ангажементы для обоих клиентов".)

Преобразование 1: Select entertainer stage name from those common to the set of entertainers who played for Bonnicksen and the set of entertainers who played for Rosales
(Выбрать псевдонимы эстрадных артистов, которые являются общими для множества эстрадных артистов, игравших для Бонниксен, и для множества эстрадных артистов, игравших для Росалес)

Преобразование 2/ Уточнение: Select entertainer stage name from (Select ~~unique~~ distinct entertainer stage names from ~~the~~ entertainers ~~table~~ join~~ed with the~~ engagements ~~table~~ on entertainer ID, ~~then~~ join~~ed with the~~ customers ~~table~~ on customer ID where customer last name ~~is~~ = 'Bonnicksen') join~~ed with~~ (Select ~~unique~~ distinct entertainer stage names from ~~the~~ entertainers ~~table~~ join~~ed with the~~ engagements ~~table~~ on entertainer ID, ~~then~~ join~~ed with the~~ customers ~~table~~ on customer ID where customer last name ~~is~~ = 'Rosales')
on entertainer ID
(Выбрать псевдонимы эстрадных артистов из (Выбрать неповторяющиеся псевдонимы из "Эстрадные артисты", соединенной с "Ангажементы" по идентификатору эстрадного артиста, соединенной с "Клиенты" по идентификатору клиента, где фамилия клиента = 'Бонниксен') соединенной с (Выбрать неповторяющиеся псевдонимы из "Эстрадные артисты", соединенной с "Ангажементы" по идентификатору эстрадного артиста, соединенной с "Клиенты" по идентификатору клиента, где фамилия клиента = 'Росалес') по идентификатору эстрадного артиста)

SQL

```
SELECT EntBonnicksen.EntStageName
FROM
(SELECT DISTINCT Entertainers.EntertainerID,
    Entertainers.EntStageName
```

```
FROM (Entertainers
INNER JOIN Engagements
ON Entertainers.EntertainerID =
   Engagements.EntertainerID)
INNER JOIN Customers
ON Customers.CustomerID = Engagements.CustomerID
WHERE Customers.CustLastName = 'Bonnicksen')
AS EntBonnicksen
INNER JOIN
(SELECT DISTINCT Entertainers.EntertainerID,
   Entertainers.EntStageName
FROM (Entertainers
INNER JOIN Engagements
ON Entertainers.EntertainerID =
   Engagements.EntertainerID)
INNER JOIN Customers
ON Customers.CustomerID = Engagements.CustomerID
WHERE Customers.CustLastName = 'Rosales')
AS EntRosales
ON EntBonnicksen.EntertainerID =
   EntRosales.EntertainerID
```

Внимание! Это другой пример запроса, который также можно решить с INTERSECT.

Entertainers_Bonnicksen_And_Rosales (4 строки)

EntStageName
Country Feeling
Katherine Ehrlich
Saturday Revue
Julia Schnebly

База данных расписания занятий

"Show me the students and teachers who have the same fist name".
("Показать студентов и преподавателей с одинаковыми именами".)

Преобразование/ Select student full name ~~and~~ staff full name from ~~the~~
Уточнение: students ~~table~~ join~~ed with the~~ staff ~~table~~ on first name
(Выбрать полное имя студента, полное имя преподавателя из "Студенты", соединенной с "Персонал" по имени)

SQL

```
SELECT (Students. StudFirstName || ' ' ||
   Students.StudLastName) AS StudFullName,
   (Staff. StfFirstName || ' ' ||
   Staff.StfLastName) AS StfFullName
FROM Students
INNER JOIN Staff
ON Students.StudFirstName = Staff.StfFirstName
```

Students_Staff_Same_FirstName (5 строк)

StudFullName	StfFullName
John Kennedy	John Leverling
Michael Viescas	Michael Davolio
Michael Viescas	Michael Hernandez
David Nathanson	David Callahan
David Nathanson	David Smith

База данных лиги игроков в боулинг

"Find the bowlers who had a raw score of 170 or better at both Thunderbird Lanes and Bolero Lanes".
("Найти игроков в боулинг с предварительным количеством очков 170 или выше как на Зандербирд Лэйнс, так и на Болеро Лэйнс".)

Да, это еще одна задача типа "решить пересечение, используя соединение". Зададим запрос другим образом, чтобы было более понятно, что должна сделать база данных.

"Find all the bowlers who had a raw score of 170 or better at Thunderbird Lanes, then find all the bowlers who had a raw score of 170 or better at Bolero Lanes, and finally list the common bowlers so we know who had good scores at both bowling alleys".
("Найти всех игроков в боулинг, которые имеют предварительное количество очков 170 и выше на Зандербирд Лэйнс, затем найти всех игроков в боулинг, которые имеют предварительное количество очков 170 и выше на Болеро Лэйнс, и, наконец, привести общий список этих игроков в боулинг, чтобы знать, у кого из них хорошее количество очков на обеих дорожках".)

Преобразование 1: Select bowler full name from those common to the set of bowlers who have a score of 170 or better at Thunderbird Lanes and the set of bowlers who have a score of 170 or better at Bolero Lanes
(Выбрать полное имя игрока в боулинг из списка множества игроков, имеющих количество очков 170 и выше на Зандербирд Лэйнс, и множества игроков, имеющих количество очков 170 и выше на Болеро Лэйнс)

Преобразование 2/ Уточнение: Select bowler full name from (Select ~~unique~~ distinct bowler ID ~~and~~ bowler full name from ~~the~~ bowlers ~~table~~ join~~ed with the~~ bowler scores ~~table~~ on bowler ID, ~~then~~ join~~ed with the~~ tourney matches ~~table~~ on match ID, ~~and finally~~ join~~ed with the~~ tournaments ~~table~~ on tourney

ID where tourney location ~~is~~ = 'Thunderbird Lanes' and raw score ~~is greater than or equal to~~ >= 170) join~~ed with~~ (Select unique bowler ID ~~and~~ bowler full name from ~~the~~ bowlers ~~table~~ join~~ed with the~~ bowler scores ~~table~~ on bowler ID, ~~then~~ join~~ed with the~~ tourney matches ~~table~~ on match ID, ~~and finally~~ join~~ed with the~~ tournaments ~~table~~ on tourney ID where tourney location ~~is~~ = 'Bolero Lanes' and raw score ~~is greater than or equal to~~ >= 170) on bowlerID
(Выбрать полное имя игрока в боулинг из (Выбрать неповторяющийся идентификатор игрока, полное имя игрока в боулинг из "Игроки в боулинг", соединенной с "Очки игроков" в боулинг по идентификатору игрока, соединенной с "Матчи турнира" по идентификатору матча, соединенной с "Турниры" по идентификатору турнира, где место проведения турнира = 'Thunderbird Lanes' и предварительное количество очков >= 170), соединенной с (Выбрать неповторяющийся идентификатор игрока, полное имя игрока в боулинг из "Игроки в боулинг", соединенной с "Очки игроков в боулинг" по идентификатору игрока, соединенной с "Матчи турнира" по идентификатору матча, соединенной с "Турниры" по идентификатору турнира, где место проведения турнира = 'Bolero Lanes' и предварительное количество очков >= 170) по идентификатору игрока в боулинг)

SQL

```
SELECT BowlerTbird.BowlerFullName
FROM
(SELECT DISTINCT Bowlers.BovlerID,
   (Bowlers.BowlerLastName || ', ' ||
   Bowlers.BowlerFirstName) AS BowlerFullName
FROM ((Bowlers
INNER JOIN Bowler_Scores
ON Bowlers.BowlerID = Bowler_Scores.BowlerID)
INNER JOIN Tourney_Matches
ON Tourney_Matches.MatchID = Bowler_Scores.MatchID)
INNER JOIN Tournaments
ON Tournaments.TourneyID = Tourney_Matches.TourneyID
WHERE Tournaments.TourneyLocation =
   'Thunderbird Lanes'
AND Bowler_Scores.RawScore >= 170)
AS BowlerTbird
INNER JOIN
```

```
(SELECT DISTINCT Bowlers.BowlerID,
    (Bowlers.BowlerLastName || ', ' ||
    Bowlers.BowlerFirstName) AS BowlerFullName
FROM ((Bowlers
INNER JOIN Bowler_Scores
ON Bowlers.BowlerID = Bowler_Scores.BowlerID)
INNER JOIN Tourney_Matches
ON Tourney_Matches.MatchID = Bowler_Scores.MatchID)
INNER JOIN Tournaments
ON Tournaments.TourneyID =
    Tourney_Matches.TourneyID
WHERE Tournaments.TourneyLocation = 'Bolero Lanes'
AND Bowler_Scores.RawScore >= 170)
AS BowlerBolero
ON BowlerTbird.BowlerID = BowlerBolero.BowlerID
```

Внимание! Поскольку игрок в боулинг мог иметь высокое количество очков на каждой из дорожек для игры в боулинг больше одного раза, то для исключения повторений добавлено ключевое слово DISTINCT.

Good_Bowlers_TBird_And_Bolero (10 строк)

BowlerFullName
Kennedy, John
Patterson, Neil
McLain, Susan
Patterson, Kathryn
Viescas, John
Piercy, Greg
Thompson, Mary
Thompson, Will
Patterson, Rachel
Pundt, Steve

База данных рецептов

"Display all the ingredients for recipes that contain carrots".
("Вывести на экран компоненты рецептов, содержащих морковь".)

Преобразование/ Уточнение:

Select recipe ID, recipe title, ~~and~~ ingredient name from ~~the~~ recipes ~~table~~ join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID, join~~ed with the~~ ingredients ~~table~~ on ingredient ID, ~~then finally~~ join~~ed with~~ (Select recipe ID from ~~the~~ ingredients ~~table~~ join~~ed with the~~ recipe ingredients ~~table~~ on ingredient ID where ingredient name ~~is~~ = 'carrot') on recipe ID
(Выбрать идентификатор рецепта, заголовок рецепта, название компонента из "Рецепты", соединенной с "Компоненты рецепта" по идентификатору рецепта,

соединенной с “Компоненты” по идентификатору компонента, соединенной с (Выбрать идентификатор рецепта из “Компоненты”, соединенной с “Компоненты рецепта” по идентификатору компоненты где название компонента = ‘морковь’) по идентификатору рецепта)

SQL

```
SELECT Recipes.RecipeID, Recipes.RecipeTitle,
    Ingredients.IngredientName
FROM ((Recipes
INNER JOIN Recipe_Ingredients
ON Recipes.RecipeID = Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID)
INNER JOIN
(SELECT Recipe_Ingredients.RecipeID
FROM Ingredients
INNER JOIN Recipe_Ingredients
ON Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID
WHERE Ingredients.IngredientName = 'Carrot')
AS Carrots
ON Recipes.RecipeID = Carrots.RecipeID
```

Recipes_Containing_Carrots (16 строк)

RecipeID	RecipeTitle	IngredientName
1	Irish Stew	Beef
1	Irish Stew	Onion
1	Irish Stew	Potato
1	Irish Stew	Carrot
1	Irish Stew	Water
1	Irish Stew	Guinness Beer
14	Salmon Filets in Parchment Paper	Salmon
14	Salmon Filets in Parchment Paper	Carrot
14	Salmon Filets in Parchment Paper	Leek
14	Salmon Filets in Parchment Paper	Red Bell Pepper
14	Salmon Filets in Parchment Paper	Butter
<< остальные строки >>		

Внимание! Используя подзапрос, можно упростить решение этого запроса (см. главу 11).

Итоги

В данной главе мы подробно обсудили, как связать две или более таблиц или наборов результатов по совпадающим значениям, а также концепцию JOIN и детали использования INNER JOIN. Мы объяснили, что "допустимо" использовать как критерий для JOIN, и предупредили о необходимости избегать бессмысленных соединений.

Здесь были показаны примеры объединения двух таблиц. Кроме того, мы рассмотрели принципы назначения корреляционных имен (псевдонимов) таблицам в условии FROM. Корреляционные имена назначаются для удобства или при включении одной и той же таблицы несколько раз, или при использовании вложенного оператора SELECT.

Вы узнали, как заменить ссылки на таблицу в операторе SELECT внутри условия FROM. Затем было показано, как расширить возможности, объединив более двух таблиц или наборов результатов. Обсуждение синтаксиса INNER JOIN было завершено повторным подчеркиванием важности хорошей структуры базы данных и понимания связей между своими таблицами.

На конкретных примерах мы обсудили, почему полезно INNER JOIN. Эти примеры включали примеры для двух таблиц, для нескольких таблиц и для соединений по совпадающим значениям. В следующей главе будет исследован другой вариант JOIN — OUTER JOIN.

Задачи для самостоятельного решения

Ниже приводятся формулировки запросов и имена решений этих запросов в учебных базах данных. Попрактикуйтесь немного и разработайте SQL для каждого запроса, а затем сверьте свой ответ с запросом, который сохранен нами в этих БД. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов,— важно, чтобы набор результатов был тем же.

База данных заказов на закупку

1. *"List customers and the dates they placed an order, sorted in order date sequence"*.
   ("Привести список клиентов и даты размещения ими заказов, упорядоченных по дате заказа".)
   (*Совет*: Здесь требуется JOIN двух таблиц.)
   Решение можно найти в Customers_And_OrderDates (944 строки).

2. *"List employees and the customers for whom they booked an order"*.
("Привести список сотрудников и клиентов, для которых они зарегистрировали заказы".)
(*Совет*: Здесь требуется JOIN более чем двух таблиц.)
Решение можно найти в Employees_And_Customers (211 строк).

3. *"Display all orders, the products in each order, and the amount owed for each product, in order number sequence"*.
("Вывести на экран все заказы, товары в каждом заказе и сумму, которую требуется заплатить за каждый товар, упорядочив их по номерам заказов".)
(*Совет*: Здесь требуется JOIN более чем двух таблиц.)
Решение можно найти в Orders_With_Products (4196 строк).

4. *"Show me the vendors and the products they supply to us for products that cost less than $100"*.
("Показать поставщиков и товары, поставляемые ими нам, для товаров стоимостью меньше 100 долларов".)
(*Совет*: Здесь требуется JOIN более чем двух таблиц.)
Решение можно найти в Vendors_And_Products_Less_Than_100 (66 строк).

5. *"Show me customers and employees that have the same first name"*.
("Показать клиентов и сотрудников с одинаковыми именами".)
(*Совет*: Здесь требуется JOIN по совпадающим значениям.)
Решение можно найти в Customers_Employees_Same_FirstName (4 строки).

6. *"Show me customers and employees that live in the same city"*.
("Показать клиентов и сотрудников, проживающих в одном и том же городе".)
(*Совет*: Здесь требуется JOIN по совпадающим значениям.)
Решение можно найти в Customers_Employees_Same_City (11 строк).

База данных агентства эстрадных мероприятий

1. *"Display agents and the engagement dates they booked, sorted by booking start date"*.
("Показать агентов и даты регистрации ангажементов, отсортированные по записанной дате начала".)
(*Совет*: Здесь требуется JOIN более чем двух таблиц.)
Решение можно найти в Agents_Booked_Dates (131 строки).

2. *"List customers and the entertainers they booked"*.
("Предоставить список клиентов и эстрадных артистов, заявленных ими".)
(*Совет*: Здесь требуется JOIN более чем двух таблиц.)
Решение можно найти в Customers_Booked_Entertainers (93 строки).

3. *"Findd the agents and entertainers that live in the same postal code"*.
("Найти агентов и эстрадных артистов, имеющих одинаковый почтовый индекс".)
(*Совет*: Здесь требуется JOIN по совпадающим значениям.)
Решение можно найти в Agents_Entertainers_Same_Postal (10 строк).

База данных расписания занятий

1. *"Display buildings and all the classrooms in each building"*.
("Отобразить на экране здания и все аудитории в каждом здании".)
(*Совет*: Здесь требуется JOIN двух таблиц.)
Решение можно найти в Buildinigs_Classrooms (44 строки).

2. *"List students and all the classes in which they are currently enrolled"*.
("Привести список студентов и все курсы лекций, на которые они записались на данный момент".)
(*Совет*: Здесь требуется JOIN более чем двух таблиц.)
Решение можно найти в Student_Enrollments (35 строк).

3. *"List the faculty staff and the subject each teaches"*.
("Предоставить список преподавателей факультета и преподаваемых ими дисциплин".)
(*Совет*: Здесь требуется JOIN более чем двух таблиц.)
Решение можно найти в Staff_Subjects (111 строк).

4. *"Show me the students who have a grade of 85 or better in art and who also have a grade of 85 or better in any computer course"*.
("Показать студентов, имеющих оценку 85 или выше по курсу "Искусство" и также имеющих оценку 85 или выше по курсу "Вычислительная техника".
(*Совет*: Здесь требуется JOIN по совпадающим значениям.)
Решение можно найти в Good_Art_CS_Students (1 строка).

База данных лиги игроков в боулинг

1. "List the bowling teams and all the team members".
(*"Предоставить список команд игры в боулинг и всех участников команд"*.)
(*Совет*: Здесь требуется JOIN двух таблиц.)
Решение можно найти в Teams_And_Bowlers (32 строки).

2. *"Display the bowlers, the matches they played in, and the bowler game scores"*.
("Вывести на экран игроков в боулинг, матчи в которых они играли, и очки игрока за игру".)
(*Совет*: Здесь требуется JOIN более чем двух таблиц.)
Решение можно найти в Bowler_Game_Scores (1344 строки).

3. *"Find the bowlers who have the same average"*.
("Найти игроков в боулинг, имеющих одинаковое среднее количество очков".)
(*Совет*: Здесь требуется JOIN по совпадающим значениям.)
Решение можно найти в Bowlers__Same_Average (70 строк).

База данных рецептов

1. *"List all the recipes for salads"*.
("Привести список всех рецептов салатов".)
(*Совет*: Здесь требуется JOIN двух таблиц.)
Решение можно найти в Salads (1 строка).

2. *"List all recipes that contain a Dairy ingredient"*.
("Привести список всех рецептов, содержащих в качестве компонента молочные продукты".)
(*Совет*: Здесь требуется JOIN более чем двух таблиц.)
Решение можно найти в Recipes_Containing_Dairy (2 строки).

3. *"Find the ingredient that use the same default measurement amount"*.
("Найти компоненты, для которых по умолчанию используются одинаковые единицы измерения количества".)
(*Совет*: Здесь требуется JOIN по совпадающим значениям.)
Решение можно найти в Ingredients_Same_Measure (628 строк).

4. *"Show me the recipes that have beef and garlic"*.
("Показать рецепты, содержащие говядину и чеснок".)
(*Совет*: Здесь требуется JOIN по совпадающим значениям.)
Решение можно найти в Beef_And_Garlic_Recipes (1 строка).

9

Внешние соединения

"Единственное различие между задачей и решением — это то, что решение люди понимают".

— Чарльз Франклин Кеттеринг,
изобретатель, 1876-1958

Вопросы, рассматриваемые в данной главе:

- Что представляет собой OUTER JOIN
- LEFT/RIGHT OUTER JOIN
- FULL OUTER JOIN
- Использование условий OUTER JOIN
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

В предыдущей главе мы рассмотрели все вопросы "внутренних" соединений — связывание двух или более таблиц или наборов результатов с помощью INNER JOIN для поиска всех строк, удовлетворяющих условию. Теперь самое время поговорить о "внешних" соединениях — связывании таблиц и поиске строк, не только удовлетворяющих, но и не удовлетворяющих условию.

Что представляет собой OUTER JOIN

Стандарт SQL определяет несколько типов операций JOIN для связи двух или более таблиц или наборов результатов. При выполнении операции OUTER JOIN база данных должна возвратить не только строки, совпадающие с определенным критерием, но также строки, не удовлетворяющие данному критерию, из одного либо из обоих множеств, которые нужно связать.

Предположим, например, что нужно извлечь информацию о студентах и курсах лекций, на которые они записались, из базы данных расписания занятий. Операция INNER JOIN возвращает имена только тех студентов, которые записались на некоторый курс лекций, и список лекций, на которые записались студенты. Эта операция

Рис. 9.1. *Возможное отношение между студентами и курсами лекций*

не возвращает имена студентов, которые уже приняты в колледж, но еще не записались на какие-либо курсы, и не возвращает курсы лекций, которые есть в расписании, но к которым пока еще студенты не проявили интереса.

А что если нужен список *всех* студентов и курсов лекций, на которые они записались, если таковые курсы имеются? Или наоборот, нужен список *всех* курсов лекций и имена студентов, которые записались на эти лекции, если имеются такие студенты. Для решения задач такого вида нужно использовать OUTER JOIN.

На рис. 9.1 с помощью диаграмм для множеств показано возможное отношение между студентами и лекциями. Как можно видеть, некоторое количество студентов не записалось еще ни на какой курс. Также имеются курсы лекций, на которые пока еще не записался ни один студент.

Рис. 9.2. *Студенты и курсы лекций, на которые они записались*

Если требуется список *всех* студентов с указанием курсов лекций, на которые они записались, будет получен набор результатов, подобный изображенному на рис. 9.2.

Может возникнуть вопрос: “Что будет возвращено для студентов, которые не записались ни на какой курс лекций?” Если вы помните концепцию значения Null, или “ничего” (см. главу 5), то поймете, что вы увидите. Если запрашиваются все студенты с указанием любых курсов лекций, то СУБД, обнаружив студента, не записавшегося ни на какой курс, возвратит значение Null во всех столбцах из таблицы Classes. Если обратиться к концепции разности двух множеств (см. главу 7), то строки со значением Null в столбцах из таблицы Classes будут представлять собой разность множества всех студентов и множества студентов, записавшихся на какой-либо курс лекций.

Таким же образом, если запрашиваются все курсы лекций и все студенты, записавшиеся на них, то строки со значениями Null в столбцах из таблицы Students будут представлять собой разность между множествами всех курсов лекций и множеством лекций, на которые зарегистрировались студенты. Использование OUTER JOIN с проверкой на значения Null является альтернативным способом найти разность двух множеств. В отличие от настоящей операции EXCEPT, для которой требуется

полное совпадение строк из двух множеств, в операции JOIN можно определить совпадение только по нескольким конкретным столбцам (обычно по первичному ключу и внешнему ключу).

LEFT/RIGHT OUTER JOIN

Обычно используется такой вид OUTER JOIN, в котором запрашиваются все строки из одной таблицы или набора результатов и все строки, удовлетворяющие условию, из второй таблицы или набора результатов. Для этого определяется либо LEFT OUTER JOIN, либо RIGHT OUTER JOIN.

В чем различие между "LEFT" и "RIGHT"? Вспомните, что для определения INNER JOIN по двум таблицам указывается имя первой таблицы, ключевое слово JOIN, а затем имя второй таблицы. Начиная построение запросов с использованием OUTER JOIN, стандарт SQL рассматривает первую указанную таблицу как "левую", а вторую как "правую". Поэтому, если нужны все строки из первой таблицы и все строки, удовлетворяющие условию, из второй таблицы, то используется LEFT OUTER JOIN. И наоборот, если нужны все строки из второй таблицы и все строки, удовлетворяющие условию, из первой таблицы, то указывается RIGHT OUTER JOIN.

Синтаксис

Рассмотрим синтаксические конструкции, необходимые для построения LEFT или RIGHT OUTER JOIN.

Использование таблиц

Начнем с определения OUTER JOIN с использованием таблиц. На рис. 9.3 представлена синтаксическая диаграмма для создания запроса с OUTER JOIN по двум таблицам.

Совершенно аналогично INNER JOIN все действие происходит в условии FROM (для простоты пока исключены условия WHERE и ORDER BY). Вместо одного имени таблицы указываются имена двух таблиц, которые связываются ключевым словом JOIN. Если не определить тип нужной операции JOIN, то СУБД предполагает, что требуется INNER JOIN (см. главу 8). В данном случае, поскольку требуется OUTER JOIN, следует явно указать, что нужно LEFT OUTER или RIGHT OUTER JOIN.

Внимание! Воспользовавшиеся полной диаграммой синтаксической структуры из приложения А, обнаружат, что операция **имя_таблицы JOIN имя_таблицы** описана как часть определенного термина *"соединенные таблицы"*. *Ссылка на таблицы* включает *соединенные таблицы*, а условие FROM оператора SELECT использует *ссылку на таблицу*. Эти сложные определения были "свернуты" в одну диаграмму, чтобы облегчить изучение простого соединения двух таблиц. Такой же метод упрощения используется в синтаксических диаграммах остальной части данной главы.

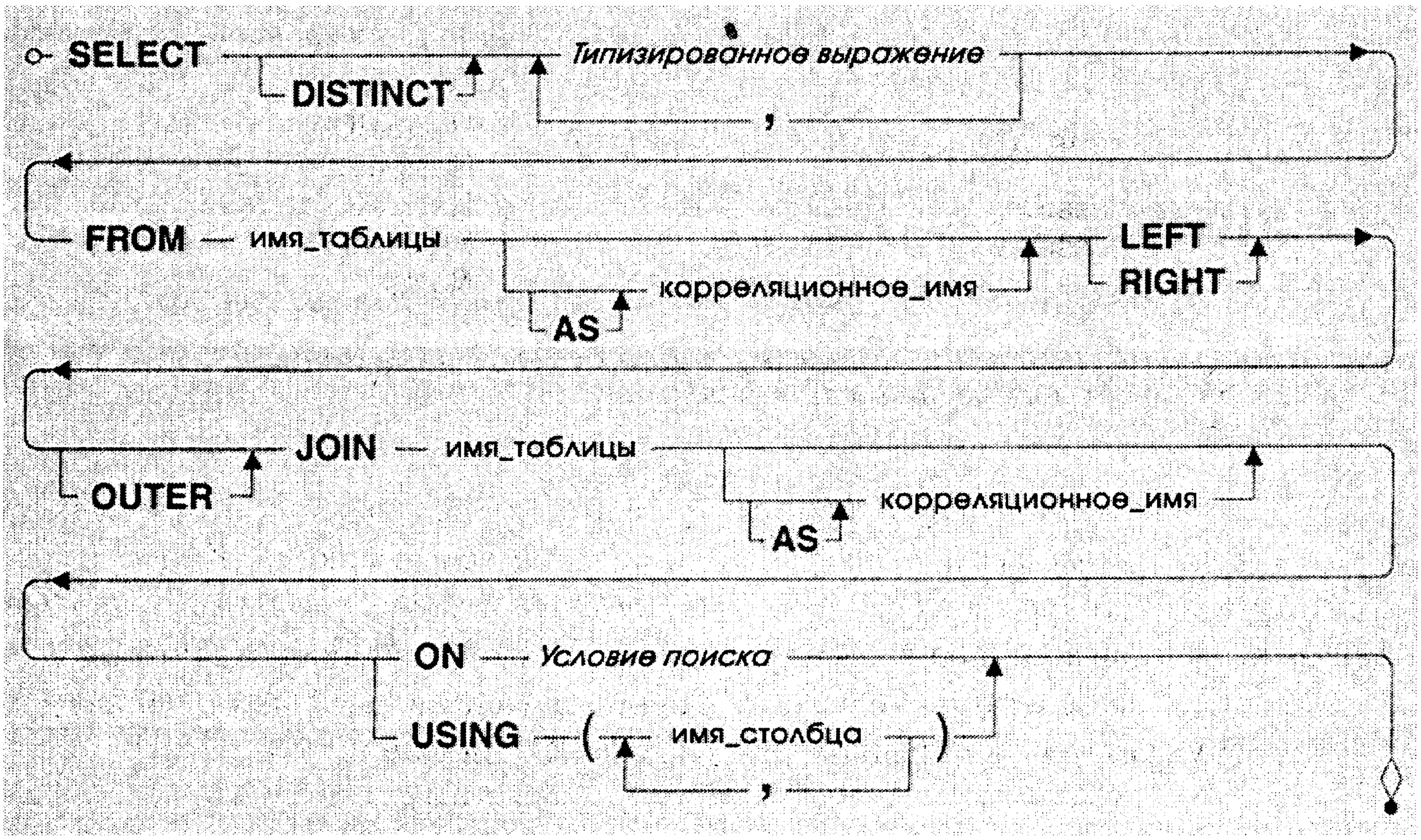


Рис. 9.3. *Определение OUTER JOIN по двум таблицам*

Необходимой частью любого JOIN является условие ON или USING, расположенное после имени второй таблицы и указывающее системе базы данных, как выполнять соединение. Для разрешения этого соединения СУБД логически объединяет каждую строку в первой таблице с каждой строкой во второй таблице (такое объединение называется *декартовым произведением*). Затем к ним применяется критерий, указанный в условиях ON или USING, чтобы найти строки, удовлетворяющие критерию, которые должны быть возвращены. Поскольку запрашивается OUTER JOIN, СУБД также возвращает строки, не удовлетворяющие условию, из "левой" либо "правой" таблицы.

В условии ON, внутри соединения, может использоваться *условие поиска* для определения логической проверки, результат которой должен иметь значение True для возвращения любых двух связанных строк. Указывать условие поиска имеет смысл только в том случае, если хотя бы один столбец из первой таблицы сравнивается по крайней мере с одним столбцом второй таблицы. Хотя можно записать очень сложное условие поиска, обычно определяется проверка простого условия равенства по первичному ключу одной таблицы со столбцами внешних ключей второй таблицы.

Для упрощения начнем с того же примера Recipes и Recipe_Classes, который использовался в предыдущей главе. Вспомните, что в хорошо спроектированной базе данных следует выделить сложные квалификационные имена во вторую таблицу, а затем связывать эти имена с исходной таблицей предмета через простое значение ключа. В учебной базе данных "Рецепты" Recipe_Classes заносятся в отдельную от Recipes таблицу. На рис. 9.4 показано отношение между Recipe_Classes и Recipes.

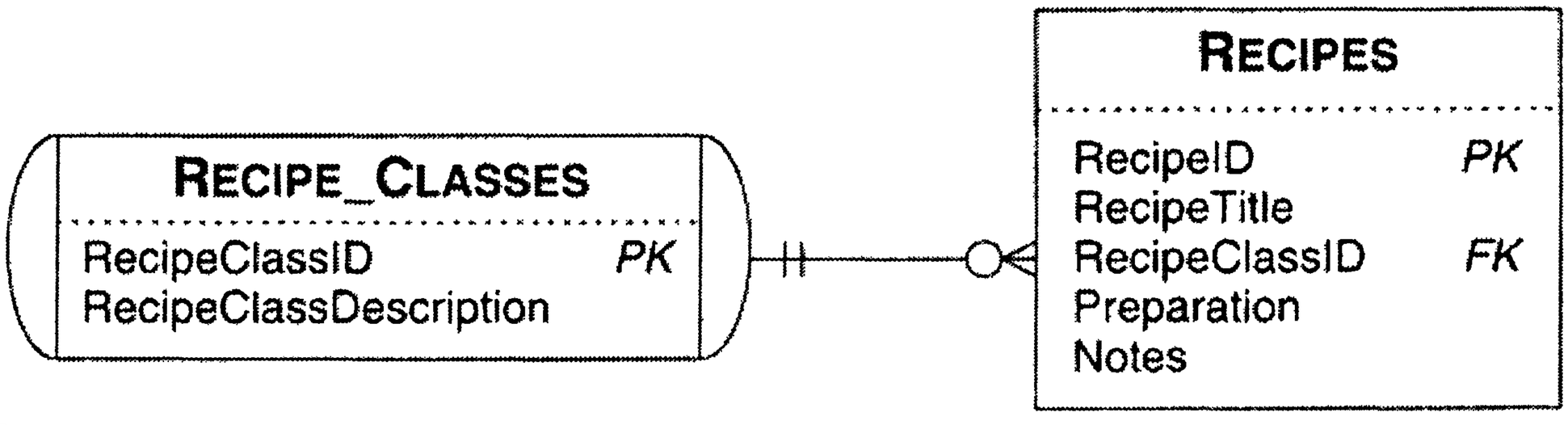


Рис. 9.4. *Виды рецептов находятся в отдельной таблице, отличной от таблицы рецептов*

Когда изначально заполняются виды рецептов для сохранения в базе данных, можно начать с ввода всех видов рецептов, которые вспоминаются. Когда введено большое количество рецептов, возможно, будет интересно выяснить, для каких видов рецептов пока еще не введено ни одного. Также стоит попробовать вывести список *всех* видов рецептов вместе с названиями рецептов, введенных для каждого из видов. Эту задачу можно решить с помощью OUTER JOIN.

> **Внимание!** В этой главе используется метод "Запрос/Преобразование/Уточнение/SQL", введенный в главе 4.

"Show me all of the recipe types and any matching recipes in my database". ("Показать все типы рецептов и соответствующие им рецепты в базе данных".)

Преобразование: Select recipe class description and recipe title from the recipe classes table outer joined with the recipes table on recipe class ID in the recipes classes table matching recipe class ID in the recipes table
(Выбрать описание вида рецепта и наименование рецепта из таблицы "Виды рецептов" с внешним соединением с таблицей "Рецепты" по идентификатору вида рецепта в таблице "Виды рецептов", совпадающие с идентификатором вида рецептов в таблице "Рецепты")

Уточнение: Select recipe class description ~~and~~ recipe title from ~~the~~ recipe classes ~~table~~ outer join~~ed with the~~ recipes ~~table~~ on recipe class ID ~~in the recipes classes table matching~~ = recipe class ID ~~in the recipes table~~
(Выбрать описание вида рецепта, название рецепта из "Виды рецептов" с внешним соединением с "Рецепты" по идентификатору вида рецепта = идентификатору вида рецептов)

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
    Recipes.RecipeTitle
FROM Recipe_Classes
LEFT OUTER JOIN Recipes
ON  Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID
```

Внимание! Хотя OUTER JOIN поддерживается большинством коммерческих реализаций SQL, некоторые реализации все же его не поддерживают. Если ваша база данных не поддерживает OUTER JOIN, задачу все-таки можно решить путем перечисления всех нужных таблиц в условии FROM, а затем перемещения *условия поиска* из условия ON в условие WHERE. Обратитесь к документации по своей базе данных, чтобы узнать конкретный нестандартный синтаксис, который требуется вашей базе данных для определения OUTER JOIN. Например, ранние версии Microsoft SQL Server поддерживают такой синтаксис (обратите внимание на звездочку в условии WHERE):

```
SELECT  Recipe_Classes.RecipeClassDescription,
     Recipes.RecipeTitle
FROM  Recipe_Classes,  Recipes
WHERE  Recipe_Classes.RecipeClassID  *=
     Recipes.RecipeClassID
```

Если используется Oracle, то синтаксис следующий (обратите внимание на знак “плюс” в условии WHERE):

```
SELECT  Recipe_Classes.RecipeClassDescription,
     Recipes.RecipeTitle
FROM  Recipe_Classes,  Recipes
WHERE  Recipe_Classes.RecipeClassID  =
     Recipes.RecipeClassID(+)
```

Для начинающего этот синтаксис, вероятно, интуитивно намного более понятен в применении к простым запросам. Однако синтаксис стандарта SQL позволяет полностью определить источник для конечного набора результатов целиком внутри условия FROM. Рассматривайте условие FROM как полное определение связанного набора результатов, из которого СУБД получает ответ. В стандарте SQL условие WHERE используется только для отфильтровывания строк набора результатов, определенного условием FROM. Также, поскольку специальный синтаксис для определения OUTER JOIN через условие WHERE отличается для разных продуктов, при работе с несколькими нестандартными продуктами необходимо изучить несколько различных синтаксисов.

При использовании нескольких таблиц в условии FROM помните о необходимости полного уточнения имени каждого столбца с именем таблицы при каждом его использовании, чтобы было абсолютно ясно, какой столбец из какой таблицы нужен. Необходимо также *уточнить* имя RecipeClassID в условии ON, потому что имеются два столбца с именем RecipeClassID — один в таблице Recipes и еще один в таблице Recipe_Classes.

Если выполнить приведенный выше запрос в учебной базе данных Recipes, то он должен вернуть 16 строк. Поскольку в эту базу данных не вводили каких-либо рецептов для вида Soup (Суп), то в строке, для которой RecipeClassDescription имеет значение Soup, в столбце RecipeTitle будет получено значение Null. Чтобы найти только эту строку, воспользуйтесь следующим подходом:

> *"List the recipe classes that do not yet have any recipes".*
> *("Привести список видов рецептов, в которых пока еще нет ни одного рецепта".)*

Преобразование: Select recipe class description from the recipe classes table outer joined with the recipes table on recipe class ID where recipe ID is empty
(Выбрать описание вида рецепта из таблицы "Вид рецепта" с внешним соединением с таблицей "Рецепты" по идентификатору вида рецепта, где идентификатор рецепта пуст)

Уточнение: Select recipe class description from ~~the~~ recipe classes ~~table~~ outer join~~ed with the~~ recipes ~~table~~ on recipe class ID where recipe ID is ~~empty~~ NULL
(Выбрать описание вида рецепта из "Вид рецепта" с внешним соединением с "Рецепты" по идентификатору вида рецепта, где идентификатор рецепта NULL)

SQL

```
SELECT Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
LEFT OUTER JOIN Recipes
ON  Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID
WHERE Recipes.RecipeID IS NULL
```

Подумав немного, можно понять, что здесь просто выполняется операция DIFFERENCE или EXCEPT (см. главу 7) с использованием JOIN. Это несколько похоже на запрос: "Показать все виды рецепты за исключением (EXCEPT) тех, которые уже имеются в таблице "Рецепты". Диаграмма для множеств на рис. 9.5 поможет представить, о чем идет речь.

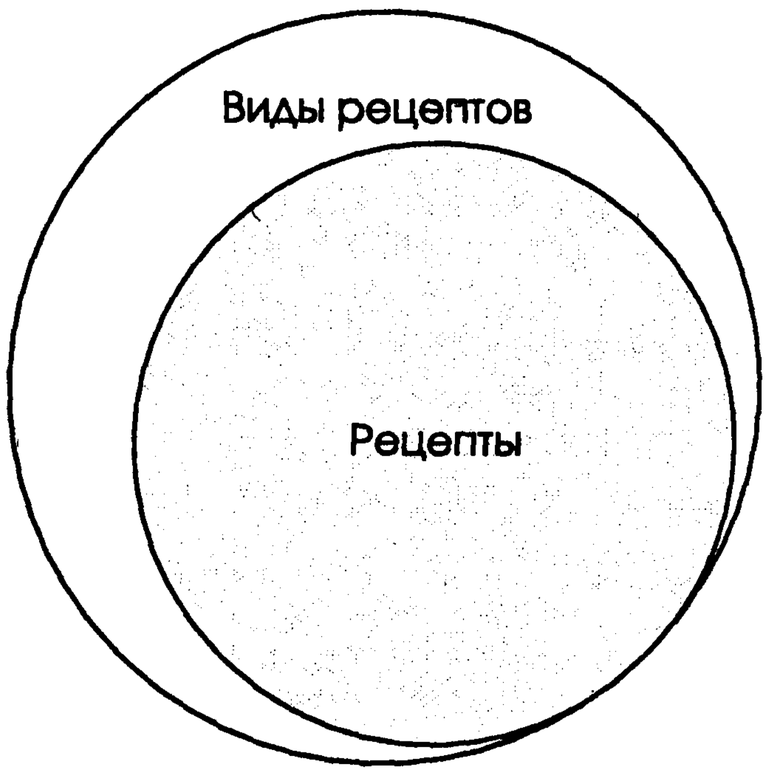


Рис. 9.5. *Возможное отношение между видами рецептов и рецептами*

На рис. 9.5 все рецепты относятся к какому-то виду, но существуют некоторые виды, для которых рецепты еще не определены. При добавлении проверки IS NULL запрашиваются все строки в более светлом внешнем круге, для которых отсутствуют совпадения во множестве рецептов, представленном в более темном внутреннем круге.

Диаграмма для OUTER JOIN в таблице на рис. 9.3 также содержит необязательное условие USING. Если столбцы, удовлетворяющие условию, в обеих таблицах имеют одинаковое имя и нужно просто соединить их по равным значениям, то можно воспользоваться условием USING и перечислить имена столбцов. Решим предыдущую задачу заново, на этот раз используя USING:

"Display the recipe classes that do not yet have any recipes".
("Вывести на экран виды рецептов, в которых пока еще нет рецептов".)

Преобразование: Select recipe class description from the recipe classes table outer joined with the recipes table using recipe class ID where recipe ID is empty
(Выбрать описание вида рецепта из таблицы "Виды рецептов" с внешним соединением с таблицей "Рецепты", используя идентификатор вида рецепта, где идентификатор рецепта является пустым)

Уточнение: Select recipe class description from ~~the~~ recipe classes ~~table~~ outer join~~ed with the~~ recipes ~~table~~ using recipe class ID where recipe ID is ~~empty~~ NULL
(Выбрать описание вида рецепта из "Виды рецептов" с внешним соединением с "Рецепты", используя идентификатор вида рецепта, где идентификатор рецепта NULL)

SQL
```
SELECT Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
LEFT OUTER JOIN Recipes
USING (RecipeClassID)
WHERE Recipes.RecipeID IS NULL
```

Синтаксис для USING гораздо проще, не так ли? Однако некоторые системы баз данных пока еще не поддерживают USING. Если в вашей базе данных не может использоваться USING, всегда можно получить тот же результат, применив условие ON и условие сравнения на равенство.

Внимание! Стандарт SQL также определяет вид операции JOIN, называемый NATURAL JOIN. Операция NATURAL JOIN связывает две определенные таблицы по совпадению всех столбцов с одинаковыми именами. Если общими столбцами являются только связывающие столбцы и база данных поддерживает NATURAL JOIN, то приведенную выше задачу можно решить таким образом:

```
SELECT Recipe_Classes.RecipeClassDescription
FROM Recipe_Classes
NATURAL LEFT OUTER JOIN Recipes
WHERE Recipes.RecipeID IS NULL
```

Не определяйте условие ON или USING при использовании ключевого слова NATURAL.

Вложение оператора SELECT

Большинство реализаций SQL позволяют заменять взятый в целом оператор SELECT на имя таблицы в условии FROM. Конечно, затем обязательно нужно назначить корреляционное имя (см. раздел о присвоении имен-псевдонимов в главе 8), чтобы результат оценки вложенного запроса имел имя. На рис. 9.6 представлен процесс составления условия OUTER JOIN с использованием вложенных операторов SELECT.

Оператор SELECT может включать все условия запроса, за исключением условия ORDER BY. Также можно смешивать и согласовывать операторы SELECT с именами таблиц по любую сторону ключевого слова OUTER JOIN.

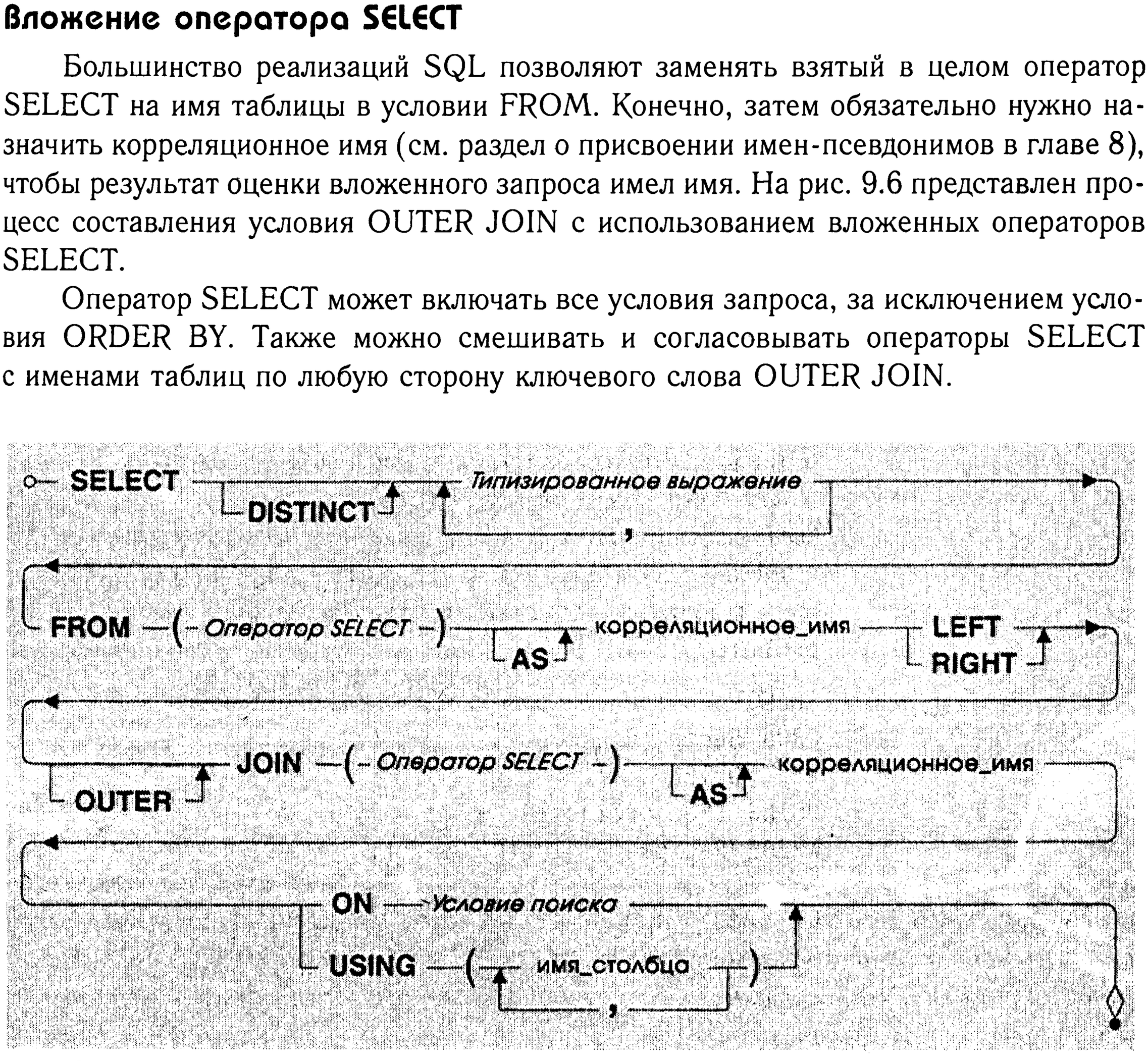


Рис. 9.6. *OUTER JOIN с использованием операторов SELECT*

Еще раз рассмотрим таблицы Recipes и Recipe Classes. Для данного примера предположим, что нас интересуют только виды Salads (Салаты), Soups (Супы) и Main courses (Главные блюда). Вот запрос с использованием таблицы Recipe_Classes, отфильтрованной в операторе SELECT, который является частью INNER JOIN:

```
SQL                 SELECT RCFiltered.ClassName, R.RecipeTitle
                    FROM
                    (SELECT RecipeClassID, RecipeClassDescription
                    AS ClassName
                    FROM Recipe_Classes AS RC
                    WHERE RC.ClassName = 'Salads'
                    OR RC.ClassName = 'Soup'
                    OR RC.ClassName = 'Main Course') AS RCFiltered
                    LEFT OUTER JOIN Recipes AS R
                    ON RCFiltered.RecipeClassID = R.RecipeClassID
```

Будьте осторожны при использовании оператора SELECT в условии FROM. В первую очередь, когда принимается решение заменить оператор SELECT на имя таблицы, обязательно нужно включить не только столбцы, которые должны присутствовать в окончательном результате, но также все связывающие столбцы, необходимые для выполнения JOIN. Именно поэтому во вложенном операторе присутствуют как RecipeClassID, так и RecipeClassDescription. Просто для шутки присвоим RecipeClassDescription псевдоним "ClassName" во вложенном операторе. В результате условие SELECT запрашивает ClassName, а не RecipeClassDescription. Условие ON ссылается теперь на корреляционное имя вложенного оператора SELECT — RCFiltered — вместо исходного имени таблицы или корреляционного имени, присвоенного таблице во вложенном операторе SELECT.

Если данный запрос выполнить для реального примера базы данных Recipes, то получим одну строку с RecipeClassDescription для Soup со значением Null, возвращенным для RecipeTitle, потому что в этом примере базы данных рецепты супов отсутствуют. Также легко можно было бы построить оператор SELECT по таблице Recipes в правой части OUTER JOIN. Например, можно запросить все рецепты, содержащие слово "beef" (говядина) в своем заголовке:

```
SQL                 SELECT RCFiltered.ClassName, R.RecipeTitle
                    FROM
                    (SELECT RecipeClassID, RecipeClassDescription
                    AS ClassName
                    FROM Recipe_Classes AS RC
                    WHERE RC.ClassName = 'Salads'
                    OR RC.ClassName = 'Soup'
                    OR RC.ClassName = 'Main Course') AS RCFiltered
                    LEFT OUTER JOIN (SELECT Recipes.RecipeClassID,
                        Recipes.RecipeTitle
```

```
FROM Recipes
WHERE Recipes.RecipeTitle LIKE '%beef%') AS R
ON RCFiltered.RecipeClassID = R.RecipeClassID
```

LEFT OUTER JOIN запрашивает *все* строки из набора результатов или таблицы, указанной в левой части JOIN, независимо от того, существуют ли строки, удовлетворяющие условию, в правой части. Предыдущий запрос не только возвращает строку Soup со значением Null в RecipeTitle (поскольку в базе данных вообще отсутствуют рецепты супов), но также и строку Salad со значением Null. Можно было бы предположить, что в базе данных также отсутствуют и рецепты салатов, но фактически *они там есть*, только отсутствуют салаты с "beef" в заголовке рецепта!

Внимание! Возможно, вы обратили внимание, что можно ввести полное *условие поиска* как часть условия ON в JOIN. Совершенно правильно, поэтому вполне законно по стандарту SQL решить приведенную выше задачу следующим образом:

```
SELECT  Recipe_Classes.RecipeClassDescription,
        Recipes.RecipeTitle
FROM  Recipe_Classes
LEFT  OUTER  JOIN  Recipes
ON  Recipe_Classes.RecipeClassID  =  Recipes.RecipeClassID
AND
      (Recipe_Classes.RecipeClassDescription  =  'Salads'
OR  Recipe_Classes.RecipeClassDescription  =  'Soup'
OR  Recipe_Classes.RecipeClassDescription  =  'Main  Course')
AND  Recipes.RecipeTitle  LIKE  "%beef%"
```

К сожалению, некоторые основные реализации SQL решают эту задачу неверно или вообще не воспринимают этот синтаксис! Поэтому рекомендуется всегда указывать в *условии поиска* условия ON только такие критерии, которые сравнивают столбцы из двух таблиц или наборов результатов. Если нужно отфильтровать строки из базовых таблиц, делайте это в отдельном *условии поиска* в условии WHERE во вложенном операторе SELECT.

Вложение условий JOIN в условия JOIN

Хотя многие задачи можно решить, просто связывая две таблицы, очень часто требуется связать три, четыре или более таблиц, чтобы получить все данные, необходимые для решения запроса. Например, может потребоваться извлечь всю существенную информацию о рецептах — тип рецепта, название рецепта и все компоненты для него — в одном запросе. Зная, как использовать OUTER JOIN, можно также получить список *всех* видов рецептов — даже тех, для которых пока

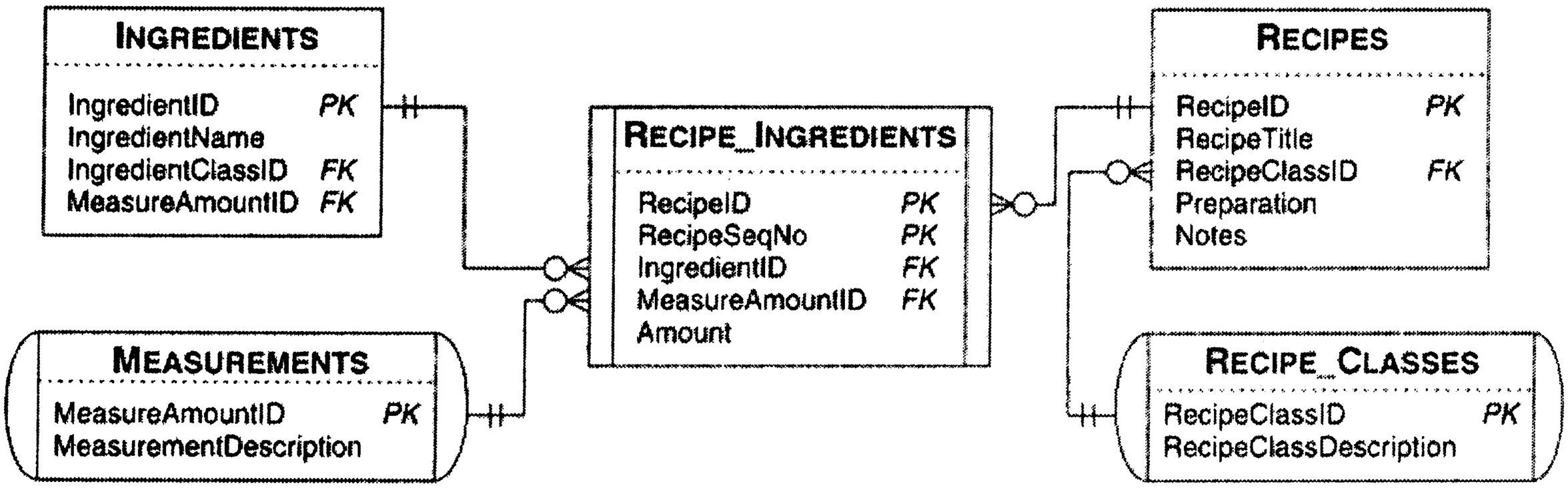


Рис. 9.7. *Таблицы из примера базы данных Recipes, необходимые для извлечения всей информации о рецептах*

еще не определено ни одного рецепта, и все детали рецептов и их компонентов. На рис. 9.7 представлены все таблицы, необходимые для ответа на данный запрос.

Похоже, что нужны данные из *пяти* различных таблиц. Их можно получить путем построения более сложного условия FROM, вкладывая условия JOIN в условия JOIN. Фокус вот в чем: везде, где можно определить имя таблицы, можно также определить взятое в целом условие JOIN, заключенное в скобки. На рис. 9.8 представлен упрощенный вариант соединения двух таблиц (для образования простого INNER или OUTER JOIN двух таблиц выбрано условие ON, а также убраны условия с корреляционными именами).

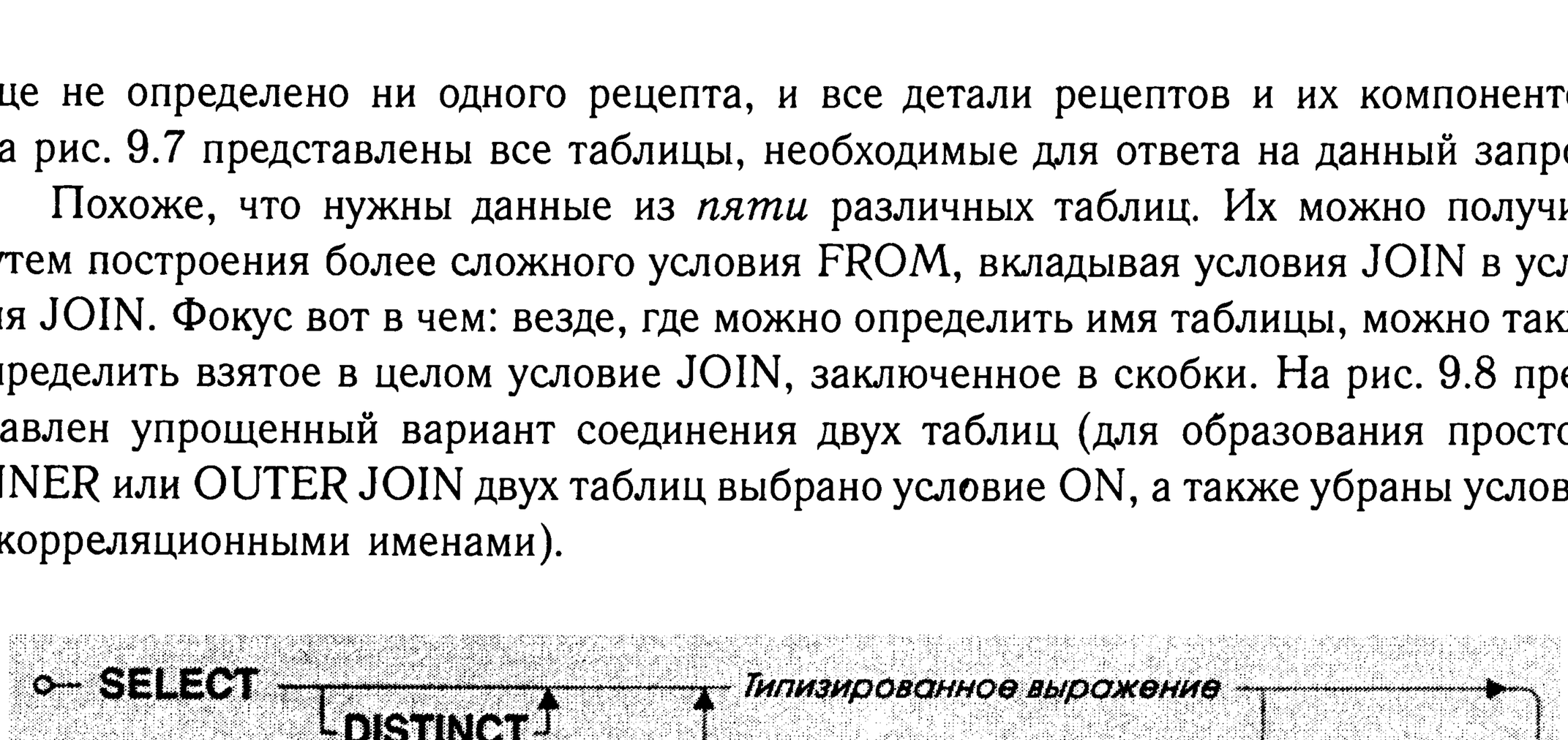


Рис. 9.8. *Простая операция JOIN двух таблиц*

Для того чтобы добавить третью таблицу к этому соединению, просто поставьте открывающую скобку перед именем первой таблицы, добавьте закрывающую скобку после условия поиска, а затем добавьте еще одно JOIN, имя таблицы, ключевое слово ON и еще одно условие поиска. На рис. 9.9 показано, как это сделать.

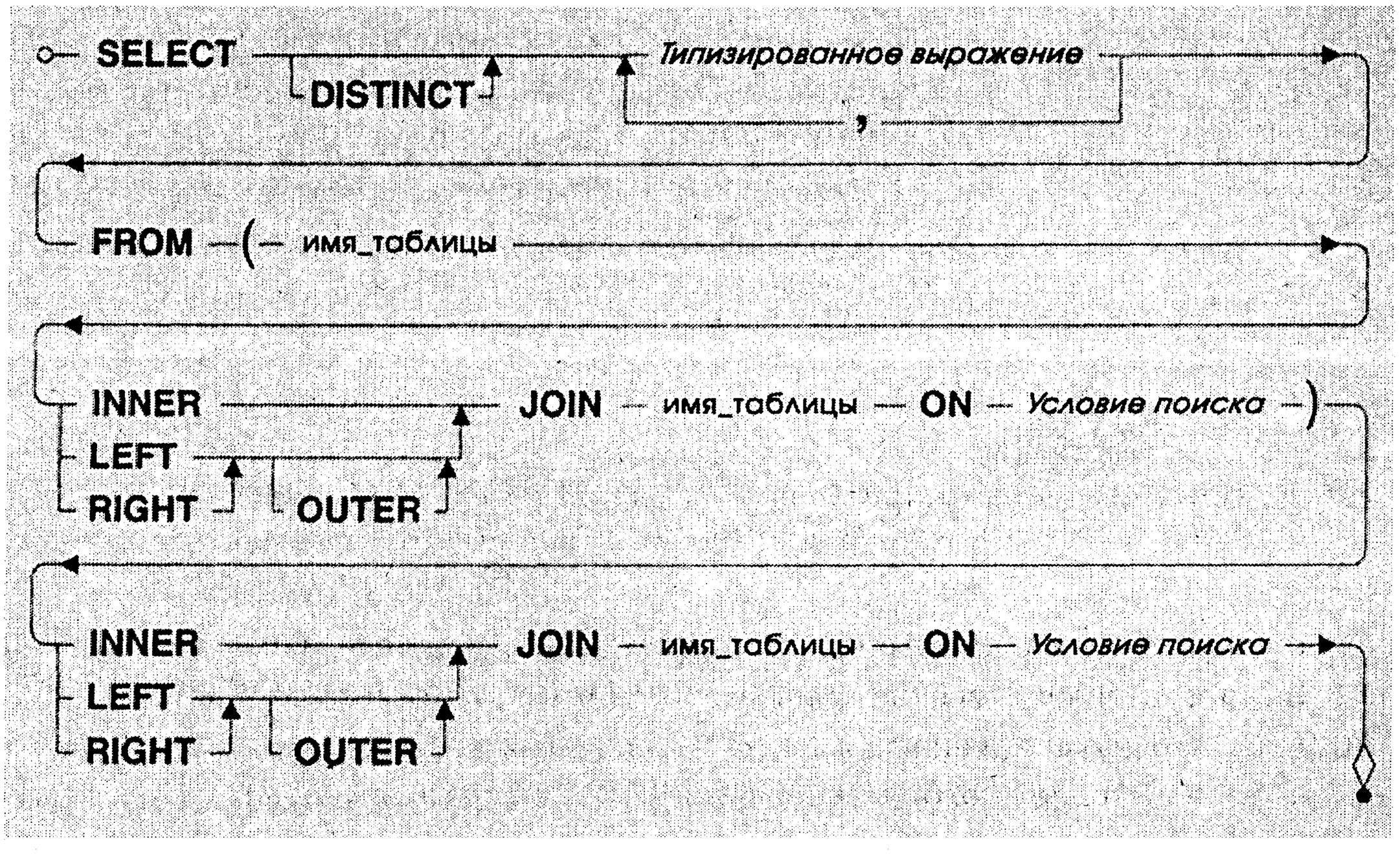


Рис. 9.9. *Простая операция JOIN трех таблиц*

JOIN двух таблиц, заключенное в скобки, образует "логическую" таблицу, или внутренний набор результатов. Этот набор теперь помещается вместо имени первой простой таблицы на рис. 9.5, и данный процесс можно продолжить, заключая в скобки полное условие JOIN, добавляя затем еще одно ключевое слово JOIN, имя таблицы, ключевое слово ON и условие поиска, пока не будут получены все необходимые наборы результатов. Создадим запрос, для которого требуются данные из всех таблиц, представленных на рис. 9.7. (этот тип запроса можно использовать для создания отчета, в котором перечисляются все виды рецептов с деталями рецептов в каждом из видов):

> *"I need all of the recipe types, and then the matching recipe name, preparation instructions, ingredient names, ingredient step number, ingredient quantities, and ingredient measurements from my Recipes database, sorted in step number sequence".*
>
> *("Нужны все виды рецептов, относящиеся к ним названия рецептов, указания по приготовлению, названия компонентов, номер этапа добавления компонента, количество компонента и единицы измерения компонента из базы данных "Рецепты", отсортированные в последовательности номеров этапов".*

Преобразование: Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table left outer joined with the recipes table on recipe class ID, then joined with the recipe ingredients

table on recipe ID, then joined with the ingredients table on ingredient ID, and then finally joined with the measurements table on measurement amount ID, order by recipe title and recipe sequence number
(Выбрать описание вида рецепта, заголовок рецепта, указания по приготовлению, название компонента, порядковый номер рецепта, количество и описание единиц измерения из таблицы “Виды рецептов” с левым внешним соединением с таблицей “Рецепты” по идентификатору вида рецептов, соединенной с таблицей “Компоненты рецепта” по идентификатору рецепта, соединенной с таблицей “Компоненты” по идентификатору компонента и, наконец, соединенной с таблицей единиц измерения по идентификатору единиц измерения, упорядоченные по заголовку рецепта и порядковому номеру рецепта)

Уточнение:

Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description from ~~the~~ recipe classes ~~table~~ left outer join~~ed with the~~ recipes ~~table~~ on recipe class ID, ~~then~~ join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID, ~~then~~ join~~ed with the~~ ingredients ~~table~~ on ingredient ID, ~~and then finally~~ join~~ed with the~~ measurements ~~table~~ on measurement amount ID, order by recipe title ~~and~~ recipe sequence number
(Выбрать описание вида рецепта, заголовок рецепта, приготовление, название компонента, порядковый номер рецепта, количество, описание единиц измерения из “Виды рецептов” с левым внешним соединением с “Рецепты” по идентификатору вида рецептов, соединенной с “Компоненты рецепта” по идентификатору рецепта, соединенной с “Компоненты” по идентификатору компонента, соединенной с “Единицы измерения” по идентификатору единиц измерения, упорядоченные по заголовку рецепта, порядковому номеру рецепта)

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
    Recipes.RecipeTitle,Recipes.Preparation,
    Ingredients.IngredientName,
    Recipe_Ingredients.RecipeSeqNo,
    Recipe_Ingredients.Amount,
    Measurements.MeasurementDescription
```

```
FROM (((Recipe_Classes
LEFT OUTER JOIN Recipes
ON  Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID)
INNER JOIN Recipe_Ingredients
ON  Recipes.RecipeID =
    Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON  Measurements.MeasureAmountID =
    Recipe_Ingredients.MeasureAmountID
ORDER BY RecipeTitle, RecipeSeqNo
```

Действительно, можно заменить взятое в целом JOIN двух таблиц в любом месте, где можно в ином случае поместить просто имя таблицы. На рис. 9.9 предполагается, что необходимо вначале соединить первую таблицу со второй, а затем этот результат соединить с третьей таблицей. Также можно вначале соединить вторую и третью таблицы (в том случае, если третья таблица фактически имеет отношение ко второй таблице, но не к первой), а затем выполнить окончательное соединение с первой таблицей. На рис. 9.10 представлен этот альтернативный метод.

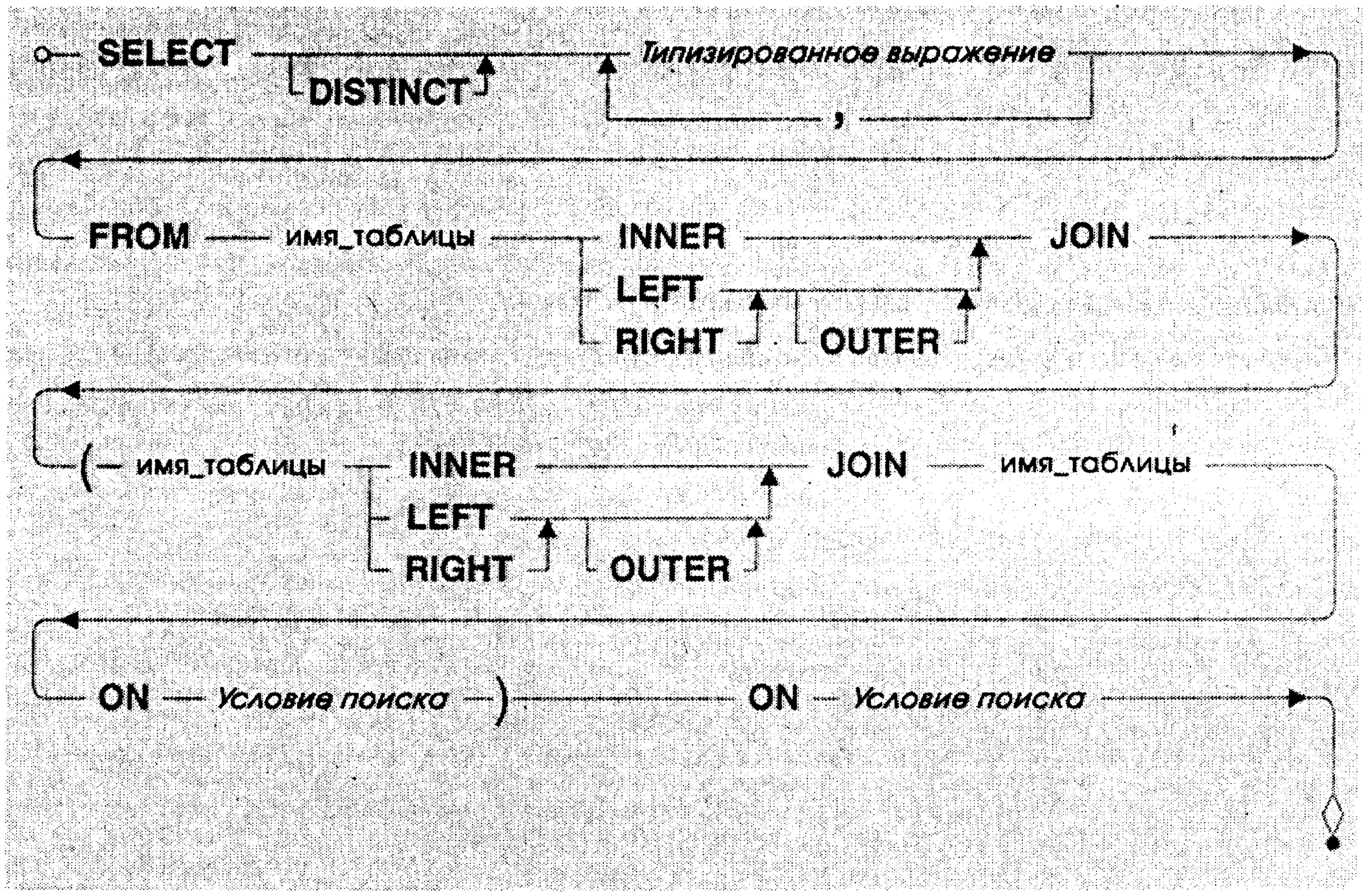


Рис. 9.10. *Соединение более чем двух таблиц в альтернативной последовательности*

Для того чтобы составить только что представленный запрос с использованием пяти таблиц, можно также записать следующий оператор SQL:

```
SQL                SELECT Recipe_Classes.RecipeClassDescnption,
                       Recipes.RecipeTitle,  Recipes.Preparation,
                       Ingredients.IngredientName,
                       Recipe_Ingredients.RecipeSeqNo,
                       Recipe_Ingredients.Amount,
                       Measurements.MeasurementDescription
                   FROM Recipe_Classes
                   LEFT OUTER JOIN
                   (((Recipes INNER JOIN Recipe_Ingredients
                   ON  Recipes.RecipeID = Recipe_Ingredients.RecipeID)
                   INNER JOIN Ingredients
                   ON  Ingredients.IngredientID =
                       Recipe_Ingredients.IngredientID)
                   INNER JOIN Measurements
                   ON  Measurements.MeasureAmountID =
                       Recipe_Ingredients.MeasureAmountID)
                   ON  Recipe_Classes.RecipeClassID =
                       Recipes.RecipeClassID
                   ORDER BY RecipeTitle,RecipeSeqNo
```

Помните, что оптимизаторы некоторых баз данных чувствительны к последовательности определения условий JOIN. Если выполнение запроса с использованием многих операций JOIN занимает много времени для больших баз данных, можно заставить его выполняться быстрее, изменив последовательность операций JOIN в операторе SQL.

В предыдущих примерах с несколькими соединениями использовалось только одно условие OUTER JOIN. Посмотрим, возможно ли и имеет ли смысл использование более одного OUTER JOIN в сложных JOIN. Предположим, что имеются не только некоторые Recipe_Classes, для которых отсутствуют сопоставленные им строки в Recipes, но также есть рецепты, для которых пока еще не определены компоненты. В предыдущем примере не было ни одной строки из таблицы Recipes, для которой не было бы сопоставления в таблице Recipe_Ingredients, потому что INNER JOIN их исключает. Запросим также все рецепты:

> *"I need all of the recipe types, and then all of the recipe names, preparations, and instructions, and then any matching ingredient names, ingredient step number, ingredient quantities, and ingredient measurements from my Recipes database, sorted in step number sequence".*
> *("Нужны все виды рецептов, все названия рецептов, приготовления и указания и затем все сопоставленные им названия компонентов, номер этапа компонента, количество компонента и единицы измерения компонента из базы данных "Рецепты", отсортированные в последовательности этапов".)*

Преобразование: Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table left outer joined with the recipes table on recipe class ID, then left outer joined with the recipe ingredients table on recipe ID, then joined with the ingredients table on ingredient ID, and then finally joined with the measurements table on measurement amount ID, order by recipe title and recipe sequence number
(Выбрать описание вида рецепта, заголовок рецепта, указания по приготовлению, название компонента, порядковый номер рецепта, количество и описание единиц измерения из таблицы “Виды рецепта” с левым внешним соединением с таблицей “Рецепты” по идентификатору рецепта, затем с левым внешним соединением с таблицей “Компоненты рецепта” по идентификатору рецепта, соединенной с таблицей “Компоненты” по идентификатору компоненты, и, наконец, соединенной с таблицей “Единицы измерения” по идентификатору единиц измерения количества, упорядоченные по заголовку рецепта и порядковому номеру рецепта)

Уточнение: Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description from ~~the~~ recipe classes ~~table~~ left outer join~~ed with the~~ recipes ~~table~~ on recipe class ID, ~~then~~ left outer join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID, ~~then~~ join~~ed with the~~ ingredients ~~table~~ on ingredient ID, ~~and then finally~~ join~~ed with the~~ measurements ~~table~~ on measurement amount ID, order by recipe title ~~and~~ recipe sequence number
(Выбрать описание вида рецепта, заголовок рецепта, приготовление, название компонента, порядковый номер рецепта, количество, описание единиц измерения из “Виды рецепта” с левым внешним соединением с “Рецепты” по идентификатору рецепта, с левым внешним соединением с “Компоненты рецепта” по идентификатору рецепта, соединенной с “Компоненты” по идентификатору компонента, соединенной с “Единицы измерения” по идентификатору единиц измерения количества, упорядоченные по заголовку рецепта, порядковому номеру рецепта)

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
    Recipes.RecipeTitle.Recipes.Preparation,
    Ingredients.IngredientName,
    Recipe_Ingredients.RecipeSeqNo,
    Recipe_Ingredients.Amount,
    Measurements.MeasurementDescription
FROM (((Recipe_Classes
LEFT OUTER JOIN Recipes
ON  Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID)
LEFT OUTER JOIN Recipe_Ingredients
ON  Recipes.RecipeID =
    Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID)
INNER JOIN Measurements
ON  Measurements.MeasureAmountID =
    Recipe_Ingredients.MeasureAmountID
ORDER BY RecipeTitle, RecipeSeqNo
```

Будьте осторожны! Эта разновидность многократного OUTER JOIN работает, только если соблюдается путь для отношения один-ко-многим. Рассмотрим отношения между Recipe_Classes, Recipes и Recipe_Ingredients, показанные на рис. 9.11.

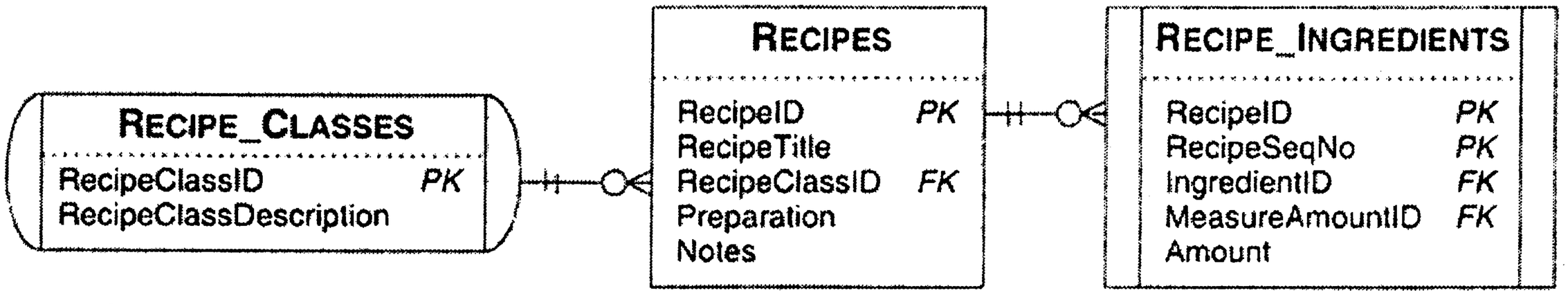


Рис. 9.11. *Отношения между таблицами Recipe_Classes, Recipes и Recipe_Ingredients*

Здесь представлено отношение один-ко-многим, иногда называемое отношением “родитель-потомок”. Каждая строка “родитель” (с одной стороны отношения) может иметь несколько потомков или не иметь их вовсе (с другой стороны отношения). Если только нет “зависших” строк (“сирот”) со стороны “многие” (например, строка в таблице “Рецепты” со значением Null в своем столбце RecipeClassID), то *каждая* строка в таблице “потомок” должна иметь сопоставленную ей строку в “родительской” таблице. Поэтому имеет смысл использовать Recipe_Classes LEFT JOIN Recipes для выбора всех “родительских” строк в Recipe_Classes, у которых пока еще отсутствует “потомок” в Recipes. Recipe_Classes RIGHT JOIN Recipes даст (за исключением всех “зависших” строк) тот же результат, что и INNER JOIN.

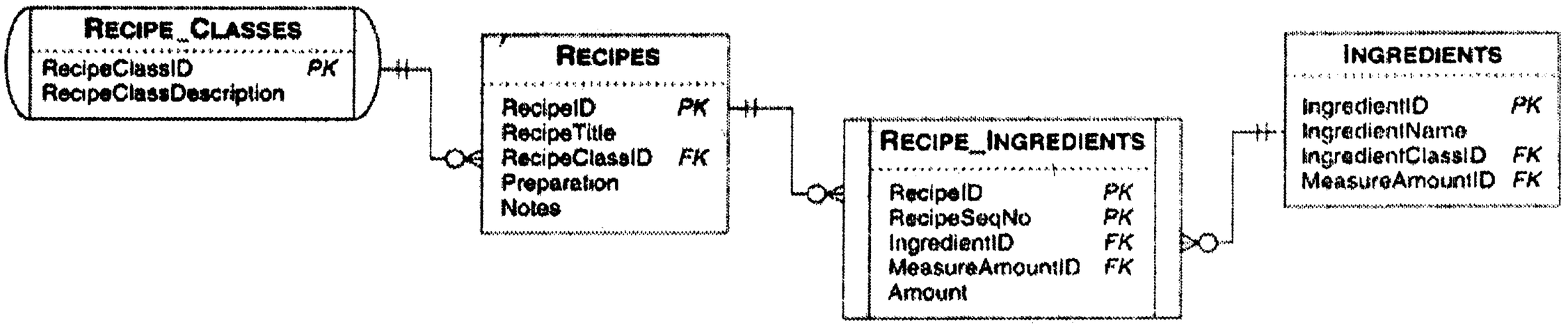


Рис. 9.12. *Отношение между таблицами Recipe_Classes, Recipes, Recipe_Ingredients и Ingredients*

Подобным образом имеет смысл запросить Recipes LEFT JOIN Recipe_Ingredients, потому что могут быть рецепты, для которых пока еще не указаны компоненты. Recipes RIGHT JOIN Recipe_Ingredients не работает, потому что связывающий столбец (RecipeID) в Recipe_Ingredients также является частью первичного ключа этой таблицы; поэтому гарантируется, что в Recipe_Ingredients отсутствуют "зависшие" строки, поскольку ни один столбец в первичном ключе не может содержать значение Null.

Продвинемся еще на шаг вперед и запросим все компоненты, включая те, которые еще не введены ни в один рецепт. Вначале посмотрим внимательно на отношения между таблицами, включая таблицу Ingredients, показанные на рис. 9.12.

Попробуем записать следующий запрос (осторожно: здесь есть ловушка!):

"I need all of the recipe types, and then all of the recipe names, preparations, and instructions, and then any matching ingredient step number, ingredient quantities, and ingredient measurements and finally all ingredient names from my Recipes database sorted in step number sequence".
("Нужны все виды рецептов, все названия рецептов, приготовления и указания, и затем все сопоставленные им номера этапа компонента, количество компонента и единица измерения компонента и, наконец, названия всех компонентов из базы данных "Рецепты", отсортированные в последовательности этапов".)

Преобразование: Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table left outer joined with the recipes table on recipe class ID, then left outer joined with the recipe ingredients table on recipe ID, then joined with the measurements table on measurement amount ID, and then finally right outer joined with the ingredients table on ingredient ID, order by recipe title and recipe sequence number
(Выбрать описание вида рецепта, заголовок рецепта, указания по приготовлению, название компонента, порядковый номер рецепта, количество и описание

единиц измерения из таблицы “Виды рецепта”
с левым внешним соединением с таблицей “Рецепты”
по идентификатору вида рецепта, затем с левым внешним
соединением с таблицей “Компоненты рецепта”
по идентификатору рецепта, соединенной с таблицей
“Единицы измерения” по идентификатору единиц
измерения количества, и, наконец, соединенной с правым
внешним соединением с таблицей “Компоненты”
по идентификатору компонента, упорядоченные
по заголовку рецепта и порядковому номеру рецепта)

Уточнение:

Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description from ~~the~~ recipe classes ~~table~~ left outer join~~ed with the~~ recipes ~~table~~ on recipe class ID, ~~then~~ left outer join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID, ~~then~~ join~~ed with the~~ measurements ~~table~~ on measurement amount ID, ~~and then finally~~ right outer join~~ed with the~~ ingredients ~~table~~ on ingredient ID, order by recipe title ~~and~~ recipe sequence number

(Выбрать описание вида рецепта, заголовок рецепта,
приготовление, название компонента, порядковый номер
рецепта, количество, описание единиц измерения
из “Виды рецепта” с левым внешним соединением
с “Рецепты” по идентификатору вида рецепта, затем
с левым внешним соединением с “Компоненты рецепта”
по идентификатору рецепта, соединенной
с “Единицы измерения” по идентификатору единиц
измерения количества, соединенной правым внешним
соединением с “Компоненты” по идентификатору
компонента, упорядоченные по заголовку рецепта,
порядковому номеру рецепта)

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
    Recipes.RecipeTitle, Recipes.Preparation,
    Ingredients.IngredientName,
    Recipe_Ingredients.RecipeSeqNo,
    Recipe_Ingredients.Amount,
    Measurements.MeasurementDescription
FROM (((Recipe_Classes
LEFT OUTER JOIN Recipes
ON Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID)
```

```
LEFT OUTER JOIN Recipe_Ingredients
ON  Recipes.RecipeID =
    Recipe_Ingredients.RecipeID)
INNER JOIN Measurements
ON  Measurements.MeasureAmountID =
    Recipe_Ingredients.MeasureAmountID)
RIGHT OUTER JOIN Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID
ORDER BY RecipeTitle, RecipeSeqNo
```

Думаете, это будет работать? Ответ ошеломляющий: НЕТ! Большинство СУБД выполняет анализ всего условия FROM целиком, а затем пытается определить наиболее эффективный способ сбора связей таблиц. Предположим, однако, что база данных решит полностью сохранить наш порядок группирования JOIN в скобках. Это означает, что СУБД будет выполнять операции, начиная от самого внутреннего JOIN (Recipe_Classes, соединенная с Recipes), а затем займется внешними операциями.

Поскольку некоторые строки в таблице Recipe_Classes могут совсем не иметь сопоставимых строк в Recipes, это первое JOIN возвратит строки, которые содержат значение Null в RecipeID. Вернувшись к рис. 9.12 можно увидеть, что между Recipe_Classes и Recipes существует отношение один-ко-многим. Если для каких-либо рецептов не был назначен Recipe_Class, мы все равно получим строки из таблицы Recipes! Следующий JOIN с таблицей Recipe_Ingredients также запрашивает LEFT OUTER JOIN. Нам требуются все строки независимо от наличия каких-либо значений Null из предыдущего JOIN (Recipe Classes с Recipes) и всех сопоставленных им строк в Recipe_Ingredients. Опять-таки, поскольку некоторые строки в Recipe Classes могут не иметь сопоставленных им строк в Recipes или у каких-либо строк в Recipes могут отсутствовать Recipe_Ingredients, некоторые строки могут иметь значение Null в столбце IngredientID из таблицы Recipe_ Ingredients. Все, что делается с обоими условиями JOIN, это просто “перемещение” по отношениям один-ко-многим от Recipe_Classes к Recipe и затем от Recipes к Recipe_Ingredients. До сих пор все хорошо. (Кстати, окончательное INNER JOIN с Measurements несущественно — мы знаем, что все Ingredients имеют допустимый MeasureAmountID.)

А вот сейчас начинаются затруднения. Завершающее RIGHT OUTER JOIN запрашивает все строки из Ingredients и *любые строки, удовлетворяющие условию,* из результата предыдущих условий JOIN. Вспомните, что Null является очень специфическим значением; оно не может быть равным ни любому другому значению, ни даже другому значению Null. Если запрашиваются *все* строки из Ingredients, то для всех этих строк IngredientID имеет не-Null значение. Ни одна из строк из предыдущего JOIN, IngredientID которых имеет значение Null, не будет сопоставлена вообще, поэтому завершающий JOIN совсем отбрасывает их! Будут

представлены все компоненты, которые пока еще даже не используются ни в одном рецепте, но будут отсутствовать виды рецептов, для которых нет рецептов, или рецепты, которые не содержат компонентов.

Если СУБД решает выполнить запрос путем исполнения условий JOIN другим образом, то будут присутствовать виды рецептов, не имеющие рецептов, и рецепты, не имеющие компонентов, но будут отсутствовать компоненты, не используемые пока еще ни в одном из рецептов, вследствие проблем, связанных с сопоставлением с Null. Некоторые системы баз данных могут распознать эту логическую проблему и отклонить выполнение этого запроса совсем. При этом будет получено сообщение об ошибке типа "неоднозначное OUTER JOIN". Проблема, с которой мы сталкиваемся в данном случае, является результатом попытки "перемещения в противоположном направлении" по отношению один-ко-многим в OUTER JOIN, имеющему другое направление. Спускаться по склону легко, но движение в обратном направлении требует специальных приспособлений. Как решить эту проблему, мы расскажем в следующем разделе.

FULL OUTER JOIN

FULL OUTER JOIN не является ни "левым", ни "правым" — оно является двусторонним! В него включаются *все* строки из обеих таблиц или наборов результатов, используемых в JOIN. Когда отсутствуют строки, выполняющие условие, с левой стороны JOIN, то в наборе результата "справа" будут присутствовать значения Null. Наоборот, когда отсутствуют строки, выполняющие условие, с правой стороны JOIN, то значения Null будут присутствовать в наборе результата "слева".

Синтаксис

Теперь, когда вы немного поработали с операциями JOIN, синтаксис для FULL OUTER JOIN вполне понятен. Синтаксическая диаграмма для FULL OUTER JOIN показана на рис. 9.13.

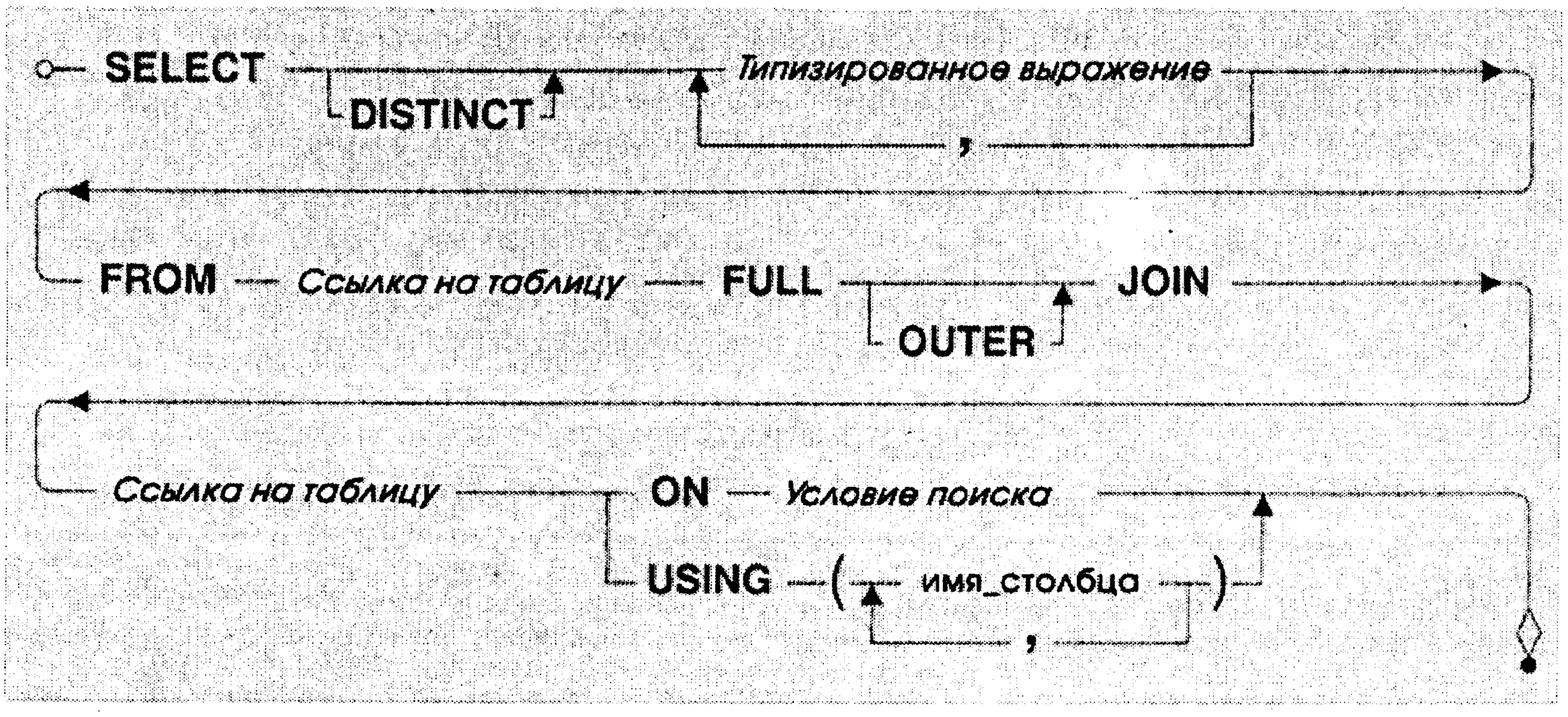


Рис. 9.13. *FULL OUTER JOIN*

Для упрощения вместо имени таблицы, оператора SELECT или результата другого JOIN теперь используется термин *"ссылка на таблицу"*. Посмотрим на задачу из предыдущей главы с другой стороны. Теперь можно решить ее надлежащим образом, используя FULL OUTER JOIN:

> *"I need all of the recipe types and then all of the recipe names, preparations, and instructions, and then any matching ingredient step numbers, ingredient quantities, and ingredient measurements, and finally all ingredient names from my Recipes database sorted in step number sequence".*
> *("Нужны все виды рецептов, все названия рецептов, последовательность приготовления и указания, инструкции, а затем номера последовательности этапов для каждого выполняющего условие компонента, количество компонента и единицы измерения компонента и, наконец, названия всех компонентов из базы данных "Рецепты", отсортированные по номерам этапов".)*

Преобразование: Select the recipe class description, recipe title, preparation instructions, ingredient name, recipe sequence number, amount, and measurement description from the recipe classes table full outer joined with the recipes table on recipe class ID, then left outer joined with the recipe ingredients table on recipe ID, then joined with the measurements table on measurement amount ID, and then finally full outer joined with the ingredients table on ingredient ID, order by recipe title and recipe sequence number
(Выбрать описание вида рецепта, заголовок рецепта, инструкции по приготовлению, название компонента, порядковый номер рецепта, количество и описание единиц измерения из таблицы "Виды рецептов" с полным внешним соединением с таблицей "Рецепты" по идентификатору вида рецепта, затем с левым внешним соединением с таблицей "Компоненты рецепта" по идентификатору рецепта, соединенной с таблицей "Единицы измерения" по идентификатору единиц измерения количества, а затем, наконец, с полным внешним соединением с таблицей "Компоненты" по идентификатору компонента, упорядоченные по заголовку рецепта и порядковому номеру рецепта)

Уточнение:

Select ~~the~~ recipe class description, recipe title, preparation ~~instructions~~, ingredient name, recipe sequence number, amount, ~~and~~ measurement description from ~~the~~ recipe classes ~~table~~ full outer join~~ed with the~~ recipes ~~table~~ on recipe class ID, ~~then~~ left outer join~~ed with the~~ recipe ingredients ~~table~~ on recipeID, ~~then~~ join~~ed with the~~ measurements ~~table~~ on measurement amount ID, ~~and then finally~~ full outer join~~ed with the~~ ingredients ~~table~~ on ingredient ID, order by recipe title ~~and~~ recipe sequence number

(Выбрать описание вида рецепта, заголовок рецепта, приготовление, название компонента, порядковый номер рецепта, количество, описание единиц измерения из “Виды рецептов” с полным внешним соединением с “Рецепты” по идентификатору вида рецепта, с левым внешним соединением с “Компоненты рецепта” по идентификатору рецепта, соединенной с “Единицы измерения” по идентификатору единиц измерения количества, с полным внешним соединением с “Компоненты” по идентификатору компонента, упорядоченные по заголовку рецепта, порядковому номеру рецепта)

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
    Recipes.RecipeTitle, Recipes.Preparation,
    Ingredients.IngredientName,
    Recipe_Ingredients.RecipeSeqNo,
    Recipe_Ingredients.Amount,
    Measurements.MeasurementDescription
FROM Recipe_Classes
FULL OUTER JOIN (((Recipes
LEFT OUTER JOIN
    Recipe_Ingredients
ON Recipes.RecipeID =
    Recipe_Ingredients.RecipeID)
INNER JOIN Measurements
ON Measurements. MeasureAmountID =
    Recipe_Ingredients.MeasureAmountID)
FULL OUTER JOIN Ingredients
ON Ingredients.IngredientID =
    RecipeIngredients.IngredientID)
ON Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID
ORDER BY RecipeTitle, RecipeSeqNo
```

Первое и последнее JOIN запрашивают *все* строки с обеих сторон JOIN, поэтому проблема с невозможностью сопоставления значений Null решена. Теперь можно увидеть не только виды рецептов, для которых отсутствуют рецепты, и рецепты, для которых отсутствуют компоненты, но также компоненты, которые пока еще не использовались в рецептах. Можно отказаться от использования LEFT OUTER JOIN для первого JOIN, но, поскольку невозможно предсказать предварительно, как СУБД решает вложение JOIN, следует запросить FULL OUTER JOIN с обеих сторон, чтобы гарантировать получение правильного ответа.

Внимание! СУБД, которая не поддерживает синтаксис стандарта SQL для LEFT OUTER JOIN или RIGHT OUTER JOIN, также имеет специальный синтаксис для FULL OUTER JOIN. Необходимо уточнить в документации по своей базе данных специальный нестандартный синтаксис, который требуется базе данных для определения OUTER JOIN. Например, ранние версии Microsoft SQL Server поддерживают этот синтаксис (обратите внимание на звездочки в условии WHERE):

```
SELECT  Recipe_Classes.RecipeClassDescription,
      Recipes.RecipeTitle
FROM  Recipe_Classes,  Recipes
WHERE  Recipe_Classes.RecipeClassID  *=*
      Recipes.RecipeClassID
```

Продукты, которые не поддерживают любой синтаксис FULL OUTER JOIN, но поддерживают LEFT или RIGHT OUTER JOIN, дают эквивалентный результат, выполняя UNION по LEFT и RIGHT OUTER JOIN (подробнее о UNION см. в следующей главе). Поскольку специальный синтаксис для определения FULL OUTER JOIN через условие WHERE различается в разных продуктах, то при работе с несколькими нестандартными продуктами необходимо изучить различные варианты синтаксиса.

FULL OUTER JOIN не по значениям ключей

До сих пор мы обсуждали использование OUTER JOIN для связывания таблиц или наборов результатов по соответствующим значениям ключей. Однако можно решить некоторые интересные задачи, используя OUTER JOIN не по значениям ключей. Например, можно получить список *всех* штатных сотрудников и *всех* студентов в учебной базе данных расписания занятий, а также показать, у кого из них совпадают имена. Это делается с помощью FULL OUTER JOIN:

"Show me all of the students and all of the teachers
and list together those who have the same first name".
("Показать всех студентов и всех преподавателей,
а также список тех, у кого совпадают имена".)

Преобразование: Select student full name and staff full name from the students table full outer joined with the staff table on first name
(Выбрать полное имя студента и полное имя преподавателя из таблицы “Студенты” с полным внешним соединением в таблицей “Персонал” по имени)

Уточнение: Select student full name ~~and~~ staff full name from ~~the~~ students ~~table~~ full outer join~~ed with the~~ staff ~~table~~ on first name
(Выбрать полное имя студента, полное имя преподавателя из “Студенты” с полным внешним соединением с “Персонал” по имени)

SQL

```
SELECT (Students.StudFirstName, || ' ' ||
    Students.StudLastName) AS StudFullName,
    (Staff.StfFirstName || ' ' ||
    Staff.StfLastName) AS StfFullName
FROM Students
FULL OUTER JOIN Staff
ON Students.StudFirstName = Staff.StfFirstName
UNION JOIN
```

Любое обсуждение OUTER JOIN будет неполным без уважительного упоминания UNION JOIN. В стандарте SQL операция UNION JOIN является FULL OUTER JOIN с исключенными совпадающими строками. На рис. 9.14 представлен синтаксис.

Не так много коммерческих реализаций поддерживают UNION JOIN. Если говорить откровенно, то следует придумать хорошую причину, по которой стоило бы применить UNION JOIN.

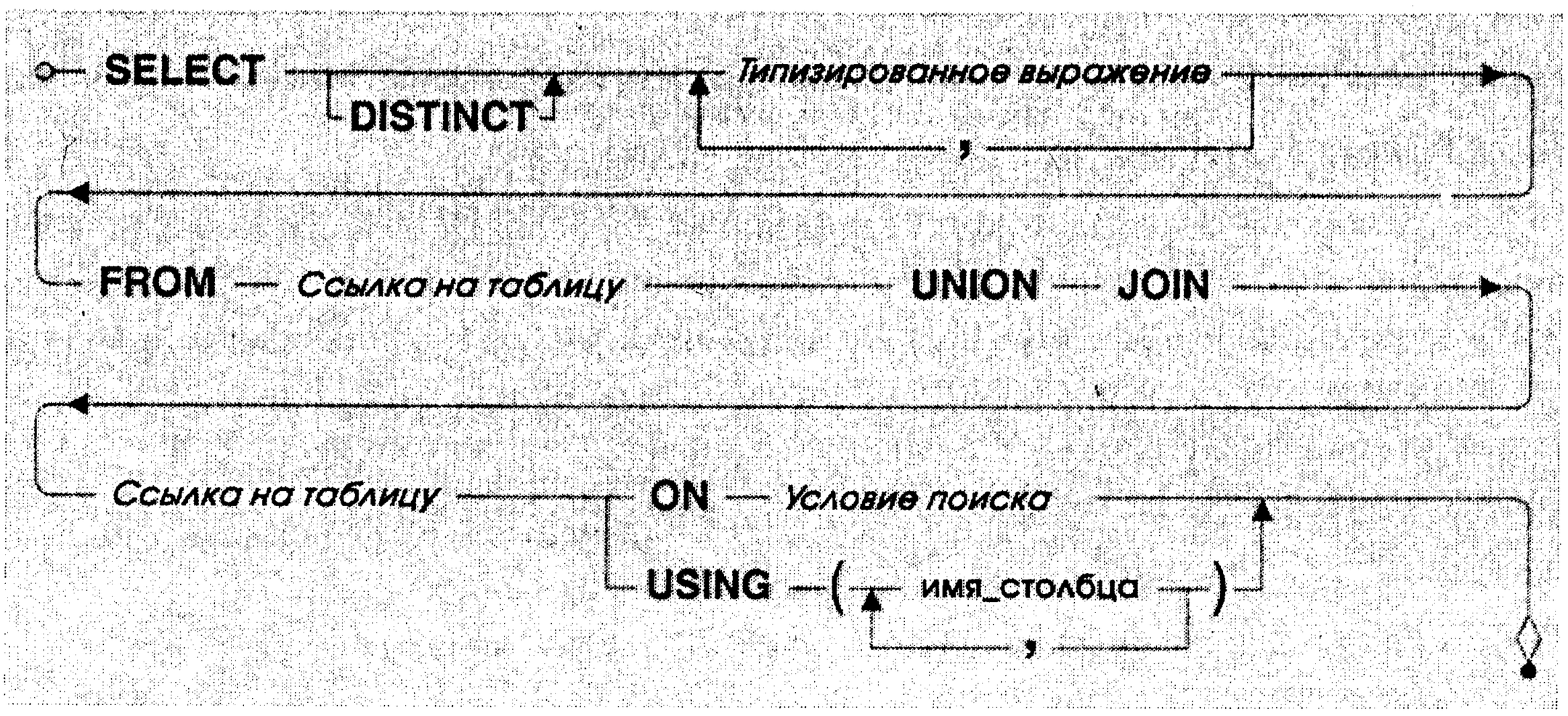


Рис. 9.14. *Синтаксис SQL для UNION JOIN*

Использование операций OUTER JOIN

Поскольку операция OUTER JOIN позволяет получить не только строки, удовлетворяющие условию, но также и не удовлетворяющие ему, то она прекрасно подходит для поиска тех строк в одной таблице, которые не имеют соответствующих строк, выполняющих условия, в другой таблице. Она также помогает найти строки, для которых есть совпадения только в нескольких строках, но не во всех. К тому же она полезна для создания входных данных в отчете, где нужно показать "все" категории (независимо от существования строк, выполняющих условия, в другой таблице) или "всех" клиентов (независимо от того, разместил ли клиент заказ). Ниже приведен пример видов запросов, которые можно решить с помощью OUTER JOIN.

Поиск пропущенных значений

Иногда нужно найти, что же "пропущено". Это можно сделать, воспользовавшись OUTER JOIN с проверкой на Null. Вот несколько задач на "пропущенные значения":

"Какие товары никогда не заказывались?"

"Показать клиентов, которые никогда не заказывали шлем".

"Привести список эстрадных артистов, на которых никогда не было заявок".

"Вывести на дисплей агентов, которые не регистрировали заявок на эстрадного артиста".

"Показать турниры, которые еще не разыгрывались".

"Привести список преподавателей, которые ничего не преподают".

"Вывести на дисплей имена студентов, которые никогда не отказывались от курса лекций".

"Показать курсы лекций, на которые не записалось ни одного студента".

"Привести список компонентов, которые пока еще не используются в рецептах".

"Вывести на дисплей виды рецептов".

Поиск частично совпадающей информации

В частности, для отчетов полезно иметь возможность представить список всех строк из одной или нескольких таблиц вместе со строками, удовлетворяющими условию, из связанных таблиц. Вот пример задач на "частичное совпадение", которые можно решить с использованием OUTER JOIN:

"Привести список всех товаров и даты всех заказов".

"Вывести на дисплей всех клиентов и все заказы на велосипеды".

"Показать все направления эстрадных концертов и клиентов, предпочитающих эти направления".

"Привести список всех эстрадных артистов и все ангажементы, в которых они зарегистрированы".

"Привести список всех игроков в боулинг и все игры, в которых они сбили больше 160".

"Вывести на дисплей все турниры и все сыгранные матчи".

"Показать все категории предметов и все курсы лекций для всех предметов".

"Привести список всех студентов и курсы лекций, на которые они в настоящее время записались".

"Вывести на дисплей всех преподавателей и курсы лекций, которые они планируют читать".

"Привести список всех типов рецептов, все рецепты и все компоненты, используемые в них".

"Показать все компоненты и все рецепты, в которых они используются".

Примеры операторов

Теперь вам известна механика построения запросов с использованием OUTER JOIN и вы видели некоторые типы запросов, ответы на которые можно найти с OUTER JOIN. Давайте ознакомимся с довольно устойчивым множеством примеров, в которых используется OUTER JOIN. Они взяты из учебных баз данных и иллюстрируют использование OUTER JOIN для поиска либо пропущенных, либо частично совпадающих значений.

Внимание! Поскольку многие их этих примеров используют сложные соединения, СУБД может выбрать другой способ решения. По этой причине несколько первых строк, которые показаны здесь, могут не совпадать точно с результатом, полученным вами, но общее количество строк должно быть то же самое. Для упрощения процесса этапы перевода и уточнения для всех последующих примеров объединены.

База данных заказов на закупку

"What products have never been ordered?"
("Какие товары никогда не заказывались?")

Преобразование/ Уточнение:

Select product number ~~and~~ product name from ~~the~~ products ~~table~~ left outer join~~ed with the~~ order details ~~table~~ on product ID where ~~the~~ order detail´order number is null

(Выбрать номер товара, наименование товара из “Товары” с левым внешним соединением с “Детали заказа” по идентификатору товара, где в деталях заказа номер заказа Null)

SQL

```
SELECT Products.ProductNumber,
    Products.ProductName
FROM Products LEFT JOIN Order_Details
ON  Products.ProductNumber =
    Order_Details.ProductNumber
WHERE Order_Details.OrderNumber IS NULL
```

Products_Never_Ordered (2 строки)

ProductNumber	ProductName
4	Victoria Pro All Weather Tires
23	Ultra-Pro Rain Jacket

“Display all customers and any orders for bicycles”.
(“Вывести на дисплей всех клиентов и все заказы на велосипеды”.)

Преобразование 1: Select customer full name, order date, product name, quantity ordered, and quoted price from the customers table left outer joined with the orders table on customer ID, then joined with the order details table on order number, then joined with the products table on product number, then finally joined with the categories table on category ID where category description is ‘Bikes’
(Выбрать полное имя клиента, дату заказа, наименование товара, заказанное количество и назначенную цену из таблицы “Клиенты” с левым внешним соединением с таблицей “Заказы” по идентификатору клиента, соединенной с таблицей “Детали заказа” по номеру заказа, соединенной с таблицей “Товары” по номеру товара, затем, наконец, соединенной с таблицей “Категории” по идентификатору категорий, где описание категории — ‘велосипед’)

Преобразование 2/ Уточнение: Select customer full name, order date, product name, quantity ordered, ~~and~~ quoted price from ~~the~~ customers ~~table~~ left outer join~~ed with~~ (Select customer ID, order date, product name, quantity ordered, ~~and~~ quoted price from ~~the~~ orders ~~table~~ join~~ed with the~~

order details ~~table~~ on order number, ~~then~~ join~~ed with the~~ products ~~table~~ on product number, ~~then finally~~ join~~ed with the~~ categories ~~table~~ on category ID where category description ~~is~~ = 'Bikes') on customer ID
(Выбрать полное имя клиента, дату заказа, наименование товара, заказанное количество, назначенную цену из "Клиенты" с левым внешним соединением с (Выбрать идентификатор клиента, дату заказа, наименование товара, заказанное количество, назначенную цену из "Заказы", соединенной с "Детали заказа" по номеру заказа, соединенной с "Товары" по номеру товара, соединенной с "Категории" по идентификатору категорий, где описание категории = 'велосипед') по идентификатору клиента)

Внимание! Поскольку мы ищем конкретные заказы (велосипеды), процесс преобразования разделен на два шага, чтобы показать, что заказы необходимо отфильтровать, прежде чем применять OUTER JOIN.

SQL

```
SELECT Customers.CustFirstName || ' ' ||
    Customers.CustLastName AS CustFullName,
    RD.OrderDate, RD.ProductName,
    RD.QuantityOrdered, RD.QuotedPrice
FROM Customers
LEFT OUTER JOIN
(SELECT Orders.CustomerID, Orders.OrderDate,
    Products.ProductName,
    Order_Details.QuantityOrdered,
    Order_Details.Quoted Price
FROM ((Orders
INNER JOIN Order Details
ON Orders.OrderNumber =
    Order_Details.OrderNumber)
INNER JOIN Products
ON Order_Details.ProductNumber =
    Products.ProductNumber)
INNER JOIN Categories
ON Categories.CategoryID = Products.CategoryID
WHERE Categories.CategoryDescription = 'Bikes')
    AS RD
ON Customers.CustomerID = RD.CustomerID
```

Внимание! Этот запрос достаточно сложен, поскольку нужно предоставить список *всех* клиентов, соединенных в OUTER JOIN только с заказами на велосипеды. Если превратить Преобразование 1 напрямую в запрос SQL, то не найдется ни одного клиента, не заказавшего велосипед! Операция OUTER JOIN из Customers к Orders *возвратит* всех клиентов и любые заказы. Если добавить фильтр для выбора только заказов на велосипед, будут получены только клиенты, заказавшие велосипеды.

Преобразование 2 показывает, как это правильно сделать: нужно создать внутренний набор результатов, возвращающий только заказы на велосипеды, затем выполнить OUTER JOIN, который связывает его с Customers, чтобы получить окончательный ответ.

All_Customers_And_Any_Bike_Orders (913 строк)

CustFullName	OrderDate	ProductName	QuantityOrdered	QuotedPrice
Suzanne Viescas				
Will Thompson	1999-10-22	Trek 9000 Mountain Bike	5	$1,164.00
Will Thompson	1999-11-14	Trek 9000 Mountain Bike	6	$1,164.00
Will Thompson	1999-08-10	Viscount Mountain Bike	2	$635.00
Will Thompson	1999-08-04	Viscount Mountain Bike	5	$615.95
Will Thompson	1999-11-14	Trek 9000 Mountain Bike	4	$1,200.00
Will Thompson	1999-08-10	Trek 9000 Mountain Bike	3	$1,200.00
Will Thompson	1999-11-06	Trek 9000 Mountain Bike	2	$1,200.00
<< остальные строки >>				

(Похоже, что Уилл Томпсон — клиент, с которого нужно сдувать пылинки!)

База данных агентства эстрадных мероприятий

"List entertainers who have never been booked".
("Привести список эстрадных артистов, на которых никогда не было заявок".)

Преобразование/Уточнение: Select entertainer ID ~~and~~ entertainer stage name from ~~the~~ entertainers ~~table~~ left outer join~~ed with the~~ engagements ~~table~~ on entertainer ID where engagement number is null

(Выбрать идентификатор эстрадного артиста, псевдоним артиста из “Эстрадные артисты” с левым внешним соединением с “Ангажементы” по идентификатору эстрадного артиста, где номер ангажемента — Null

SQL

```
SELECT Entertainers.EntertainerID,
    Entertainers.EntStageName
FROM Entertainers
LEFT JOIN Engagements
ON  Entertainers.EntertainerID =
    Engagements.EntertainerID
WHERE Engagements.EngagementNumber IS NULL
```

Entertainers_Never_Booked (1 строка)

EntertainerID	EntStageName
1009	Katherine Ehrlich

“Show me all musical styles and the customers who prefer those styles”. (“Показать все музыкальные направления и клиентов, предпочитающих эти направления”.)

Преобразование/ Уточнение:

Select style ID, style name, customer ID, customer first name, ~~and~~ customer last name from ~~the~~ musical styles ~~table~~ left outer join~~ed with~~ (~~the~~ musical preferences ~~table~~ inner join~~ed with the~~ customers ~~table~~ on customer ID) on style ID
(Выбрать идентификатор направления, идентификатор клиента, имя клиента, фамилию клиента из “Музыкальные направления” с левым внешним соединением с (“Музыкальные предпочтения” с внутренним соединением с “Клиенты” по идентификатору клиента) по идентификатору направления)

SQL

```
SELECT Musical_Styles.StyleID,
    Musical_Styles.StyleName,
    Customers.Customer ID,
    Customers.CustFirstName,
    Customers.CustLastName
FROM Musical_Styles
LEFT OUTER JOIN (Musical_Preferences
INNER JOIN Customers
ON  Musical_Preferences.CustomerID =
    Customers.CustomerID)
ON  Musical_Styles.StyleID =
    Musical_Preferences.StyleID
```

All_Styles_And_Any_Customers (41 строка)

StyleID	StyleName	CustomerID	CustFirstName	CustLastName
1	40s Ballroom Music	10015	David	Nathanson
1	40s Ballroom Music	10011	Joyce	Bonnicksen
2	50s Music			
3	60s Music	10002	Ann	Fuller
4	70s Music	10007	Amelia	Buchanan
5	80s Music	10014	Mark	Rosales
6	Country	10009	Sarah	Thompson
–	Classical	10005	Elizabeth	Hallmark
<< остальные строки >>				

(Выглядит так, будто никто не любит музыку 50-х!)

Внимание! Мы очень осторожно выражаем словами условие FROM, чтобы вследствие воздействия на систему базы данных она в первую очередь выполнила INNER JOIN между Musical_Preferences и Customers, а затем OUTER JOIN его результата с Musical_Styles. Если база данных имеет тенденцию выполнять соединение слева направо, то нужно записать условие FROM вначале с INNER JOIN, за которым следует RIGHT OUTER JOIN с Musical_Styles. В Microsoft Access требуется указать INNER JOIN как вложенный оператор SELECT, чтобы заставить его вернуть правильный ответ.

База данных лиги игры в боулинг

"Show me tournaments that haven't been played yet".
("Показать турниры, которые еще не разыгрывались".)

Преобразование/ Уточнение:

Select tourney ID, tourney date, ~~and~~ tourney location from ~~the~~ tournaments ~~table~~ left outer join~~ed with the~~ tourney matches ~~table~~ where match ID is null
(Выбрать идентификатор турнира, дату и место проведения неразыгранного турнира из "Турниры", соединенной с "Матчи турнира", где идентификатор игры — null)

SQL

```
SELECT Tournaments.TourneyID,
    Tournaments.TourneyDate,
    Tournaments.TourneyLocation
FROM Tournaments
LEFT JOIN Tourney_Matches
ON Tournaments.TourneyID =
    Tourney_Matches.TourneyID
WHERE Tournev_Matches.MatchID IS NULL
```

Torney_Not_Yet_Played (6 строк)

TourneyID	TourneyDate	TourneyLocation
15	2001-05-01	Red Rooster Lanes
16	2001-05-08	Thunderbird Lanes
17	2001-05-15	Bolero Lanes
18	2001-05-22	Sports World Lanes
19	2001-05-29	Imperial Lanes
20	2001-06-05	Totem Lanes

"List all bowlers and any games they bowled over 180".
("Привести список всех игроков в боулинг и всех игр,
в которых они набрали больше 180 очков".)

Преобразование 1: Select bowler name, tourney date, tourney location, match ID, and raw score from the bowlers table left outer joined with the bowler scores table on bowler ID, then inner joined with the tourney matches table on match ID, then finally inner joined with the tournaments table on tournament ID where raw score in the bowler scores table is greater than 180

(Выбрать имя игрока в боулинг, дату турнира, место проведения турнира, идентификатор матча и предварительное количество очков из таблицы "Игроки в боулинг" с левым внешним соединением с таблицей "Очки игроков в боулинг" по идентификатору игрока в боулинг, внутренне соединенной с таблицей "Матчи турнира" по идентификатору матча, затем, наконец, внутренне соединенной с таблицей "Турниры" по идентификатору турнира, где значение предварительных очков в таблице "Очки игрока в боулинг" больше 180)

Преобразование 2/ Уточнение:

Select bowler name, tourney date, tourney location, match ID, ~~and~~ raw score from ~~the~~ bowlers ~~table~~ left outer join~~ed with~~ (Select tourney date, tourney location, match ID, bowler ID, ~~and~~ raw score from ~~the~~ bowler scores ~~table~~ inner join~~ed with the~~ tourney matches ~~table~~ on bowler ID, ~~then~~ join~~ed with the~~ tournaments ~~table~~ on tournament ID where raw score ~~is greater than~~ > 180) on bowler ID
(Выбрать имя игрока в боулинг, дату турнира, место проведения турнира, идентификатор матча, предварительное количество очков из “Игроки в боулинг” с левым внешним соединением с (Выбрать дату турнира, место проведения турнира, идентификатор матча, предварительное количество очков из “Очки игроков в боулинг”, внутренне соединенной с “Матчи турнира” по идентификатору игрока в боулинг, соединенной с “Турниры” по идентификатору турнира, где значение предварительных очков > 180) по идентификатору игрока)

SQL

```
SELECT Bowlers.BowlerLastName || ', ' ||
    Bowlers.BowlerFirstName AS BowlerName,
    TI.TourneyDate, TI.TourneyLocation,
    TI.MatchID, TI.RawScore
FROM Bowlers
LEFT JOIN
(SELECT Tournaments.TourneyDate,
    Tournaments.TourneyLocation,
    Bowler_Scores.MatchID,
    Bowler_Scores.BowlerID,
    Bowler_Scores.RawScore
FROM (Bowler_Scores
INNER JOIN Tourney_Matches
ON Bowler_Scores.MatchID =
    Tourney_Matches.MatchID)
INNER JOIN Tournaments
ON Tournaments.TourneyID =
    Tourney_Matches.TourneyID
WHERE Bowler_Scores.RawScore>180) AS TI
ON Bowlers.BowlerID = TI.BowlerID
```

Внимание! Угадали! Это еще один пример, где вначале требуется построить отфильтрованный набор результата для INNER JOIN, а затем выполнить OUTER JOIN его с таблицей, из которой нужны “все” строки.

All_Bowlers_And_Scores_Over_180 (106 строк)

BowlerName	TourneyDate	TourneyLocation	MatchID	RawScore
Fournier, Barbara				
Fournier, David				
Kennedy, John	1999-07-10	Totem Lanes	21	189
Kennedy, John	1999-09-04	Acapulco Lanes	53	191
Kennedy, John	1999-08-14	Imperial Lanes	41	188
Kennedy, John	1999-06-26	Imperial Lanes	13	182
Kennedy, John	1999-06-05	Red Rooster Lanes	1	191
Kennedy, John	1999-07-24	Red Rooster Lanes	29	182
<< остальные строки >>				

База данных расписаний занятий

"List the faculty members not teaching a class"
("Привести список преподавателей, которые ничего не преподают".)

Преобразование/ Уточнение:

Select staff first name ~~and~~ staff last name from ~~the~~ staff ~~table~~ left join~~ed with the~~ faculty classes ~~table~~ on staff ID where class ID is null
(Выбрать имя, фамилию преподавателя из "Персонал" с левым внешним соединением с "Курсы лекций преподавателя" по идентификатору преподавателя, где идентификатор курса лекций — Null)

SQL

```
SELECT Staff.StfFirstName, Staff.StfLastName,
FROM Staff
LEFT JOIN Faculty_Classes
ON Staff.StaffID = Faculty_Classes.StaffID
WHERE Faculty.Classes.ClassID IS NULL
```

Staff_Not_Teaching (4 строки)

StfFirstName	StfLastName
Jeffrey	Smith
Tim	Smith
Kathryn	Patterson
Joe	Rosales III

"Display students who have never withdrawn from a class".
("Вывести на дисплей имена студентов, которые никогда не отказывались от курса лекций".)

Преобразование/ Уточнение:

Select student full name from ~~the~~ students ~~table~~ left outer join~~ed with~~ (Select student ID from ~~the~~ student schedules ~~table~~ inner join~~ed with the~~ student class status ~~table~~ on class status where class status description ~~is~~ = 'withdrew') on student ID where ~~the~~ student_schedules.student ID ~~in the student schedules table~~ is null
(Выбрать полное имя студента из "Студенты" с левым внешним соединением с (Выбрать идентификатор студента из "Расписание студента", внешне соединенной с "Состояние курса лекций студента" по состоянию курса лекций, где описание состояния курса лекций = 'пропущено') по идентификатору студента, где идентификатор — Null)

Student_Never_Withdrawn (15 строк)

StudFullName
Fuller, Andrew
Leverling, Sarah
Peacock, Carol
Callahan, Sally
Buchanan, Steven
Hallmark, Elizabeth
Kennedy, Sara
Fuller, Mary
<< остальные строки >>

SQL

```
SELECT Students.StudLastName || ', ' ||
    Students.StudFirstName AS StudFullName
FROM Students
LEFT OUTER JOIN
(SELECT Student_Schedules.StudentID
FROM Student_Class_Status
    INNER JOIN Student_Schedules
    ON Student_Class_Status.ClassStatus =
        Student_Schedules.ClassStatus
    WHERE Student_Class_Status.ClassStatusDescription =
        'withdrew') AS Withdrew
    ON Students.StudentID = Withdrew.StudentID
    WHERE Withdrew.StudentID IS NULL
```

"Shaw me all subject categories and any classes for all subjects".
("Показать все категории предметов и все курсы лекций для всех предметов".)

Преобразование/ Уточнение:

Select category description, subject name, classroom ID, start time, ~~and~~ duration from ~~the~~ categories ~~table~~ left outer join~~ed with the~~ subjects ~~table~~ on category ID, ~~then~~ left outer join~~ed with the~~ classes ~~table~~ on subject ID
(Выбрать описание категории, название предмета, идентификатор аудитории, время начала, продолжительность из “Категории” с левым внешним соединением с “Предметы” по идентификатору категории с левым внешним соединением с “Курсы лекций” по идентификатору предмета)

SQL

```
SELECT Categories.CategoryDescription,
    Subjects.SubjectName, Classes.ClassRoomID,
    Classes.StartTime, Classes.Duration
FROM (Categories
LEFT OUTER JOIN Subjects
ON Categories.CategoryID = Subjects.CategoryID)
LEFT OUTER JOIN Classes
ON Subjects.SubjectID = Classes.SubjectID
```

All_Categories_All_Subjects_Any_Classes (82 строки)

CategoryDescription	SubjectName	Classroom	StartTime	Duration
Accounting	Financial Accounting Fundamentals I	3313	9:00	50
Accounting	Financial Accounting Fundamentals I	3313	13:00	50
Accounting	Financial Accounting Fundamentals II	3415	8:00	50
Accounting	Fundamentals of Managerial Accounting	3415	10:00	50
Accounting	Intermediate Accounting	3315	11:00	50
Accounting	Business Tax Accounting	3313	14:00	50
Art	Introduction to Art	1231	10:00	50
Art	Design	1619	15:30	110
<< остальные строки >>				

Внимание! Мы снова были очень осторожны при построении последовательности и вложении соединений, чтобы быть уверенными, что получим ожидаемый ответ.

В последующем в наборе результатов не будет курсов лекций, запланированных для Developing a Business Plan (Разработка бизнес-плана), Computer Programming (Программирование на компьютере) и American Government (Американское правительство). Также будут отсутствовать предметы, запланированные для категорий Psychology (Психология), French (Французский язык) или German (Немецкий язык).

База данных рецептов

"List ingredients not use in any recipe yet".
("Привести список компонентов, которые пока еще не используются в рецептах".)

Преобразование/ Уточнение:	Select ingredient name from ~~the~~ ingredients ~~table~~ left outer join~~ed with the~~ recipe ingredients ~~table~~ on ingredient ID where recipe ID is null (Выбрать названия компонентов из "Компоненты" с левым внешним соединением с "Компоненты рецепта" по идентификатору компонента, где идентификатор рецепта — Null)

Ingredients_Not_Used (20 строк)

IngredientName
Halibut
Chicken, Fryer
Bacon
Iceberg Lettuce
Butterhead Lettuce
Scallop
Vinegar
Red Wine
<< остальные строки >>

```
SQL                 SELECT Ingredients.IngredientName
                    FROM Ingredients
                    LEFT OUTER JOIN Recipe_Ingredients
                    ON  Ingredients.IngredientID =
                        Recipe_Ingredients.IngredientID
                    WHERE Recipe_Ingredients.RecipeID IS NULL
```

"I need all the recipe types, and then all the recipe names,
and then any matching ingredient step numbers, ingredient quantities,
and ingredient measurements, and finally all ingredient names
from my Recipes database".

("Мне нужны виды рецептов, все заголовки рецептов, соответствующие номера шагов включения компонентов, количество компонента, измерение компонентов и, наконец, названия компонентов из базы данных "Рецепты")

Преобразование/ Уточнение:

Select ~~the~~ recipe class description, recipe title, ingredient name, recipe sequence number, amount ~~and~~ measurement description from ~~the~~ recipe classes ~~table~~ full outer join~~ed with the~~ recipes ~~table~~ on recipe class ID, ~~then~~ left outer join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID, ~~then~~ join~~ed with the~~ measurements ~~table~~ on measurement amount ID, ~~and then finally~~ full outer join~~ed with the~~ ingredients ~~table~~ on ingredient ID
(Выбрать описание вида рецепта, заголовок рецепта, название компонента, порядковый номер рецепта, количество, описание единиц измерения из "Виды рецепта" с полным внешним соединением с "Рецепты" по идентификатору вида рецептов, с левым внешним соединением с "Компоненты рецепта" по идентификатору рецепта, соединенной с "Единицы измерения" по идентификатору единиц измерения количества, с полным внешним соединением с "Компоненты" по идентификатору компонента)

SQL

```
SELECT Recipe_Classes.RecipeClassDescription,
    Recipes.RecipeTitle,
    Ingredients.IngredientName,
    Recipe_Ingredients.RecipeSeqNo,
    Recipe_Ingredients.Amount,
    Measurements.MeasurementDescription
FROM Recipe_Classes
FULL OUTER JOIN
    (((Recipes
LEFT OUTER JOIN Recipe_Ingredients
ON  Recipe.RecipeID =
    Recipe_Ingredients.RecipeID
INNER JOIN Measurements
ON  Measurements.MeasureAmount.ID =
    Recipe_Ingredients.MeasureAmountID)
FULL OUTER JOIN Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID)
ON  Recipe_Classes.RecipeClassID =
    Recipes.RecipeClassID
```

Внимание! Приведенный пример представляет собой запрос, решенный в разделе о FULL OUTER JOIN. Мы включили его сюда, чтобы можно было видеть фактический результат. В примерах баз данных этот запрос в версии для Microsoft Access отсутствует, потому что Microsoft Access не поддерживает FULL OUTER JOIN.

All_Recipe_Classes_All_Recipes (109 строк)

RecipeClass Description	RecipeTitle	Ingredient Name	RecipeSeq No	Amount	Measurement Description
Main course	Irish Stew	Beef	1	1	Pound
Main course	Irish Stew	Onion	2	2	Whole
Main course	Irish Stew	Potato	3	4	Whole
Main course	Irish Stew	Carrot	4	6	Whole
Main course	Irish Stew	Water	5	4	Quarts
Main course	Irish Stew	Guinness beer	6	12	Ounce
Hors d'oeuvres	Salsa Buena	Jalapeño	1	6	Whole
Hors d'oeuvres	Salsa Buena	Tomato	2	2	Whole
<< остальные строки >>					

Итоги

В данной главе вы познакомились с миром операций OUTER JOIN и его отличием от INNER JOIN.

Вы также узнали, как построить LEFT или RIGHT OUTER JOIN, на простых примерах с использованием двух таблиц, а затем вложенные операторы SELECT и операторы с использованием нескольких операций JOIN. Операция OUTER JOIN, объединенная с проверкой на Null, эквивалентна операции DIFFERENCE (см. главу 7). При построении операторов с использованием нескольких операций OUTER JOIN можно столкнуться с некоторыми затруднениями, и мы объяснили, как их преодолеть. При обсуждении LEFT и RIGHT OUTER JOIN была рассмотрена задача, требующая нескольких операций OUTER JOIN, которая не может быть решена только с помощью LEFT или RIGHT JOIN.

При необходимости можно использовать тип соединения FULL OUTER JOIN вместе с другими операциями INNER и OUTER JOIN для получения правильного ответа. Были даны краткие пояснения варианта FULL OUTER JOIN — UNION JOIN.

С помощью OUTER JOIN можно решить самые разнообразные запросы. Мы привели почти дюжину примеров использования OUTER JOIN и показали логику, положенную в основу построения оператора, являющегося решением для каждого запроса.

Задачи для самостоятельного решения

Ниже приведены формулировки запросов и имена решений для этих запросов в учебных базах данных. Попрактикуйтесь немного и разработайте SQL для каждого запроса, а затем сверьте свой ответ с запросом, который сохранен нами в этих базах данных. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов,— важно, чтобы набор результатов был тем же.

База данных заказов на закупку

1. *"Show me customers who haven 't ever ordered a helmet"*.
   ("Показать клиентов, никогда не заказывавших шлем".
   (*Совет*: Это еще один запрос, где необходимо вначале построить INNER JOIN, чтобы найти все заказы на шлемы, а затем выполнить OUTER JOIN с Customers).
   Решение можно найти в Customers_No_Helmets (2 строки).

2. *"Display customers who have no sales rep (employees) in the same zip code"*.
   ("Вывести на экран клиентов, у которых нет торговых представителей (сотрудников) с одинаковым почтовым индексом".)
   Решение можно найти в Customers_No_Rep_Same_Zip (20 строк).

3. *"List all products and the dates for any orders"*.
   ("Привести список всех товаров и даты всех заказов".)
   Решение можно найти в All_Products_Any_Order_Dates (2682 строки).

База данных агентства эстрадных мероприятий

1. *"Display agents who haven't booked any entertainer"*.
   ("Вывести на экран агентов, которые никогда не подавали заявок на эстрадного артиста".)
   Решение можно найти в Agents_No_Contracts (1 строка).

2. *"List customers with no bookings"*.
   ("Привести список клиентов, не имеющих заявок".)
   Решение можно найти в Customers_No_Bookings (2 строки).

3. *"List all entertainers and any engagements they have booked"*.
   ("Привести список всех эстрадных артистов и всех ангажементов, где на них имеется заявка".)
   Решение можно найти в All_Entertainers_And_Any_Engagements (112 строк).

База данных лиги игры в боулинг

1. *"Display matches with no game data"*.
   ("Вывести на экран матчи без даты проведения игры".)
   Решение можно найти в Matches_Not_Played_Yet (1 строка).

2. *"Display all tournaments and any matches that have been played"*.
   ("Вывести на экран все турниры и все матчи, которые были сыграны".)
   Решение можно найти в All_Tourneys_Match_Results (174 строки).

База данных расписания занятий

1. *"Show me classes that have no students enrolled"*.
   ("Показать курсы лекций, на которые не записалось ни одного студента".)
   (*Совет*: Из Student_Classes требуются только строки с "enrolled" (записанные), но не с "completed" (заполненные) или "withdrew" (аннулированные).)
   Решение можно найти в Classes_No_Students_Enrolled (63 строки).

2. *"Display subjects with no faculty assigned"*.
   ("Вывести на экран предметы, по которым не назначены преподаватели".)
   Решение можно найти в Subjects_No_Faculty (1 строка)

3. *"List students not currently enrolled in any classes"*.
   ("Привести список всех студентов, которые в настоящее время не записались ни на один курс лекций".)
   (*Совет*: Нужно найти студентов, у которых состояние для курса лекций — "enrolled" (Записан), а затем найти студентов, которые отсутствуют в этом множестве.)
   Решение можно найти в Students_Not_Currently_Enrolled (2 строки).

4. *"Display all faculty and the classes they are scheduled to teach"*.
   ("Вывести на дисплей всех преподавателей и курсы лекций, которые они планируют читать".)
   Решение можно найти в All_Faculty_And_Any_Classes (79 строк).

База данных рецептов

1. *"Display missing types of recipes"*.
   ("Вывести на экран пропущенные виды рецептов".)
   Решение можно найти в Recipe_Classes_No_Recipes (1 строка).

2. *"Show me all ingredients and any recipes they're used in"*.
   ("Показать все компоненты и все рецепты, в которых они используются".)
   Решение можно найти в All_Ingredients_Any_Recipes (108 строк).

10

Операции UNION

"Я умоляю всех тех, добродетель которых
позволяет им благочестиво просить, чтобы они молились
о заключении этого союза".

— Сэм Хьюстон, *великий техасец*

Вопросы, рассматриваемые в данной главе:

- Что представляет собой UNION
- Запись запроса с использованием UNION
- Использование UNION
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

В главе 7 были представлены три фундаментальные операции с множествами: пересечение, разность и объединение. В главе 8 показано, как выполнить операцию, эквивалентную операции пересечения, связывая наборы результатов по значениям ключей и используя INNER JOIN. В главе 9 обсуждается, как запросить разность множеств, используя OUTER JOIN и проверяя наличие значений Null. В данной главе поясняется, как выполнить третью операцию — UNION.

Что представляет собой UNION

UNION позволяет выбрать с помощью SELECT строки из двух (или более) подобных наборов результата и объединить их в один набор. (Мы употребили слово "строки", а не "столбцы", поскольку вы уже знаете, как собрать вместе столбцы из двух и более наборов результатов, используя JOIN.) При выполнении в запросе JOIN столбцы из набора результата располагаются друг за другом. Например, запрашивая в JOIN столбец RecipeClassDescription из таблицы Recipe_Classes и RecipeTitle из таблицы Recipes, получим набор результатов, показанный на рис. 10.1.

Сначала посмотрим на синтаксис основной операции UNION (рис. 10.2).

RecipeClassDescription	RecipeTitle
Main course	Irish Stew
Main course	Fettuccini Alfredo
Main course	Pollo Picoso
Main course	Roast Beef
Main course	Huachinango Veracruzana (Red Snapper, Veracruz style)
Main course	Tourtière (French-Canadian Pork Pie)
Main course	Salmon Filets in Parchment Paper
Vegetable	Garlic Green Beans
<< остальные строки >>	

Рис. 10.1. *Извлечение данных из таблиц, используя JOIN*

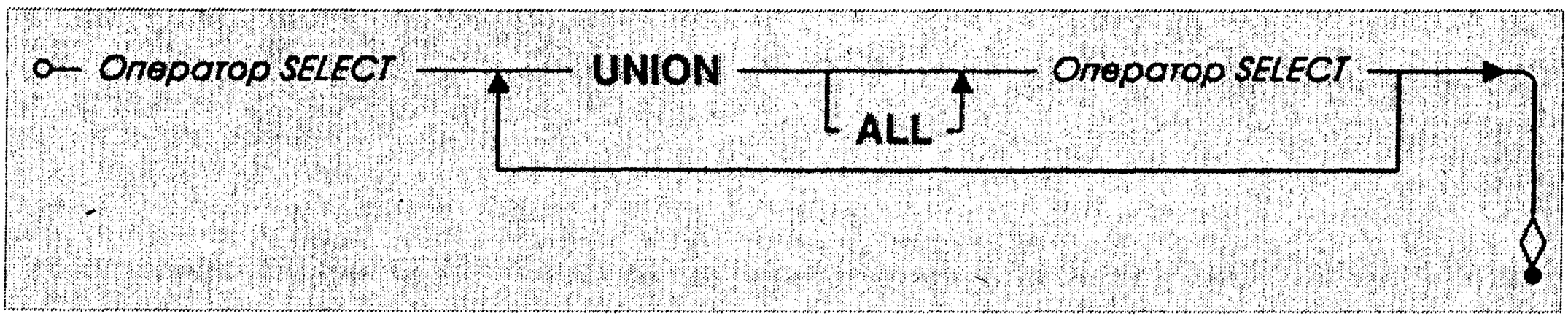


Рис. 10.2. *Диаграмма основного оператора UNION*

UNION чередует строки из одного набора результатов со строками из другого набора результатов. Каждый набор результатов определяется записью оператора SELECT, который может включать не только сложный JOIN в условие FROM, но также условия WHERE, HAVING и GROUP BY. Затем они соединяются с ключевым словом UNION. Если запрашивается RecipeClass-Description из таблицы Recipe_Classes, объединенный (UNION) с RecipeTitle из таблицы Recipes, то будет получен ответ, показанный на рис. 10.3.

Заметьте, что в наборе результатов получен только один столбец. Имя столбца наследуется от столбца первой таблицы, выбранной для включения в выражение SELECT, но столбец включает как информацию RecipeTitle (Asparagus), так и RecipeClassDescriptions (Dessert). Вместо появления их друг за другом данные из этих двух столбцов чередуются по вертикали.

RecipeClassDescription
Asparagus
Coupe Colonel
Dessert
Fettuccini Alfredo
Garlic Green Beans
Hors d'oeuvres
Huachinango Veracruzana (Red Snapper, Veracruz style)
Irish Stew
<< остальные строки >>

Рис. 10.3. *Извлечение данных из двух таблиц с использованием UNION*

При изучении диаграммы на рис. 10.2 у вас может возникнуть вопрос, для чего здесь указано необязательное ключевое слово ALL. Когда это слово не включается, СУБД исключает все строки, имеющие повторяющиеся значения. Например, если присутствуют RecipeClassDescription из Dessert и RecipeTitle из Dessert, то в окончательном наборе результатов будет получена только одна строка Dessert. Наоборот, при указании ключевого слова ALL повторяющиеся строки не исключаются.

Для выполнения UNION два набора результатов должны удовлетворять определенным требованиям. В первую очередь каждый из двух операторов SELECT, связываемых в UNION, должен иметь одинаковое количество выходных столбцов, определенных после ключевого слова SELECT, так что набор результата будет иметь то же самое количество столбцов. Во-вторых, каждый соответствующий столбец должен обладать свойством, которое в стандарте SQL называется "допускающим сопоставление".

Внимание! Полный стандарт SQL-92 позволяет соединять в UNION множества, не являющиеся подобными. Однако большинство коммерческих реализаций поддерживают базовый стандарт, или стандарт "начального уровня", описываемый здесь. Возможно, ваша СУБД позволяет использовать UNION более творчески.

Допускается сравнение символьных значений только с символьными значениями, цифровых значений с цифровыми значениями, а значений дата/время со значениями дата/время. Хотя некоторые системы баз данных допускают смешение типов данных при сравнении, сравнение символьного значения, например "John", с цифровым значением, например 55, обычно не имеет смысла. Чтобы сравнение двух столбцов в условии WHERE имело смысл, эти столбцы должны "допускать сопоставление". Именно это подразумевается стандартом, когда он требует, чтобы столбцы из одного набора результатов, которые нужно соединить в UNION со столбцом из другого набора результатов, имели тип данных, "допускающий сопоставление".

Запись запросов с UNION

В предыдущих главах, касающихся INNER JOIN и OUTER JOIN, мы изучали построение оператора SELECT с использованием условий SELECT, FROM и WHERE. Основное внимание было сосредоточено на построении сложных соединений в условии FROM. Для построения UNION необходимо пойти дальше, до *выражений SELECT*, которые связывают два или более операторов SELECT в операторе UNION. Каждый оператор SELECT может иметь как простое, так и сложное условие FROM, которое требуется для выполнения задания.

Использование простого оператора SELECT

Начнем с простого: запишем UNION двух простых операторов SELECT, которые используют одну таблицу в условии FROM. На рис. 10.4 представлена синтаксическая диаграмма для UNION двух простых операторов SELECT.

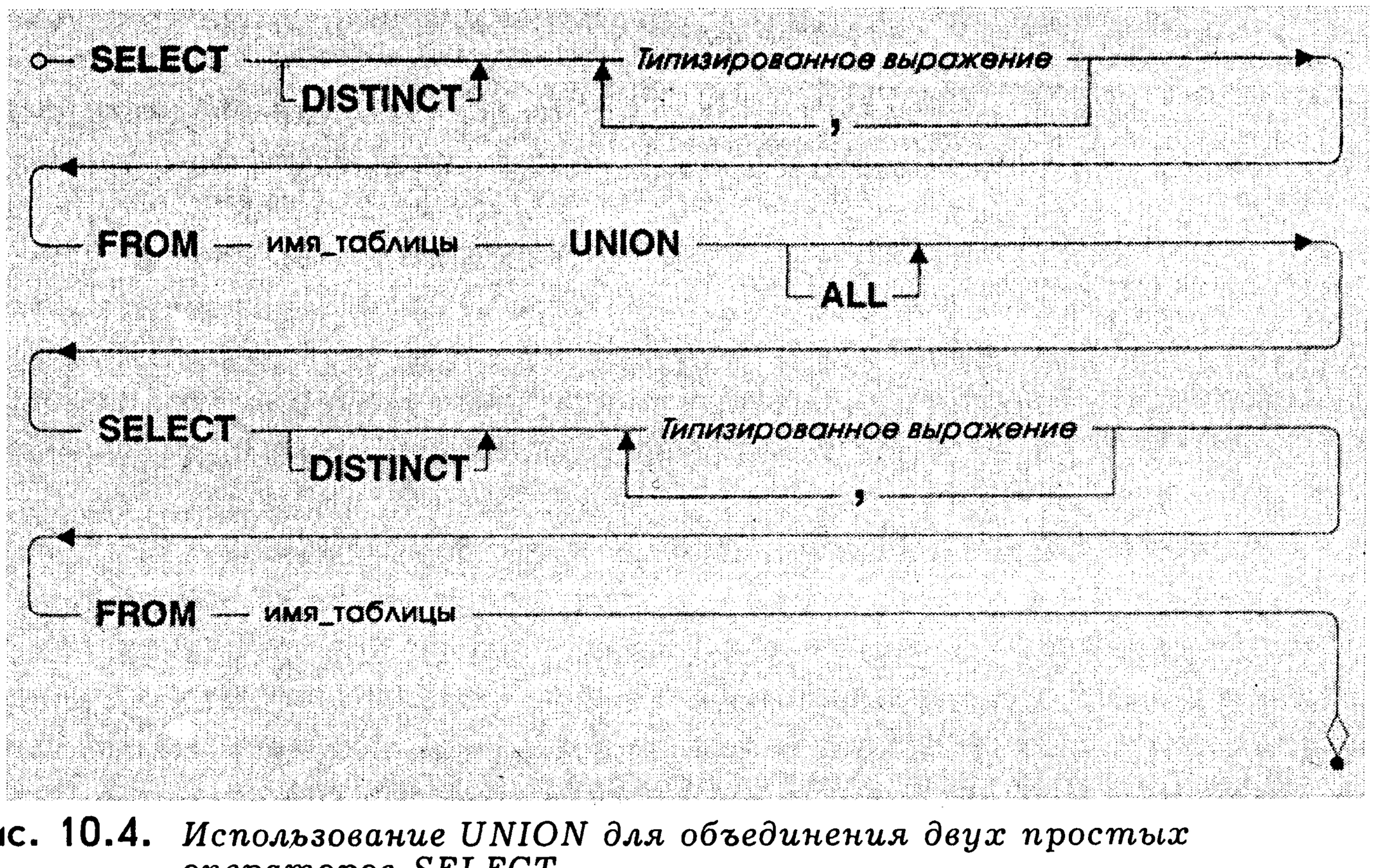


Рис. 10.4. *Использование UNION для объединения двух простых операторов SELECT*

В отличие от случая, когда запрашивался JOIN, сейчас все выполняется в операторе UNION, который определяется для объединения двух операторов SELECT. Если опущено необязательное ключевое слово ALL, СУБД исключит все найденные повторяющиеся строки. Это означает, что набор результата из запроса может иметь меньше строк, чем сумма строк, возвращенных из каждого набора результатов, участвующего в UNION. Однако, если ключевое слово ALL включено, количество строк в наборе результатов будет равно сумме количества строк в двух участвующих наборах результатов.

Внимание! Стандарт SQL также определяет условие CORRESPONDING, которое можно поместить после ключевого слова UNION для указания того, что нужен UNION, выполненный путем сравнения столбцов, имеющих одинаковые имена в каждом из наборов результата. Также можно еще больше ограничить множество сравнения, включив конкретный список имен столбцов после ключевого слова CORRESPONDING. Нам не удалось найти эту возможность в основных коммерческих реализациях, но не исключено, что она будет поддерживаться в следующих версиях.

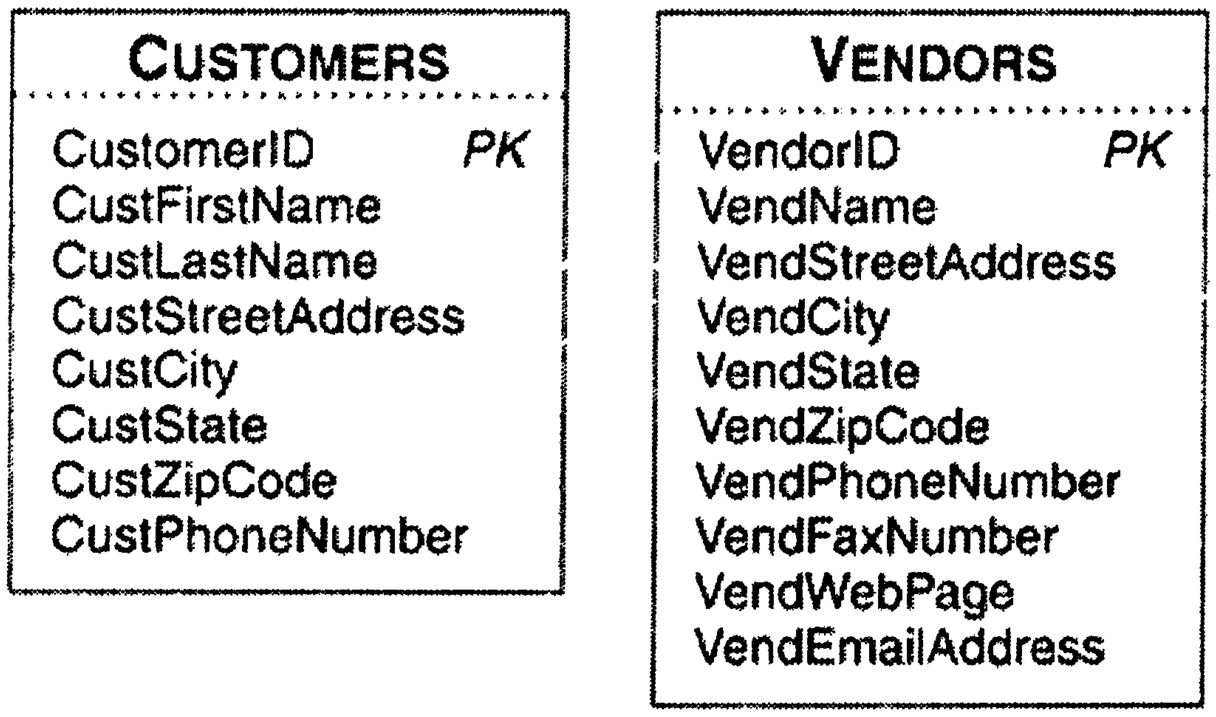


Рис. 10.5. *Таблицы Customers и Vendors из учебной базы данных Sales Order*

Создадим простой UNION — список адресов клиентов и поставщиков из учебной базы данных Sales Order (Заказы на закупку). На рис. 10.5 представлены две нужные таблицы.

Между этими двумя таблицами нет "естественной" связи, но они обе содержат столбцы, имеющие подобное значение и одинаковый тип данных. Для списка адресов требуются имя, наименование улицы, город, штат и почтовый индекс. Поскольку все эти поля в обеих таблицах представляют собой сопоставимые символьные данные, не стоит беспокоиться о типах данных. (Некоторые проектировщики БД могут представить почтовый индекс как число, но в данном случае все будет в порядке, пока столбец с почтовым индексом из одной таблицы имеет тип данных, сопоставимый с типом данных столбца с почтовым индексом из второй таблицы.)

Одна из проблем состоит в том, что в таблице Vendors имя представлено одним столбцом, а в Customers — двумя: CustFirstName и CustLastName. Чтобы получить одинаковое количество столбцов из обеих таблиц, нужно построить выражение по двум столбцам из Customers для получения выражения с одним столбцом, чтобы выполнить UNION с одним столбцом для имени из Vendors. Построим запрос.

> **Внимание!** В данной главе будет использоваться метод "Запрос/Преобразование/Уточнение/SQL", введенный в главе 4.

"Build a single mailing list that consists of the name, address, city, state, and zip for customers and the name, address, city, state, and zip for vendors".
("Построить единый список адресов, который состоит из имени, адреса, города, штата и почтового индекса для клиентов и имени, адреса, города, штата и почтового индекса для поставщиков".)

Преобразование: Select customer full name, customer address, customer city, customer state, and customer zip code from the customers table combined with vendor name, vendor address, vendor city, vendor state, and vendor zip code from the vendors table
(Выбрать полное имя клиента, адрес клиента, город клиента, штат клиента и почтовый индекс клиента

из таблицы “Клиенты”, в соединении с именем поставщика, адресом поставщика, городом поставщика, штатом поставщика и почтовым индексом поставщика из таблицы “Поставщики”)

Уточнение: Select customer full name, customer address, customer city, customer state, ~~and~~ customer zip code from ~~the~~ customers ~~table combined with~~ UNION vendor name, vendor address, vendor city, vendor state, ~~and~~ vendor zip code from ~~the~~ vendors ~~table~~
(Выбрать полное имя клиента, адрес клиента, город клиента, штат клиента, почтовый индекс клиента из “Клиенты” UNION (в соединении) с именем поставщика, адресом поставщика, городом поставщика, штатом поставщика, почтовым индексом поставщика из “Поставщики”)

SQL
```
SELECT Customers.CustLastName || ', ' ||
    Customers.CustFirstName AS MailingName,
    Customers.CustStreetAddress, Customers.CustCity,
    Customers.CustState, Customers.CustZipCode
FROM Customers
UNION
SELECT Vendors.VendName,
    Vendors.VendStreetAddress, Vendors.VendCity,
    Vendors.VendState, Vendors.VendZipCode
FROM Vendors
```

Каждый оператор SELECT генерирует пять столбцов, но мы должны использовать выражение для объединения двух столбцов с именем в таблице “Клиенты” в один столбец. Все столбцы из обоих операторов SELECT являются символьными данными, поэтому отсутствует проблема обеспечения их сопоставимости.

Может быть, вы хотите знать, какие имена будут иметь столбцы, которые представляют вывод для этого запроса. Хороший вопрос! Стандарт SQL определяет, что когда имена соответствующих столбцов одинаковы (например, имя четвертого столбца первого оператора SELECT и имя четвертого столбца второго оператора SELECT), то оно является именем выходного столбца. Если имена столбцов различаются (как в уже построенном нами примере), Стандарт SQL устанавливает, что имя “зависит от реализации и отличается от <column name> любого иного столбца любой таблицы, ... содержащейся в операторе SQL”.

На обычном языке это означает, что СУБД решает, какие имена назначить выходным столбцам. Система соответствует стандарту SQL до тех пор, пока имя не появляется в позиции некоторого другого столбца в одном из наборов результатов, участвующих в UNION. Большинство коммерческих СУБД по умолчанию определяют

их именами столбцов в первом операторе SELECT. Для предыдущего примера это означает, что имена столбцов будут MailingName, CustStreetAddress, CustCity, CustState и CustZipCode.

Здесь в UNION не включено ключевое слово ALL. Хотя маловероятно, что фамилия и имя клиента совпадут с именем поставщика (не беспокойтесь про адрес, город, штат и почтовый индекс), желательно все же избежать повторяющихся почтовых адресов. Если вы уверены, что какие-либо повторения в двух или более множествах в UNION отсутствуют, то можете включить ключевое слово ALL. Использование ALL ускорит выполнение запроса, поскольку системе базы данных не потребуется устранять повторения.

Объединение сложных операторов SELECT

Операторы SELECT, объединенные в операторе UNION, могут быть настолько сложными, насколько это требуется для выполнения задачи. Единственное ограничение состоит в том, что оба оператора SELECT должны, естественно, обеспечить одинаковое количество столбцов, а столбцы в каждой соответствующей позиции должны иметь сопоставимые типы данных.

Предположим, что нужен список всех клиентов и заказанных ими велосипедов, объединенный со всеми поставщиками и поставляемыми ими велосипедами. Вначале определим все необходимые таблицы. На рис. 10.6 представлены таблицы, необходимые для связывания клиентов с товарами.

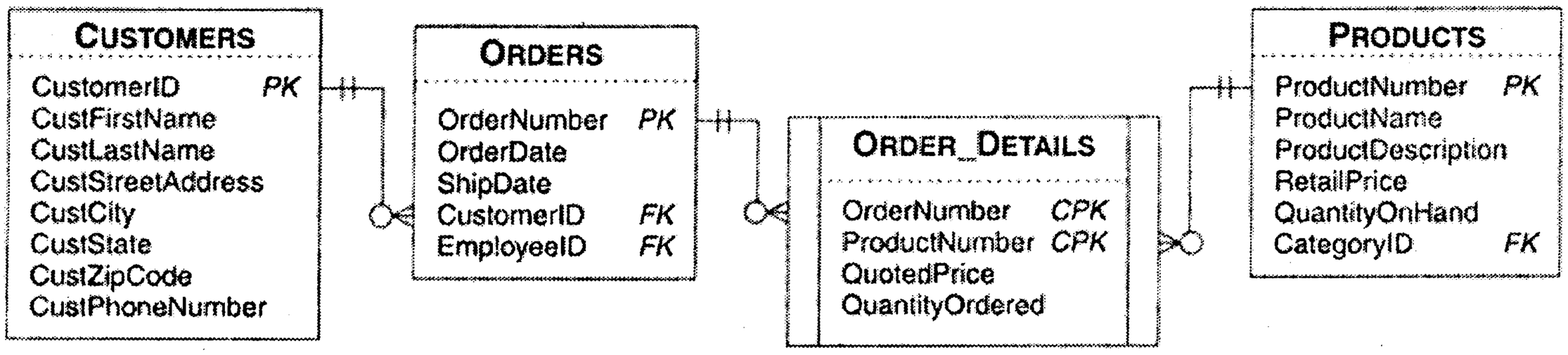


Рис. 10.6. *Отношения между таблицами, необходимые для связывания клиентов с заказанными ими товарами*

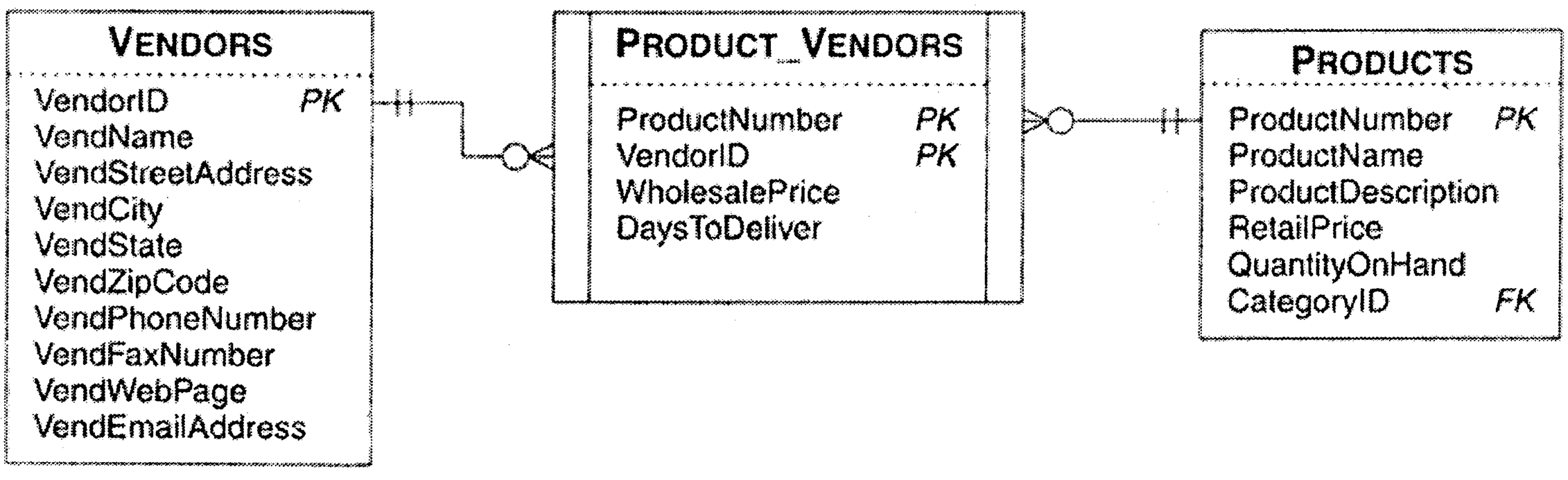


Рис. 10.7. *Отношения между таблицами, необходимые для связывания поставщиков с продаваемыми ими товарами*

Похоже, что требуется связать четыре таблицы. Если нужно найти поставщиков и продаваемые ими товары, то потребуются таблицы, представленные на рис. 10.7.

Можно использовать вложение нескольких условий JOIN, чтобы связать некоторое количество таблиц вместе для сбора информации, требуемой для решения сложной задачи. На рис. 10.8 представлено вложение для трех таблиц.

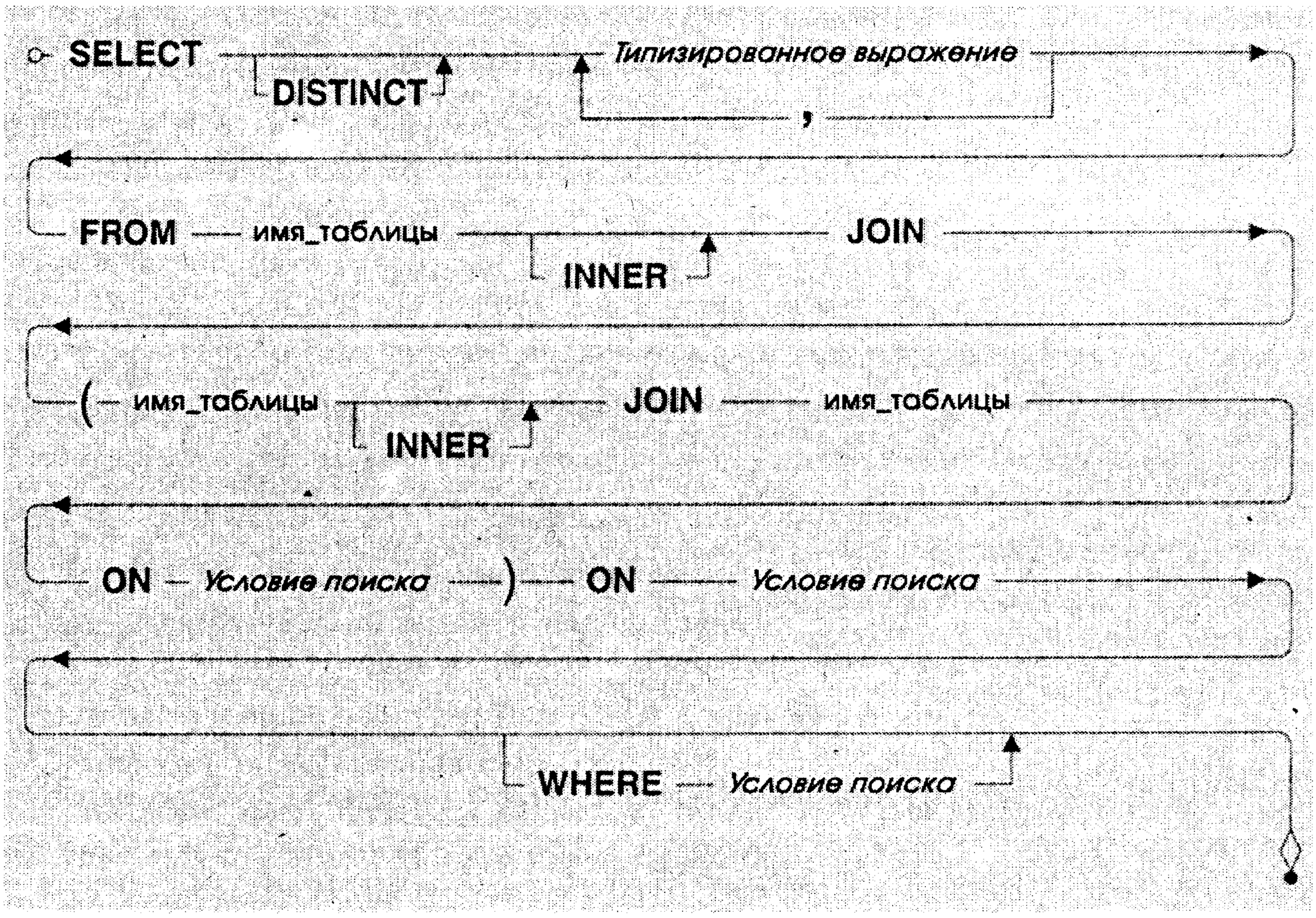


Рис. 10.8. *Соединение в JOIN трех таблиц*

Теперь у нас имеются все детали, необходимые для решения головоломки. Можно построить составной INNER JOIN для извлечения информации клиента, вставить ключевое слово UNION, а затем построить составной INNER JOIN для информации поставщика.

"List customers and the bikes they ordered combined with vendors and the bikes they sell".
("Привести список клиентов и заказанных ими велосипедов, вместе с поставщиками и продаваемыми ими велосипедами".)

Преобразование: Select customer full name and product name from the customers table joined with the orders table on customer ID, then joined with the order details table on order number, and then joined with the products table on product number where product name contains 'bike', combined with select vendor name and product name from the

vendors table joined with the product vendors table on vendor ID, and then joined with the products table on product number where product name contains 'bike' (Выбрать полное имя клиента и наименование товара из таблицы "Клиенты", соединенной с таблицей "Заказы" по идентификатору клиента, затем соединенной с таблицей "Детали заказа" по номеру заказа, затем соединенной с таблицей "Товары" по номеру товара, где наименование товара содержит 'bike', в соединении с (Выбрать имя поставщика и наименование товара из таблицы "Поставщики", соединенной с таблицей "Товары поставщика" по идентификатору поставщика и затем соединенной с таблицей "Товары" по наименованию товара, где наименование товара содержит 'велосипед'))

Уточнение:

Select customer full name ~~and~~ product name from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~then~~ join~~ed with the~~ order details ~~table~~ on order number, ~~and then~~ join~~ed with the~~ products ~~table~~ on product number where product name ~~contains~~ LIKE '%bike%', ~~combined with~~ UNION select vendor name ~~and~~ product name from ~~the~~ vendors ~~table~~ join~~ed with the~~ product vendors ~~table~~ on vendor ID, ~~and then~~ join~~ed with the~~ products ~~table~~ on product number where product name ~~contains~~ LIKE '%bike%'
(Выбрать полное имя клиента, наименование товара из "Клиенты", соединенной с "Заказы" по идентификатору клиента, соединенной с "Детали заказа" по номеру заказа, соединенной с "Товары" по номеру товара, где наименование товара LIKE '%bike%', UNION (в соединении с) (Выбрать имя поставщика, наименование товара из "Поставщики", соединенной с "Товары поставщика" по идентификатору поставщика, соединенной с "Товары" по наименованию товара, где наименование товара содержит LIKE '%bike%'))

SQL

```
SELECT Customers.CustLastName || ', ' ||
    Customers.CustFirstName AS FullName,
    Products.ProductName, 'Customer' AS RowID
FROM ((Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNúmber = Order_Details.OrderNumber)
```

```
INNER JOINProducts
ON  Products.ProductNumber =
    Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%bike%'
UNION
SELECT Vendors.VendName,  Products.ProductName,
    'Vendor'  AS RowID
FROM (Vendors INNER JOIN Product_Vendors
ON  Vendors.VendorID = Product_Vendors.VendorID)
    INNER JOIN Products
    ON  Products.ProductNumber =
        Product_Vendors.ProductNumber
    WHERE Products.ProductName LIKE '%bike%'
```

Хотя полученный результат оказался размером с ранчо, однако дело сделано! Обратите внимание, что в оба оператора SELECT также добавлен строковый литерал с именем RowID, так что будет легко увидеть, какие строки ведут свое происхождение от клиентов, а какие — от поставщиков. Можно было попробовать вставить ключевое слово DISTINCT в первый оператор SELECT, поскольку хороший клиент, возможно, заказывал конкретную модель велосипеда более одного раза. Поскольку в UNION не использовалось ключевое слово ALL, запрос исключит дубликаты. Если добавить DISTINCT, это может означать, что у системы базы данных дважды запрошено исключение дубликатов!

Когда необходимо построить запрос UNION, рекомендуется построить вначале отдельные операторы SELECT. Затем можно легко скопировать и вставить синтаксис каждого из операторов SELECT в новый запрос, отделив каждый оператор ключевым словом UNION.

Использование UNION более одного раза

UNION можно использовать для объединения нескольких множеств наборов результата. Можно последовательно записать спецификацию второго оператора SELECT с другим ключевым словом UNION и еще один оператор SELECT. Хотя в некоторых реализациях имеются ограничения на количество наборов результатов, которые можно объединить в UNION, теоретически можно добавлять UNION SELECT без всяких ограничений.

Предположим, что необходимо построить единый список адресов из трех различных таблиц — Customers, Employees и Vendors. На рис. 10.9 показана диаграмма синтаксиса для построения этого списка.

Очевидно, что нужно создать один оператор SELECT для извлечения всех имен и адресов из таблицы Customers, объединив (UNION) его с оператором SELECT для получения этой же информации из таблицы Employees и, наконец, с оператором SELECT для получения имен и адресов из таблицы Vendors.

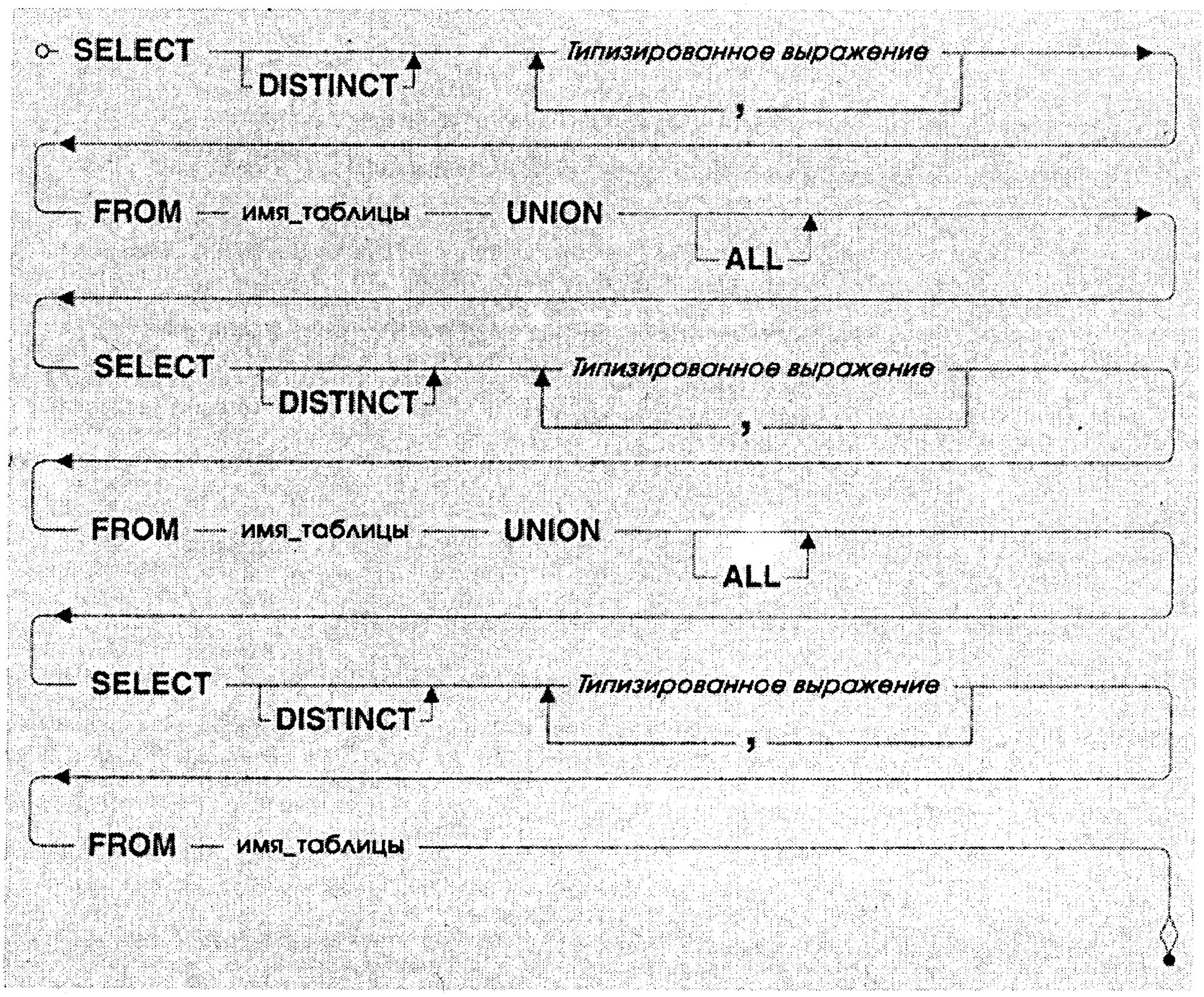


Рис. 10.9. *Создание UNION трех таблиц*

"Create a single mailing list for customers, employees, and vendors". ("Создать простой список почтовой рассылки для клиентов, сотрудников и поставщиков".)

Преобразование/ Уточнение:

Select customer full name, customer street address, customer city, customer state, ~~and~~ customer zip code from ~~the~~ customers ~~table combined with~~ UNION Select employee full name, employee street address, employee city, employee state, ~~and~~ employee zip code from ~~the~~ employees ~~table combined with~~ UNION Select vendor name, vendor street address, vendor city, vendor state, ~~and~~ vendor zip code from ~~the~~ vendors ~~table~~
(Выбрать полное имя, улицу в адресе клиента, город клиента, штат клиента, почтовый индекс клиента из "Клиенты" UNION (в соединении с) (Выбрать полное имя, улицу в адресе сотрудника, город сотрудника, штат сотрудника, почтовый индекс сотрудника из "Сотрудники") UNION (Выбрать полное имя, улицу в адресе поставщика, город поставщика, штат поставщика, почтовый индекс поставщика из "Сотрудники"))

```
SQL                SELECT Customers.CustFirstName || ' ' ||
                       Customers.CustLastName AS CustFullName,
                       Customers.CustStreetAddress, Customers.CustCity,
                       Customers.CustState, Customers.CustZipCode
                   FROM Customers
                   UNION
                   SELECT Employees.EmpFirstName || ' ' ||
                       Employees.EmpLastName AS EmpFullName,
                       Employees.EmpStreetAddress, Employees.EmpCity,
                       Employees.EmpState, Employees.EmpZipCode
                   FROM Employees
                   UNION
                   SELECT Vendors.VendName, Vendors.VendStreetAddress,
                       Vendors.VendCity, Vendors.VendState,
                       Vendors.VendZipCode
                   FROM Vendors
```

Конечно, если нужно отфильтровать список адресов для конкретного города, штата или диапазона почтовых индексов, то можно добавить условие WHERE для любого или всех операторов SELECT. Если, например, нужно создать список клиентов, сотрудников и поставщиков только в конкретном штате, то потребуется добавить условие WHERE в *каждый* из вложенных операторов SELECT. Также можно применить фильтр только к одному из операторов SELECT — создать список поставщиков из штата Техас, объединенный со всеми клиентами и всеми сотрудниками.

Сортировка UNION

Что касается сортировки результата UNION, то во многих СУБД полученный набор результатов уже отсортирован по выходным столбцам слева направо. Например, в UNION трех таблиц, только что построенном в предыдущем разделе, строки появятся в следующей последовательности: имя, улица и т. д.

Для сортировки строк по почтовому индексу можно добавить условие ORDER BY, но хитрость состоит в том, что это условие должно располагаться в самом конце последнего оператора SELECT. Условие ORDER BY применяется к результату UNION, а не к последнему оператору SELECT. На рис. 10.10 показано, как это сделать.

Как показано на диаграмме, можно повторить цикл UNION — оператор SELECT столько раз, сколько хотелось бы собрать наборов результатов, которые нужны для объединения, но условие ORDER BY должно появляться в конце. Что же нужно использовать в column_name или в column_# условия ORDER BY?. Вспомните, что сортируется вывод всех предыдущих частей выражения SELECT. Выходные имена столбцов “зависят от реализации”, но большинство СУБД используют имена столбцов, сгенерированные первым оператором SELECT.

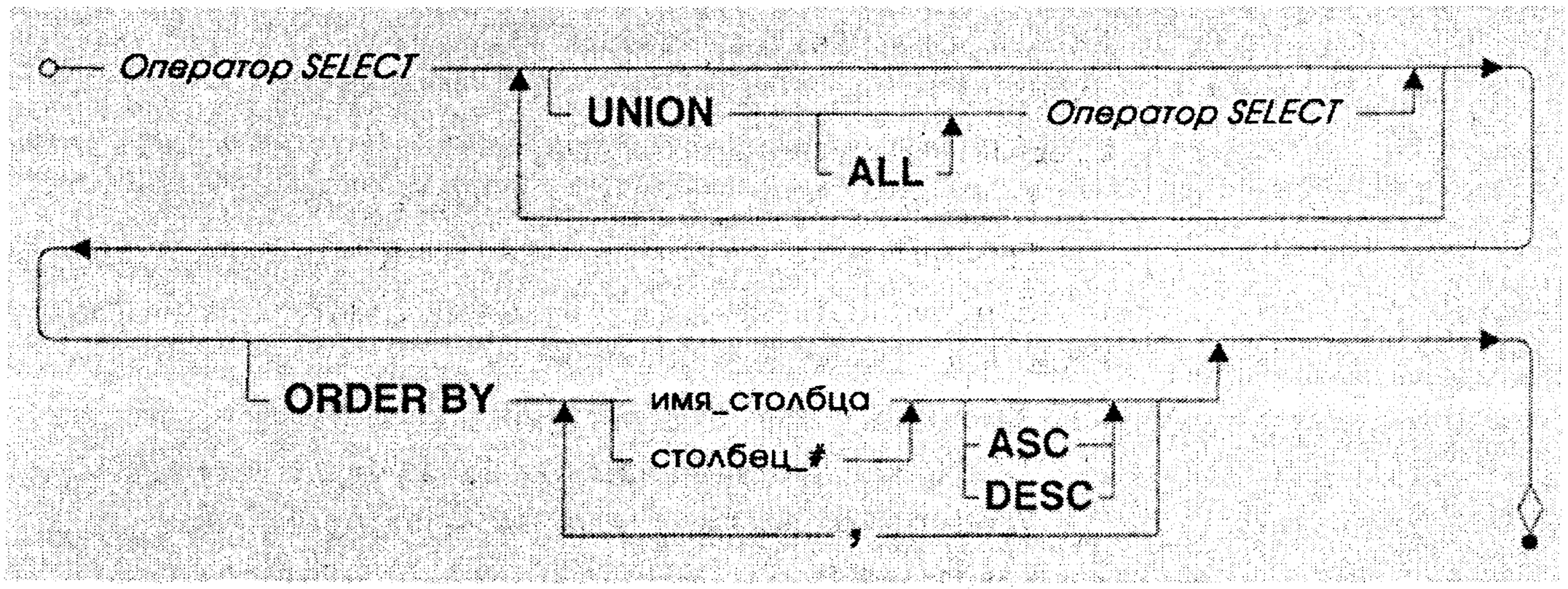


Рис. 10.10. *Добавление спецификации сортировки к запросу UNION*

Также можно определить относительные номера столбцов, начиная с 1 для первого столбца вывода. В запросе, который выводит имя, улицу, город, штат и почтовый индекс, необходимо определить column_# равным 5 (почтовый индекс является пятым столбцом) для сортировки по почтовому индексу.

Отсортируем запрос адресного списка, используя оба метода. Вот правильный синтаксис выполнения сортировки по имени столбца:

```
SQL                    SELECT Customers.CustFirstName, || ' ' ||
                           Customers.CustLastName AS CustFullName,
                           Customers.CustStreetAddress, Customers.CustCity,
                           Customers.CustState, Customers.CustZipCode
                       FROM Customers
                       UNION
                       SELECT Employees.EmpFirstName || ' ' ||
                           Employees.EmpLastName AS EmpFullName,
                           Employees.EmpStreetAddress, Employees.EmpCity,
                           Employees.EmpState, Employees.EmpZipCode
                       FROM Employees
                       UNION
                       SELECT Vendors.VendName, Vendors.VendStreetAddress,
                           Vendors.VendCity, Vendors.VendState,
                           Vendors.VendZipCode
                       FROM Vendors
                       ORDER BY CustZipCode
```

Конечно, предполагается, что имя столбца вывода, который нужно отсортировать, является именем столбца из первого оператора SELECT. Использование относительного номера столбца для определения выглядит так:

```
SQL                    SELECT Customers.CustFirstName, || ' ' ||
                           Customers.CustLastName AS CustFullName,
                           Customers.CustStreetAddress, Customers.CustCity,
                           Customers.CustState, Customers.CustZipCode
```

```
FROM Customers
UNION
SELECT Employees.EmpFirstName || ' ' ||
    Employees.EmpLastName AS EmpFullName,
    Employees.EmpStreetAddress, Employees.EmpCity,
    Employees.EmpState, Employees.EmpZipCode
FROM Employees
UNION
SELECT Vendors.VendName, Vendors.VendStreetAddress,
    Vendors.VendCity, Vendors.VendState,
    Vendors.YendZipCode
FROM Vendors
ORDER BY 5
```

Применение UNION

Скорее всего, UNION не будет использоваться так часто, как INNER JOIN и OUTER JOIN. Наиболее вероятно, что UNION будет использоваться для объединения двух или более подобных наборов результатов из различных таблиц. Хотя *можно* выполнить операцию UNION по двум наборам результатов из одной и той же таблицы или ряда таблиц, обычно такой тип задач можно решить, используя простой оператор SELECT, содержащий более сложное условие WHERE. Несколько подобных примеров приведено ниже и показан наиболее эффективный способ решения этих же задач с использованием условия WHERE вместо UNION.

Вот примеры типов задач, которые можно решить с использованием UNION:

"Показать имена и адреса всех клиентов и сотрудников".

"Привести список всех клиентов, заказавших велосипед, вместе со всеми клиентами, заказавшими шлем". (Это одна из тех задач, которую также можно решить в одном операторе SELECT и сложном условии WHERE.)

"Создать список адресов клиентов и поставщиков".

"Привести список клиентов, заказавших велосипед, вместе с поставщиками, поставляющими велосипеды".

"Создать список, в котором объединены агенты и эстрадные артисты".

"Выведите на экран дисплея объединенный список клиентов и эстрадных артистов".

"Создать список клиентов, которые любят современную музыку, вместе со списком эстрадных артистов, исполняющих современную музыку".

"Создать список адресов студентов и преподавателей".

"Показать студентов, имеющих средний балл 85 и выше по курсу "Искусство", вместе с преподавателями, читающими курс "Искусство" и имеющими рейтинг квалификации 9 или выше".

"Найти игроков в боулинг, имеющих предварительное количество очков 155 или больше на Зандербирд Лэйнс, вместе с игроками в боулинг, имеющими предварительное количество очков 140 или больше на Болеро Лэйнс". (Это еще одна задача, которая также может быть решена в одном операторе SELECT и сложном условии WHERE.)

"Привести список турнирных матчей, названий команд и капитанов команд для команд, начинающих на нечетных дорожках, вместе с матчами, названиями команд и капитанами команд для команд, начинающих на четных дорожках".

"Создать индексный указатель всех названий рецептов и компонентов".

"Отобразить на экране список всех компонентов и единиц измерения их количества по умолчанию вместе с компонентами, используемыми в рецептах, и единицами измерения количества для каждого рецепта".

Примеры операторов

Теперь вам известна механика построения запросов с использованием UNION, и вы видели некоторые типы запросов, ответы на которые можно найти с помощью UNION. Рассмотрим довольно устойчивое множество примеров из учебных БД.

Внимание! Поскольку многие их этих примеров используют сложные соединения, оптимизатор системы базы данных может выбрать другой способ решения. По этой причине несколько первых строк, которые показаны здесь, могут не совпадать точно с результатом, полученным вами, но общее количество строк должно быть тем же самым. Для упрощения процесса этапы перевода и уточнения для всех последующих примеров объединены.

База данных заказов на закупку

"Show me all the customer and employee names and addresses, including any duplicates, sorted by zip code".
("Показать имена и адреса всех клиентов и сотрудников, включая все повторяющиеся строки, отсортированные по почтовому индексу".)

Преобразование/ Уточнение:

Select customer first name, customer last name, customer street address, customer city, customer state, ~~and~~ customer zip code from ~~the~~ customers ~~table combined with~~ UNION

all select employee first name, employee last name, employee street address, employee city, employee state, ~~and~~ employee zip code from ~~the~~ employees ~~table~~, order by zip code
(Выбрать имя клиента, фамилию клиента, улицу клиента, город клиента, штат клиента, почтовый индекс клиента из “Клиенты” UNION ALL (Выбрать имя сотрудника, фамилию сотрудника, улицу сотрудника, город сотрудника, штат сотрудника, почтовый индекс сотрудника из “Сотрудники”,) упорядоченные по почтовому индексу)

SQL

```
SELECT Customers.CustFirstName,
    Customers.CustLastName,
    Customers.CustStreetAddress, Customers.CustCity,
    Customers.CustState, Customers.CustZipCode
FROM Customers
UNION ALL
SELECT Employees.EmpFirstName,
    Employees.EmpLastName,
    Employees.EmpStreetAddress, Employees.EmpCity,
    Employees.EmpState, Employees.EmpZipCode
FROM Employees
ORDER BY CustZipCode
```

Customers_UNION_ALL_Employees (37 строк)

CustFirstName	CustLastName	CustStreetAddress	CustCity	CustState	CustZipCode
Estella	Pundt	2500 Rosales Lane	Dallas	TX	75260
Michael	Davolio	672 Lamont Ave	Houston	TX	77201
Ryan	Ehrlich	455 West Palm Ave	San Antonio	TX	78284
Ryan	Ehrlich	455 West Palm Ave	San Antonio	TX	78284
Margaret	Peacock	667 Red River Road	Austin	TX	78710
Mark	Rosales	323 Advocate Lane	El Paso	TX	79915
Consuelo	Maynez	3445 Cheyenne Road	El Paso	TX	79915
Gregory	Piercy	4501 Wetland Road	Long Beach	CA	90809
<< остальные строки >>					

(Ehrlich, должно быть, является и клиентом, и сотрудником.)

"List all the customers who ordered a bicycle combined with all the customers who ordered a helmet".

Преобразование/ Уточнение:

Select customer first name ~~and~~ customer last name from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~then~~ join~~ed with the~~ order details ~~table~~ on order number, ~~and then~~ join~~ed with the~~ products ~~table~~ on product number where product name ~~contains~~ LIKE '%bike%', ~~combined with~~ UNION Select ~~unique~~ DISTINCT customer first name ~~and~~ customer last name from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~then~~ join~~ed with the~~ order details ~~table~~ on order number, ~~and then~~ join~~ed with the~~ products ~~table~~ on product number where product name ~~contains~~ LIKE '%helmet%'
(Выбрать имя клиента, фамилию клиента из "Клиенты", соединенной с "Заказы" по идентификатору клиента, соединенной с "Детали заказа" по номеру заказа, соединенной с "Товары" по номеру товара, где наименование товара LIKE '%bike%' UNION (Выбрать DISTINCT имя клиента, фамилию клиента из "Клиенты", соединенной с "Заказы" по идентификатору клиента, соединенной с "Детали заказа" по номеру заказа, соединенной с "Товары" по номеру товара, где наименование товара LIKE '%helmet%'))

SQL

```
SELECT Customers.CustFirstName,
    Customers.CustLastName, 'Bike' AS ProdType
FROM ((Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Detai1s
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
ON Products.ProductNumber =
    Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%bike%'
UNION
SELECT Customers.CustFirstName,
    Customers.CustLastName, 'Helmet' AS ProdType
FROM ((Customers INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
```

```
ON  Products.ProductNumber =
    Order_Details.ProductNumber
WHERE Products.ProductName LIKE '%helmet%'
```

Customer_Order_Bikes_UNION_Customer_Order_Helmets (54 строки)

CustFirstName	CustLastName	ProdType
Alaina	Hallmark	Bike
Alaina	Hallmark	Helmet
Allan	Davis	Bike
Allan	Davis	Helmet
Amelia	Buchanan	Bike
Amelia	Buchanan	Helmet
Andrea	Buchanan	Bike
Andrea	Buchanan	Helmet
<< остальные строки >>		

Это одна из тех задач, которую можно также решить с одним оператором SELECT и немного более сложным условием WHERE. Единственное преимущество использования UNION состоит в том, что легче добавить искусственный столбец "идентификатор множества" (в данном случае ProdType) к каждому набору результата, чтобы можно было видеть, к какому набору результатов относится каждый клиент. Однако большинство СУБД выполняет условие WHERE — даже имеющее сложный критерий — намного быстрее, чем UNION. Ниже приводится SQL для решения этой же задачи с условием WHERE:

```
SQL                 SELECT DISTINCT Customers.CustFirstName,
                       Customers.CustLastName
                    FROM
                       ((Customers INNER JOIN Orders
                    ON  Customers.CustomerID = Orders.CustomerID)
                    INNER JOIN Order_Details
                    ON  Orders.OrderNumber = Order_Details.OrderNumber)
                    INNER JOIN Products
                    ON  Products.ProductNumber =
                       Order_Details.ProductNumber
                    WHERE Products.ProductName LIKE '%bike%'
                       OR Products.ProductName LIKE '%helmet%'
```

Внимание! Если не используется UNION, то для исключения повторяющихся строк требуется ключевое слово DISTINCT. Вспомните, что UNION автоматически исключает повторения, если не определено UNION ALL. Можно определить DISTINCT в примере с UNION, но это потребует от системы базы данных выполнить больше работы, чем необходимо.

Customers_Bikes_Or_Helmets (27 строк)

CustFirstName	CustLastName
Alaina	Hallmark
Allan	Davis
Amelia	Buchanan
Andrea	Buchanan
Consuelo	Maynez
David	Callahan
David	Smith
<< остальные строки >>	

База данных эстрадных мероприятий

"Create a list that combines agents and entertainers".
("Создать список, объединяющий агентов и эстрадных артистов".)

Преобразование/ Уточнение:
Select agent full name from ~~the~~ agents ~~table combined~~ ~~with~~ UNION Select entertainer stage name from ~~the~~ entertainers ~~table~~
(Выбрать полное имя агента из "Агенты" UNION
(Выбрать псевдоним эстрадного артиста
из "Эстрадные артисты"))

SQL

```
SELECT Agents.AgtLastName || ', ' ||
    Agents.AgtFirstName AS Name, 'Agent' AS Type
FROM Agents
UNION
SELECT Entertainers.EntStageName,
    'Entertainer' AS Type
FROM Entertainers
```

Agents_ UNION_Entertainers (21 строка)

Name	Type
Albert Buchanan	Entertainer
Buchanan, Steven	Agent
Carol Peacock Trio	Entertainer
Caroline Coie Cuartet	Entertainer
Coldwater Cattle Company	Entertainer
Country Feeling	Entertainer
Fuller, Mary	Agent
<< остальные строки >>	

База данных расписания занятий

"Show me the students who have a grade of 85 or better in Art together with the faculty members who teach Art and have a proficiency rating of 9 or better".
("Показать студентов, имеющих балл 85 и выше по курсу лекций "Искусство", вместе с сотрудниками факультета, преподающими курс "Искусство" и имеющими рейтинг квалификации 9 и выше".)

Преобразование/ Уточнение:	Select student first name ~~aliased~~ as FirstName, student last name ~~aliased~~ as LastName, ~~and~~ grade ~~aliased~~ as Score from ~~the~~ students ~~table~~ join~~ed with the~~ student schedules ~~table~~ on student ID, ~~then~~ join~~ed with the~~ student class status ~~table~~ on class status, ~~then~~ join~~ed with the~~ classes ~~table~~ on class ID, ~~and then~~ join~~ed with the~~ subjects ~~table~~ on subject ID where class status description ~~is~~ = 'completed' and grade ~~is greater than or equal to~~ >= 85 and category ID ~~is~~ = 'art' ~~combined with~~ UNION Select staff first name, staff last name, ~~and~~ proficiency rating ~~aliased~~ as Score from ~~the~~ staff ~~table~~ join~~ed with the~~ faculty subjects ~~table~~ on staff ID, ~~and then~~ join~~ed with the~~ subjects ~~table~~ on subject ID where proficiency rating ~~is greater than~~ > 8 and category ID ~~is~~ = 'ART' (Выбрать имя студента как FirstName, фамилию студента как LastName, балл как Score из "Студенты", соединенной с "Расписание занятий студента" по идентификатору студента, соединенной со "Статус студента" по статусу курса, соединенной с "Курсы лекций" по идентификатору курса, соединенной с "Предмет" по идентификатору предмета, где описание состояния курса = 'завершено' и оценка >= 85, и идентификатор категории = 'искусство' UNION (Выбрать имя преподавателя, фамилию преподавателя, рейтинг квалификации как Score из "Персонал", соединенной с "Предметы факультета" по идентификатору преподавателя, соединенной с "Предметы" по идентификатору предмета, где рейтинг квалификации > 8 и идентификатор категории = 'ART'))

SQL	`SELECT Students.StudFirstName AS FirstName,`
	`  Students.StudLastName AS LastName,`
	`  Student_Schedules.Grade AS Score,`
	`  'Student' AS Type`

```
FROM (((Students INNER JOIN Student_Schedules
ON  Students.StudentID =
    Student_Schedules.StudentID)
INNER JOIN Student_Class_Status
ON  Student_Class_Status.ClassStatus =
    Student_Schedules.ClassStatus)
INNER JOIN Classes
ON  Classes.ClassID = Student_Schedules.ClassID)
INNER JOIN Subjects
ON  Subjects.SubjectID = Classes.SubjectID
WHERE Student_Class_Status.ClassStatusDescription =
    'Completed'
AND Student_Schedules.Grade >= 85
AND Subjects.CategoryID = 'ART'
UNION
SELECT Staff.StfFirstName,  Staff.StfLastName,
    Faculty_Subjects.ProficiencyRating AS Score,
    'Faculty'  AS Type
FROM (Staff INNER JOIN Faculty_Subjects
ON  Staff.StaffID = Faculty_Subjects.StaffID)
INNER JOIN Subjects
ON  Subjects.SubjectID = Faculty_Subjects.SubjectID
WHERE Faculty_Subjects.ProficiencyRating > 8
AND Subjects.CategoryID = 'ART'
```

Good_Art_Students_And_Faculty (15 строк)

FirstName	LastName	Score	Type
Alaina	Hallmark	10	Faculty
Amelia	Buchanan	10	Faculty
David	Nathanson	87.05	Student
Elizabeth	Hallmark	93.27	Student
James	Leverling	9	Faculty
John	Leverling	9	Faculty
John	Leverling	10	Faculty
Kendra	Bonnicksen	88.27	Student
<< остальные строки >>			

База данных лиги игры в боулинг

"List the tourney matches, team names and team captains for the teams starting on the odd lane together with the tourney matches, team names, and team captains for the teams starting on the even lane". ("Предоставить список матчей, названия команд и имена капитанов команд, начинающих на нечетных дорожках, вместе со списком матчей, названиями команд и именами капитанов команд, начинающих на четных дорожках".)

Преобразование/ Уточнение:

Select tourney location, tourney date, match ID, team name, ~~and~~ captain name from ~~the~~ tournament ~~table~~ join~~ed with the~~ tourney matches ~~table~~ on tourney ID, ~~then~~ join~~ed with the~~ teams ~~table~~ on odd lane team ID ~~in the tourney matches table equals~~ = team ID ~~in the teams table, and then~~ join~~ed with the~~ bowlers ~~table~~ on captain ID ~~in the teams table equals~~ = bowler ID ~~in the bowlers table, combined with~~ UNION Select tourney location, tourney date, match ID, team name, ~~and~~ captain name from ~~the~~ tournaments ~~table~~ join~~ed with the~~ tourney matches ~~table~~ on tourney ID, ~~then~~ join~~ed with the~~ teams ~~table~~ on even lane team ID ~~in the tourney matches table equals~~ = team ID ~~in the teams table, and then~~ join~~ed with the~~ bowlers ~~table~~ on captain ID ~~in the teams table equals~~ = bowler ID ~~in the bowlers table~~, order by tourney date and match ID (Выбрать место проведения турнира, дату турнира, идентификатор матча, название команды, имя капитана из "Турниры", соединенной с "Матчи турнира" по идентификатору турнира, соединенной с "Команды" по идентификатору команды нечетной дорожки = идентификатору команды, соединенной с "Игроки в боулинг" по идентификатору капитана = идентификатору игрока в боулинг UNION (Выбрать место проведения турнира, дату турнира, идентификатор матча, название команды, имя капитана из "Турниры", соединенной с "Матчи турнира" по идентификатору турнира, соединенной с "Команды" по идентификатору команды четной дорожки = идентификатору команды, соединенной с "Игроки в боулинг" по идентификатору капитана = идентификатору игрока в боулинг), отсортированные по дате турнира и идентификатору матча)

Bowling_Schedule (112 строк)

TourneyLocation	TourneyDate	MatchID	TeamName	Captain	Lane
Red Rooster Lanes	1999-06-05	1	Marlins	Fournier David	Odd Lane
Red Rooster Lanes	1999-06-05	1	Sharks	Patterson Ann	Even Lane
Red Rooster Lanes	1999-06-05	2	Barracudas	Sheskey Richard	Even Lane
Red Rooster Lanes	1999-06-05	2	Terrapins	Morgen-stern Iris	Odd Lane
Red Rooster Lanes	1999-06-05	3	Dolphins	Viescas Suzanne	Odd Lane
Red Rooster Lanes	1999-06-05	3	Orcas	Thomp-son Sarah	Even Lane
Red Rooster Lanes	1999-06-05	4	Manatees	Viescas Michael	Odd Lane
Red Rooster Lanes	1999-06-05	4	Swordfish	Rosales Joe	Even Lane
Thunderbird Lanes	1999-06-12	5	Marlins	Fournier David	Even Lane
Thunderbird Lanes	1999-06-12	5	Terrapins	Morgen-stern Iris	Odd Lane
<< остальные строки >>					

SQL

```
SELECT Tournaments.TourneyLocation,
    Tournaments.Tourney Date,
    Tourney_Matches.MatchID, Teams.TeamName,
    Bowlers.BowlerLastName || ' ' ||
    Bowlers.BowlerFirstName AS Captain,
    'Odd Lane' AS Lane
FROM ((Tournaments INNER JOIN Tourney_Matches
ON Tournaments.TourneyID =
    Tourney_Matches.TourneyID)
INNER JOIN Teams
ON Teams.TeamID =
    Tourney_Matches.OddLaneTeamID)
INNER JOIN Bowlers
ON Bowlers.BowlerID = Teams.CaptainID
UNION
```

```
SELECT Tournaments.TourneyLocation,
    Tournaments.TourneyDate,
    Tourney_Matches.MatchID, Teams.TeamName,
    Bowlers.BowlerLastName || ' ' ||
    Bowlers.BowlerFirstName AS Captain,
    'Even Lane' AS Lane
FROM ((Tournaments INNER JOIN Tourney_Matches
ON Tournaments.TourneyID =
    Tourney_Matches.TourneyID)
    INNER JOIN Teams
    ON Teams.TeamID =
        Tourney_Matches.EvenLaneTeamID)
    INNER JOIN Bowlers
    ON Bowlers.BowlerID = Teams.CaptainID
        ORDER BY 2, 3
```

Эти два оператора SELECT почти идентичны! Единственное различие между ними в том, что первый оператор SELECT связывает Tourney_Matches с Teams по Odd-LaneTeamID, а второй оператор использует EvenLaneTeamID.

База данных рецептов

"Create an index list of all the recipe classes, recipe titles, and ingredients".
("Создать индексный указатель всех видов рецептов, заголовков рецептов и компонентов".

Преобразование/ Уточнение:

Select recipe class description from ~~the~~ recipe classes ~~table~~ ~~combined with~~ UNION select recipe title from ~~the~~ recipes ~~table combined with~~ UNION select ingredient name from ~~the~~ ingredients ~~table~~
(Выбрать описание вида рецепта из "Виды рецептов" UNION (Выбрать заголовок рецепта из "Рецепты") UNION (Выбрать название компонента из "Компоненты"))

SQL

```
SELECT Recipe_Classes.RecipeClassDescription
AS IndexName, 'Recipe Class' AS Type
FROM Recipe_Classes
UNION
SELECT Recipes.RecipeTitle, 'Recipe' AS Type
FROM Recipes
UNION
SELECT Ingredients.IngredientName,
    'Ingredient' AS Type
FROM Ingredients
```

Classes_Recipes_Ingredients (101 строка)

IndexName	Type
Asparagus	Ingredient
Asparagus	Recipe
Bacon	Ingredient
Balsamic vinaigrette dressing	Ingredient
Beef	Ingredient
Beef drippings	Ingredient
Bird's custard powder	Ingredient
Black olives	Ingredient
<< остальные строки >>	

Итоги

В данной главе мы рассмотрели UNION и отличие между связыванием двух таблиц в JOIN и объединением двух таблиц в UNION.

Можно построить простой оператор UNION с использованием двух операторов SELECT, каждый из которых запрашивает столбцы из одной таблицы. Можно объединить два сложных оператора SELECT, каждый из которых использует JOIN по нескольким таблицам. Можно использовать UNION для объединения более двух наборов результатов. В завершение обсуждения синтаксиса UNION было показано, как отсортировать результат.

UNION полезен для решения самых разнообразных запросов. Мы показали примеры использования UNION для каждой из учебных баз данных, включая логику, лежащую в основе построения этих запросов.

Задачи для самостоятельного решения

Ниже приведены формулировки запросов и имена решений этих запросов в учебных базах данных. Попрактикуйтесь немного и разработайте SQL для каждого запроса, а затем сверьте свой ответ по запросу, который сохранен нами в этих базах данных. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов,— важно, чтобы набор результатов был тот же.

База данных заказов на закупку

1. *"List the customers who ordered a helmet together with the venders who provide helmets"*.
   ("Привести список клиентов, заказавших шлем, вместе с поставщиками этих шлемов".)
   (*Совет*: Для этого нужно создать UNION двух сложных операторов JOIN.)
   Решение можно найти в Customer_Helmets_Vendor_Helmets (96 строк).

База данных агентства эстрадных мероприятий

1. *"Display a combined list of customers and entertainers"*.
   ("Отобразить на экране объединенный список клиентов и эстрадных артистов".)
   (*Совет*: Будьте внимательны при создании выражения для одного из имен, обеспечивая одинаковое количество столбцов в обоих операторах SELECT.)
   Решение можно найти в Customers_UNION_Entertainers (28 строк).

2. *"Produce a list of customers who like contemporary music together with a list of entertainers who play contemporary music"*.
   ("Составить список клиентов, которым нравится современная музыка, вместе со списком эстрадных артистов, исполняющих современную музыку".)
   (*Совет*: Для решения этой задачи потребуется UNION двух сложных операторов JOIN.)
   Решение можно найти в Customers_Entertainers_Contemporary (5 строк).

База данных расписания занятий

1. *"Create a mailing list for students and staff, sorted by zip code"*.
   (Создать список адресов студентов и персонала, отсортированный по почтовому индексу".)
   (*Совет*: Для сортировки попытайтесь воспользоваться относительным номером столбца.)
   Решение можно найти в Student_Staff_Mailing_List (45 строк).

База данных лиги игроков в боулинг

1. *"Find the bowlers who had a raw score of 165 or better at Thunderbird Lanes combined with bowlers who had a raw score of 150 or better at Bolero Lanes"*.
   ("Найти игроков в боулинг с количеством очков 165 и выше на Зандербирд Лэйнс вместе с игроками в боулинг с количеством очков 150 и выше на Болеро Лэйнс".)
   (*Совет*: Это еще одна из тех задач, которые можно решить с использованием одного оператора SELECT и сложного условия WHERE.)

Решение с использованием UNION можно найти
в (Good_Bowlers_TBird_Bolero_UNION (120 строк).
Решение с использованием WHERE можно найти
в Good_Bowlers_TBird_Bolero_WHERE (125 строк).

2. *"Can you explain why the row counts are different in the above solution queries?"*.
("Можно ли объяснить, почему в приведенных решениях для запросов предварительные счета отличаются?)
(*Совет*: Попытайтесь воспользоваться UNION ALL из первого запроса.)

База данных рецептов

1. *"Display a list of all ingredients and their default measurement amounts together with ingredients used in recipes and the measurement amount for each recipe"*.
("Отобразить на экране все компоненты и единицы измерения их количества, определяемые по умолчанию, вместе с компонентами, используемыми в рецептах, и единицами измерения для каждого рецепта".)
(*Совет*: Для решения этой задачи требуется один простой JOIN и один сложный JOIN.)
Решение можно найти в Ingredient_Recipe_Measurements (144 строки).

11

Подзапросы

"Невозможно решить проблему,
используя тот же способ мышления, который ее породил".

— Альберт Эйнштейн

Вопросы, рассматриваемые в данной главе:

- Что представляет собой подзапрос
- Подзапросы как выражения со столбцами
- Подзапросы как фильтры
- Использование подзапросов
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

В предыдущих главах было показано множество способов работы с данными из нескольких таблиц. Все методы, рассмотренные до сих пор, фокусировались на связывании подмножеств информации — одного или нескольких столбцов и одной или нескольких строк из единой таблицы или запроса, вложенного в условие FROM, или с использованием оператора UNION. В данной главе представлены эффективные способы извлечения единственного столбца из таблицы или запроса и использования его в качестве типизированного выражения либо в условии SELECT, либо в условии WHERE.

Следует усвоить два основных момента:

1. В SQL всегда существует несколько способов решения конкретной задачи. В этой главе приводятся новые способы решения задач, уже рассмотренных в предыдущих главах.

2. Можно построить сложные фильтры, которые не зависят от таблиц условия FROM. Это важная концепция, потому что использование подзапросов в условии WHERE является единственным способом получения правильного количества строк в ответах, когда требуются строки из одной таблицы, основанные на отфильтрованном содержимом из других связанных таблиц.

Что представляет собой подзапрос

Говоря простым языком, *подзапрос* — это выражение SELECT, которое вложено в одно из условий оператора SELECT для образования оператора окончательного запроса. В данной главе подзапрос будет определен более формально и будет показано, как использовать его иным образом, чем в условии FROM.

Стандарт SQL определяет три типа подзапросов:

1. **Строковый подзапрос** — вложенное выражение SELECT, возвращающее больше одного столбца и только одну строку.
2. **Табличный подзапрос** — вложенное выражение SELECT, возвращающее один или несколько столбцов и ни одной или несколько строк.
3. **Скалярный подзапрос** — вложенное выражение SELECT, возвращающее только один столбец и не более одной строки.

Строковый подзапрос

Строки, возвращенные запросом, фильтруются путем построения предиката сравнения в условии WHERE, которое сравнивает значение столбца с литералом, выражением или другим столбцом. Простой запрос с использованием единственного предиката сравнения должен выглядеть подобно следующему:

```
SQL                 SELECT Customers.CustLastName
                    FROM Customers
                    WHERE Customers.CustAreaCode > 415
```

Стандарт SQL определяет *конструктор строкового значения*, используемый как часть предиката условия поиска в условии WHERE, HAVING или ON. Однако еще мало коммерческих систем базы данных поддерживают этот синтаксис. Вот пример условия WHERE, которое использует конструктор строкового значения:

```
SQL                 SELECT Customers.CustLastName
                    FROM Customers
                    WHERE
                        (Customers.CustAreaCode, Customers.CustZipCode) >
                        (415, '94110')
```

Здесь условие WHERE запрашивает строки, в которых комбинация CustAreaCode и CusrZipCode больше, чем комбинация 415 и 94110. Это то же самое, что и запрос:

```
SQL                 SELECT Customers.CustLastName
                    FROM Customers
                    WHERE (Customers.CustAreaCode > 415)
                    OR ((Customers.CustAreaCode = 415)
                    AND (Customers.CustZipCode > '94110'))
```

Здесь можно заменить оператор SELECT, который возвращает одну строку из двух столбцов — строковый подзапрос,— второй частью сравнения. Большинство коммерческих баз данных не поддерживает ни конструктор строкового значения, ни строковые подзапросы.

Табличные подзапросы

Табличные подзапросы уже широко использовались в предыдущих главах для определения сложного результата, который затем вкладывался в условие FROM другого запроса. В данной главе будет показано, как использовать табличные подзапросы в качестве источника списка значений для сравнения предиката IN. Мы также изучим несколько новых ключевых слов предиката сравнения, которые используются только в табличных подзапросах.

Скалярные подзапросы

Скалярный подзапрос используется в тех местах, где иначе требуется типизированное выражение. Он позволяет извлечь отдельный столбец или вычислить выражение из другой таблицы, которая не обязательно должна присутствовать в условии FROM основного запроса. Можно использовать отдельное значение, извлеченное скалярным подзапросом в списке столбцов, запрошенных в условии SELECT, или как значение для сравнения в условии WHERE.

Подзапросы как выражения со столбцами

Из главы 5 вы узнали об использовании выражений для формирования вычисляемых столбцов, которые должны выводиться запросом. Можно также использовать специальный тип оператора SELECT — подзапрос — для извлечения данных из другой таблицы, даже если эта таблица отсутствует в условии FROM.

Синтаксис

Вернемся к основам и рассмотрим простую форму оператора SELECT, показанного на рис. 11.1.

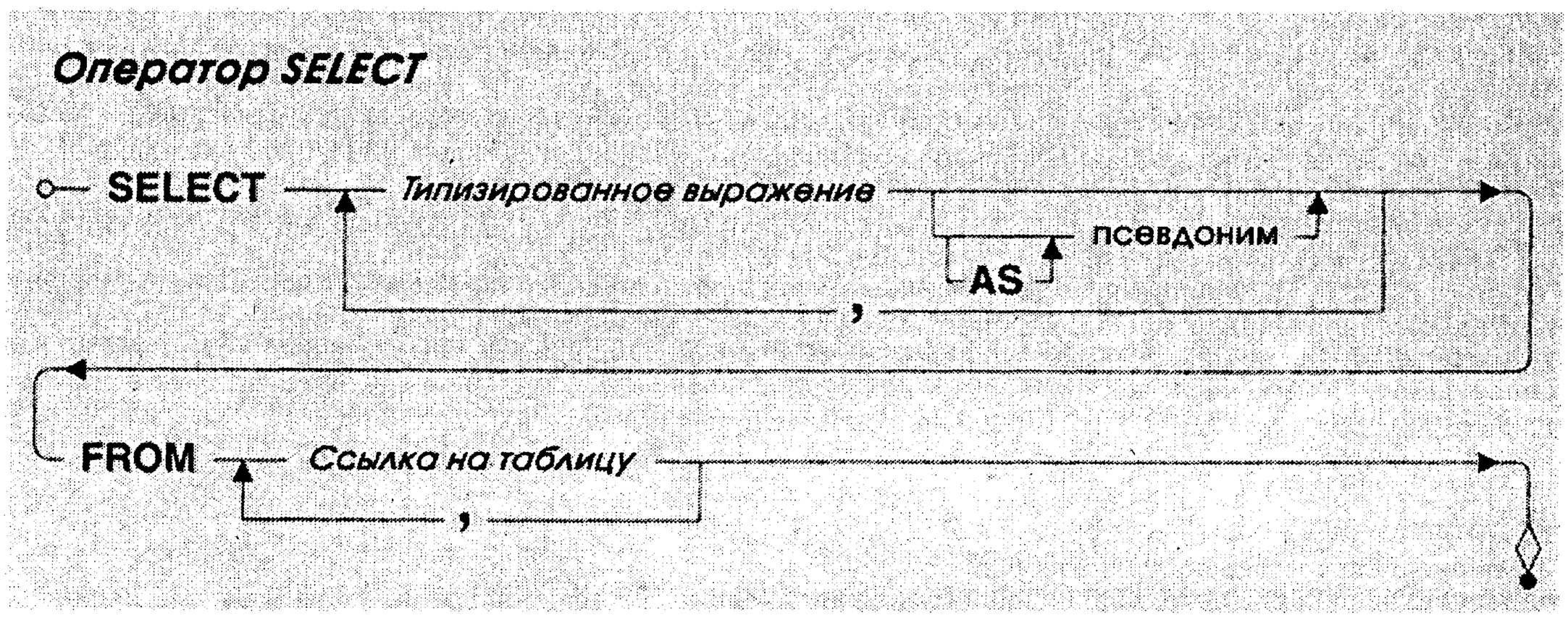


Рис. 11.1. *Простой оператор SELECT*

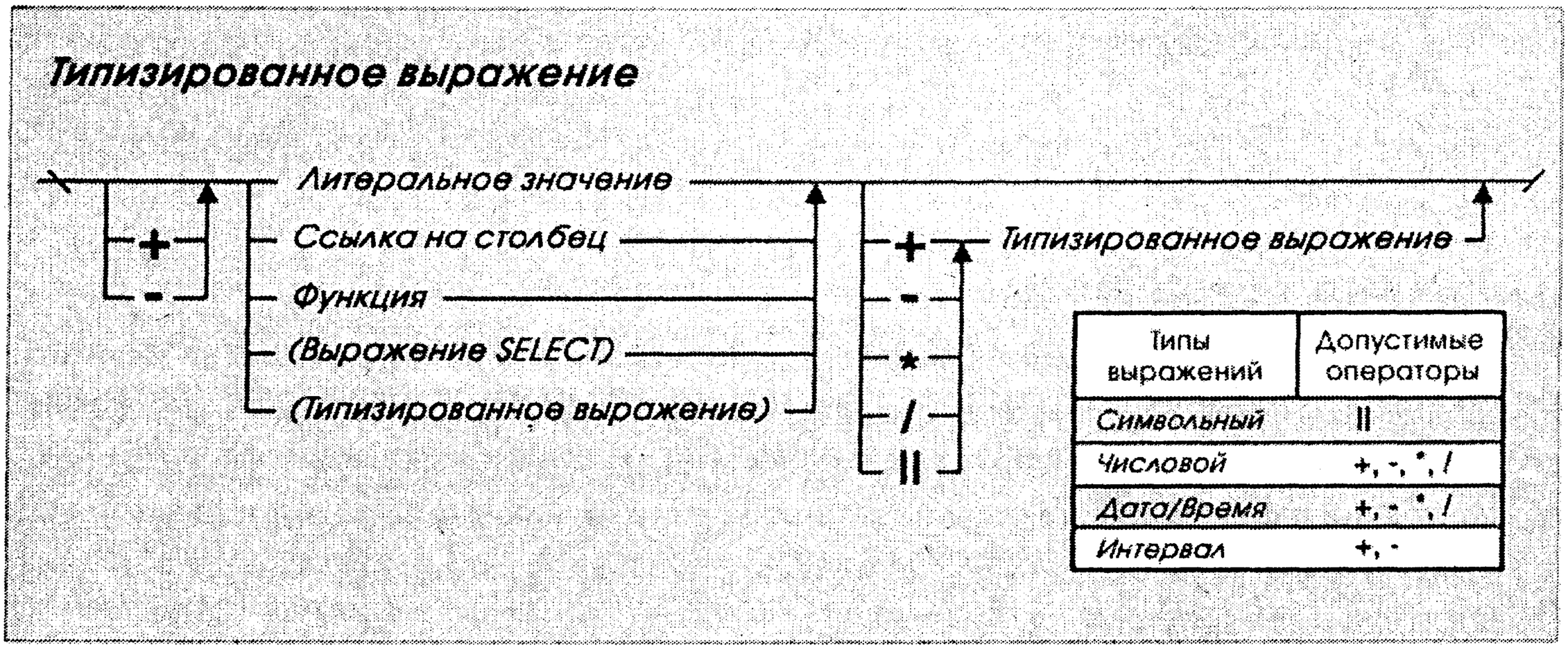


Рис. 11.2. *Диаграмма для типизированного выражения*

Это выглядит просто, но в действительности все не так! На самом деле *типизированное выражение* может быть достаточно сложным. На рис. 11.2 показаны все опции, которые могут составлять типизированное выражение.

Мы уже объясняли, как создать базовое типизированное выражение, используя значения типа "литерал", ссылки на столбцы и функции (см. главу 5). Обратите внимание, что теперь в списке появилось *выражение SELECT*. Это означает, что скалярный подзапрос можно вложить в список выражений сразу же после ключевого слова SELECT. Скалярный подзапрос является выражением SELECT, которое возвращает только один столбец и не более одной строки. Это имеет смысл, потому что осуществляется замена подзапроса, где обычно вводится имя отдельного столбца или выражения, дающего в результате один столбец.

Возможно, вы спросите, какая же от этого польза. Использование подзапроса таким способом позволяет "выдернуть" отдельное значение из некоторой другой таблицы или запроса, чтобы включить его в вывод для своего запроса. Таблица или запрос, которые являются источником данных в условии FROM подзапроса, совсем не требуют своего указания в условии FROM внешнего запроса. В большинстве случаев потребуется добавить критерий в условие WHERE подзапроса, чтобы гарантировать возвращение не более чем одной строки. Можно даже иметь критерий в подзапросе, указывающий значение, возвращаемое внешним запросом, чтобы выбрать данные, имеющие отношение к текущей строке.

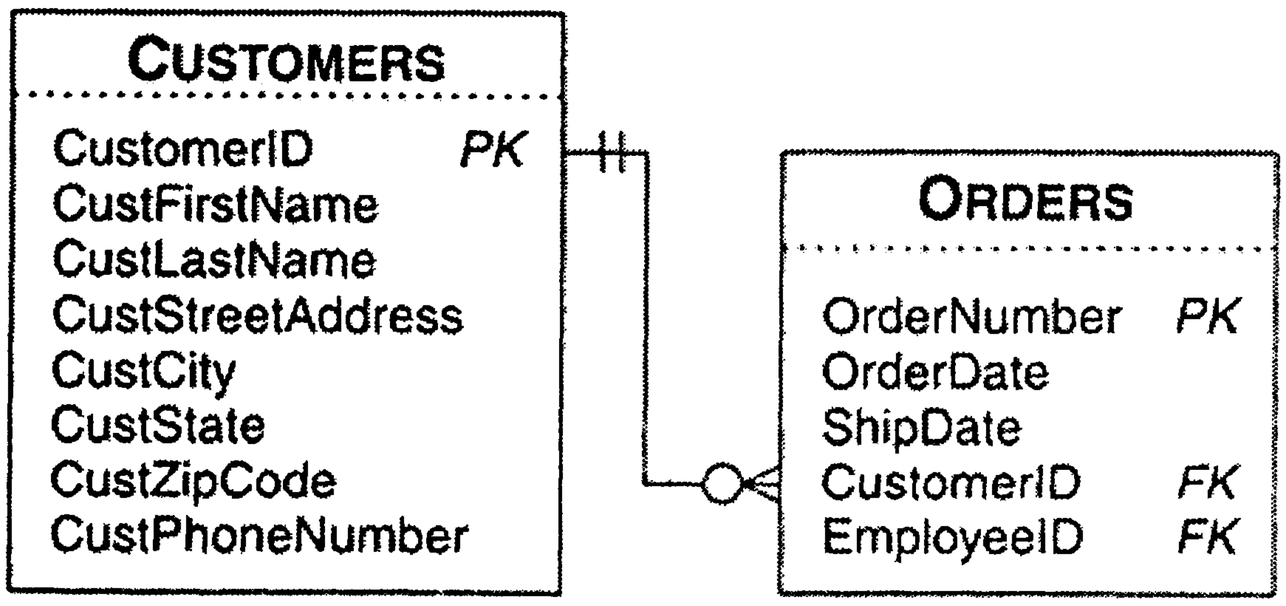


Рис. 11.3. *Таблицы Customers и Orders*

Ознакомимся с некоторыми простыми примерами, используя только таблицы Customers и Orders из учебной базы данных заказов. На рис. 11.3 показана связь между этими двумя таблицами.

Теперь построим запрос, который выдает список заказов для конкретной даты и выбирает фамилию соответствующего клиента из таблицы Customers, исподьзуя подзапрос.

Внимание! В данной главе будет использоваться метод "Запрос/Преобразование/Уточнение/SQL", введенный в главе 4. Кроме того, части, которые являются подзапросами, на шаге "Уточнение" заключаются в скобки и, там где это возможно, подзапросы выделяются отступом, чтобы привлечь внимание к тому, как они используются.

"Show me all the orders shipped on December 24, 1999 and the related customer last name".
("Показать все заказы, отгруженные на 24 декабря 1999 г., и фамилии соответствующих клиентов".)

Преобразование: Select order number, order date, shipped date,
and also select the related customer last name
out of the customers table from the orders table
where shipped date is December 24, 1999
(Выбрать номер заказа, дату заказа, дату отгрузки,
а также фамилию соответствующего клиента
из таблицы "Клиенты" из таблицы "Заказы",
где дата отгрузки — 24 декабря 1999)

Уточнение: Select order number, order date, shipped date,
~~and also~~ (select ~~the related~~ customer last name
~~out of the~~ from customers ~~table~~ from ~~the~~ orders ~~table~~)
where shipped date ~~is~~ = December 24, 1999
(Выбрать номер заказа, дату заказа, дату отгрузки
(выбрать фамилию клиента из "Клиенты" из таблицы
"Заказы"), где дата отгрузки = 24 декабря 1999)

SQL

```
SELECT Orders.OrderNumber, Orders.OrderDate,
    Orders.ShippedDate,
    (SELECT Customers.CustLastName
    FROM Customers
    WHERE Customers.CustomerID = Orders.CustomerID)
FROM Orders
WHERE Orders.ShippedDate = '1999-12-24'
```

Нам потребовалось ограничить значение идентификатора клиента (Customer ID) в подзапросе значением идентификатора клиента в каждой строке, извлекаемой из таблицы Orders. В ином случае в подзапросе будут получены *все* строки из

Customers. Помните, что это должен быть скалярный подзапрос, поэтому нужно что-то предпринять для ограничения результата, чтобы возвращалось не более одной строки.

Вспомнив INNER JOIN, вы можете спросить, почему бы не решить эту задачу так, как уже описано, вместо того чтобы объединять Orders в JOIN с Customers в условии FROM внешнего запроса. На самом деле именно сейчас наше внимание сосредоточивается на *концепции* использования подзапросов для создания столбца вывода в очень простом примере, и действительно, возможно, следует решить эту конкретную задачу, используя INNER JOIN:

```
SQL                SELECT Orders.OrderNumber, Orders.OrderDate,
                      Orders.ShippedDate, Customers.CustLastName
                   FROM Customers INNER JOIN Orders
                   ON Customers.CustomerID = Orders.OrderID
                   WHERE Orders.ShippedDate = '1999-12-24'
```

Введение в агрегатные функции — COUNT и MAX

Теперь расширим наши горизонты и посмотрим, какую реальную пользу может принести эта возможность. В первую очередь нужно дать обзор нескольких агрегатных функций (подробнее о них см. в следующей главе).

Стандарт SQL определяет множество функций, которые вычисляют значения в запросе. Один из подклассов функций — агрегатные функции — позволяет вычислять одно значение для группы строк в наборе результатов. Например, агрегатной функцией можно воспользоваться для подсчета строк, поиска наибольшего или наименьшего значения для некоторого множества строк или для вычисления среднего или общего значения или выражения по набору результата.

Рассмотрим две из этих функций. На рис. 11.4 показана диаграмма для функций COUNT и MAX, которые могут сформировать столбец вывода в условии SELECT.

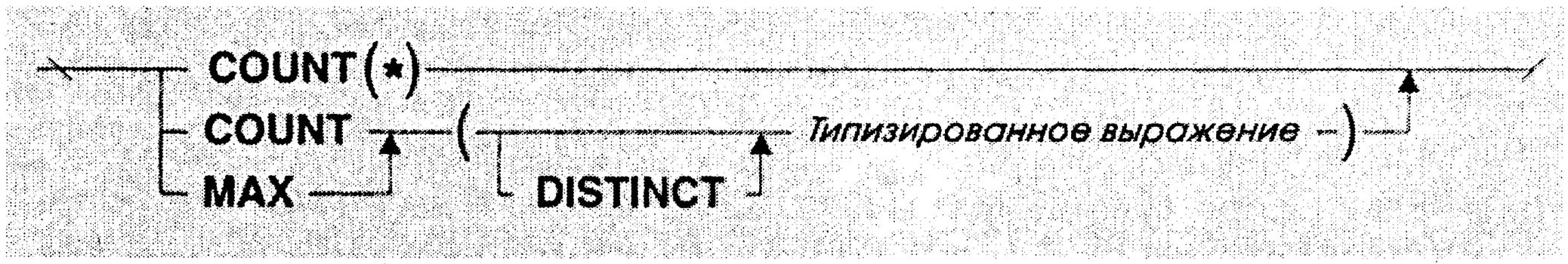


Рис. 11.4. *Использование агрегатных функций COUNT и MAX*

Функция COUNT может использоваться для определения количества строк или отличных от Null значений в наборе результата. COUNT(*) используется для определения количества строк во всем множестве. Если в наборе результатов конкретный столбец определяется с использованием COUNT (column_name), то СУБД высчитывает количество строк, в которых значение этого столбца не равно Null. Можно также запросить подсчитать только уникальные значения, добавив ключевое слово DISTINCT.

Подобным образом можно найти наибольшее значение в столбце, используя функцию MAX. Если типизированное выражение является числовым, то наибольшее число будет получено из определенного вами столбца или выражения. Если же типизированное выражение возвращает данные символьного типа, то наибольшее значение будет зависеть от сортирующей последовательности, используемой системой базы данных.

Попробуем применить эти функции в подзапросе для решения нескольких интересных задач.

"List all the customer names and a count of the orders they placed". ("Привести список имен всех клиентов и количество размещенных ими заказов".)

Преобразование: Select customer first name, and customer last name, and also select the count of orders from the orders table for this customer from the customers table
(Выбрать имя и фамилию клиента, а также подсчитать количество заказов из таблицы "Заказы" для этого клиента из таблицы "Клиенты")

Уточнение: Select customer first name, ~~and~~ customer last name, ~~and also~~ (select ~~the~~ count ~~of orders~~ (*) from ~~the~~ orders ~~table for this~~ where customerID = customerID) from ~~the~~ customers ~~table~~
(Выбрать имя клиента, фамилию клиента
(Выбрать количество (*) из "Заказы",
где идентификатор клиента = customerID)
из "Клиенты")

SQL

```
SELECT Customers.CustFirstName,
    Customers.CustLastName,
    (SELECT COUNT(*)
    FROM Orders
    WHERE Orders.CustomerID =
        Customers.CustomerID)
AS CountOfOrders
FROM Customers
```

Подзапросы как выходные столбцы начинают теперь выглядеть интересно! Из следующих глав вы узнаете больше о творческих подходах к использованию агрегатных функций. Но, если требуется только подсчет соответствующих строк, то подзапрос является хорошим способом. Теперь рассмотрим интересную задачу, которая использует преимущества другой агрегатной функции — MAX.

"Show me a list of customers and the last date on which they placed an order".
("Показать список клиентов и последнюю дату размещения ими заказа".)

Преобразование: Select customer first name, and customer last name, and also select the highest order date from the orders table for this customer from the customers table
(Выбрать имя и фамилию клиента, а также выбрать последнюю дату заказа из таблицы "Заказы" для этого клиента из таблицы "Клиенты")

Уточнение: Select customer first name, ~~and~~ customer last name, ~~and also~~ (select ~~the highest~~ max(order date) from ~~the~~ orders ~~table for this~~ where customerID = customerID) from ~~the~~ customers table
(Выбрать имя клиента, фамилию клиента и (Выбрать max(дата заказа) из "Заказы", где идентификатор клиента = customerID) из "Клиенты")

SQL
```
SELECT Customers.CustFirstName,
    Customers.CustLastName, (SELECT MAX(OrderDate)
    FROM Orders
    WHERE Orders.CustomerID = Customers.CustomerID)
AS LastOrderDate
FROM Customers
```

Использование функции MAX таким образом хорошо работает для поиска наибольших или новейших значений из любых связанных таблиц.

Подзапросы как фильтры

Вы уже знаете, как отфильтровать извлеченную информацию путем добавления условия WHERE и как использовать сложные и простые сравнения для получения в наборе результатов только нужных строк. Теперь мы покажем, как выполнить усложненную фильтрацию, используя подзапрос как один из аргументов сравнения.

Синтаксис

Вернемся к предыдущему оператору SELECT и рассмотрим синтаксис построения запроса с простым предикатом сравнения в условии WHERE. На рис. 11.5 представлена упрощенная диаграмма.

На рис. 11.2 показано, что типизированное выражение может быть подзапросом. В простом примере на рис. 11.5 типизированное выражение сравнивается с отдельным столбцом. Таким образом, типизированное выражение должно быть одиночным значением, т. е. скалярным подзапросом, который возвращает точно один столбец

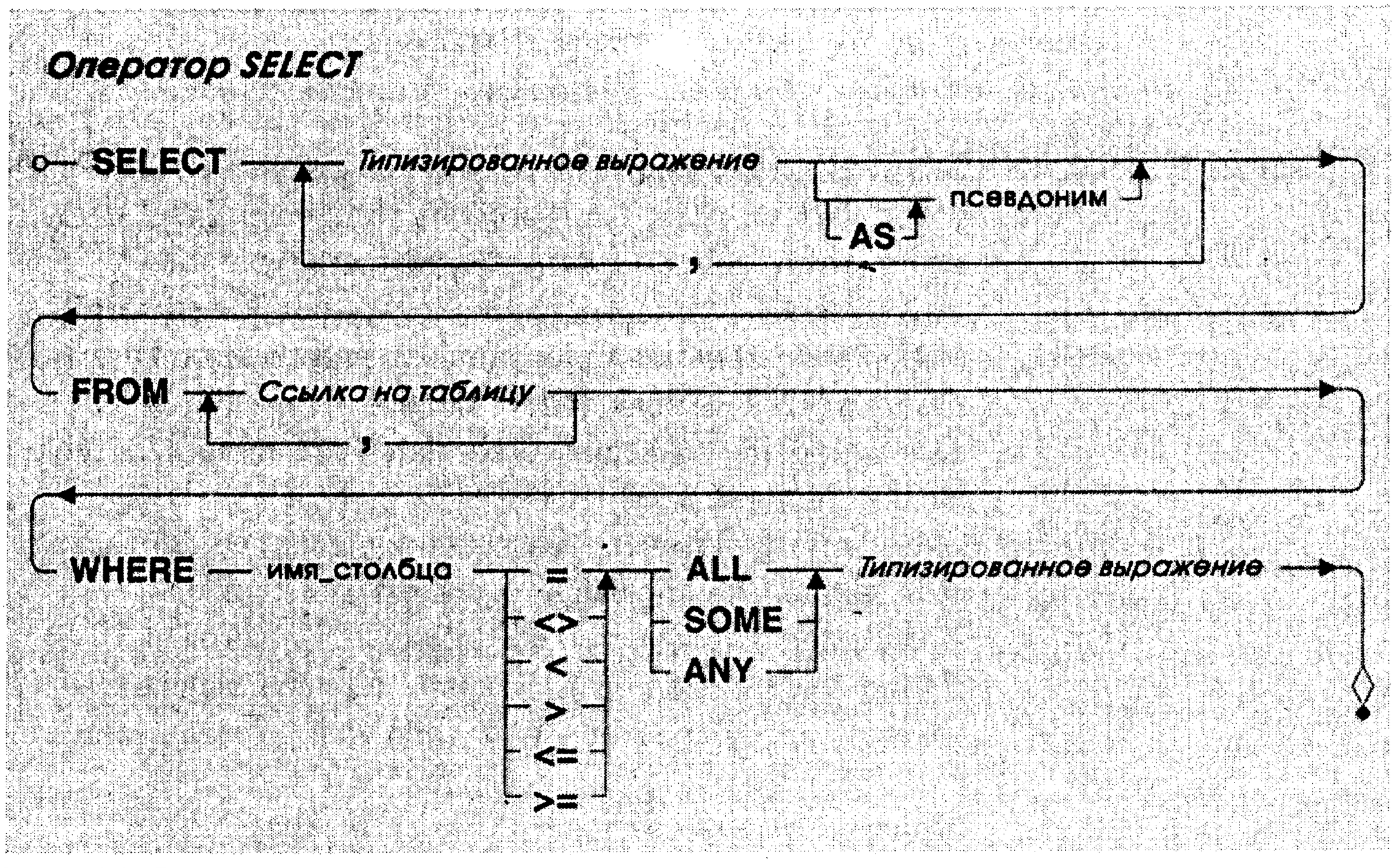


Рис. 11.5. *Фильтрация результата с использованием простого предиката сравнения*

и не более одной строки. Решим простую задачу, требующую сравнения со значением, возвращенным из подзапроса. В этом примере предполагается запросить все детали о заказах клиентов, но нас интересует только *последний* заказ каждого клиента. На рис. 11.6 показаны необходимые для этого таблицы.

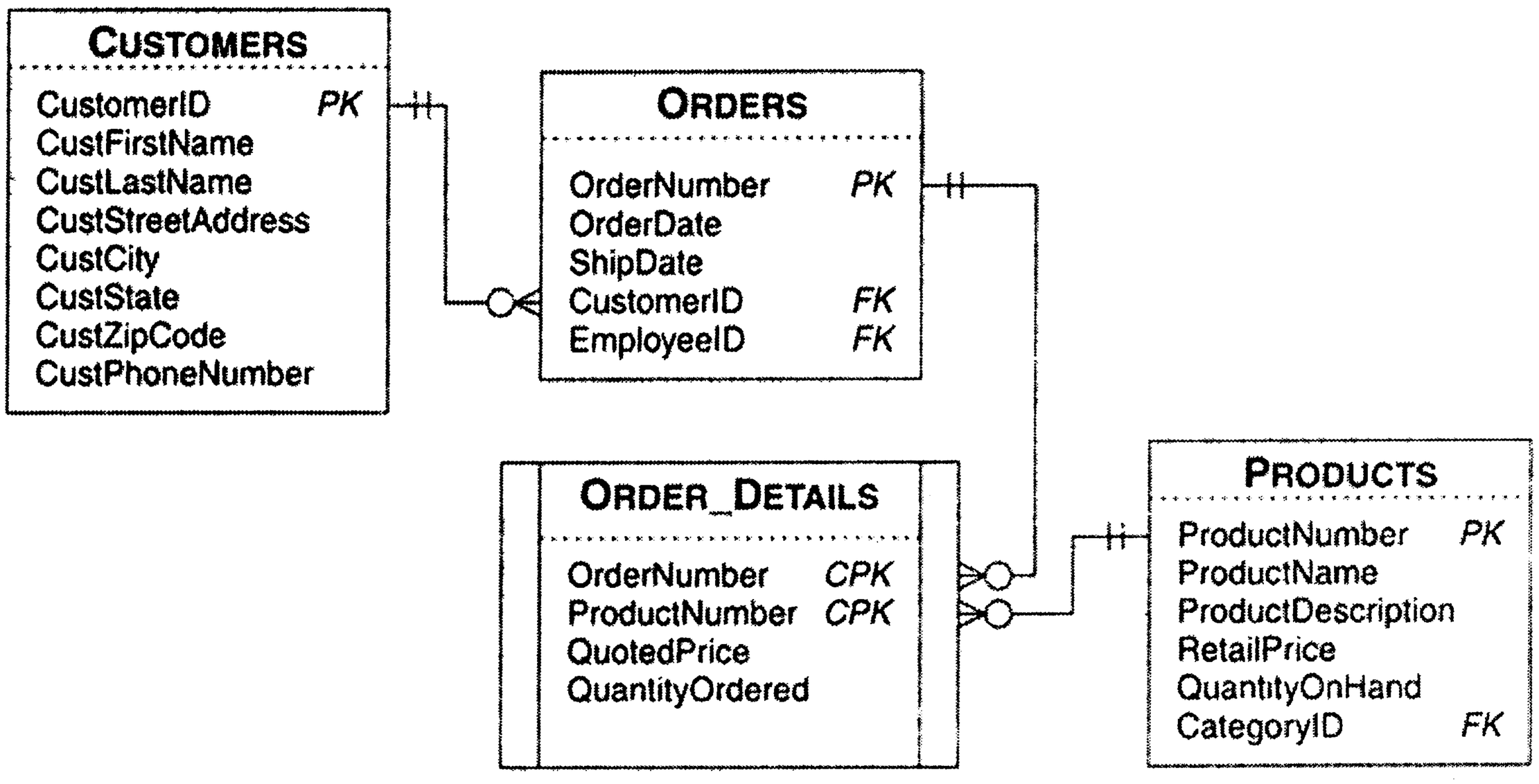


Рис. 11.6. *Таблицы, необходимые для подготовки списка всех подробностей о заказе*

"List customers and all the details from their last order".
("Показать список клиентов и все детали их последнего заказа".)

Преобразование: Select customer first name, customer last name, order number, order date, product number, product name, and quantity ordered from the customers table joined with the orders table on customer ID, then joined with the order details table on order number, and then joined with the products table on product number where the order date equals the maximum order date from the orders table for this customer
(Выбрать имя клиента, фамилию клиента, номер заказа, дату заказа, номер товара, наименование товара и заказанное количество из таблицы "Клиенты", соединенной с таблицей "Заказы" по идентификатору клиента, затем соединенной с таблицей "Детали заказа" по номеру заказа, затем соединенной с таблицей "Товары" по номеру товара, где дата заказа равна максимальной дате заказа из таблицы "Заказы" для этого клиента)

Уточнение: Select customer first name, customer last name, order number, order date, product number, product name, ~~and~~ quantity ordered from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~then~~ join~~ed with the~~ order details ~~table~~ on order number, ~~and then~~ join~~ed with the~~ products ~~table~~ on product number where the order date ~~equals~~ = (Select ~~the~~ max~~imum~~ (order date) from ~~the~~ orders ~~table for this customer~~ where orders.customerID = customers.customerID)
(Выбрать имя клиента, фамилию клиента, номер заказа, дату заказа, номер товара, наименование товара, заказанное количество из "Клиенты", соединенной с "Заказы" по идентификатору клиента, соединенной с "Детали заказа" по номеру заказа, соединенной с "Товары" по номеру товара где дата заказа = (Выбрать MAX(дата заказа) из "Заказы", где orders.customerID = customers.customerID))

SQL

```
SELECT Customers.CustFirstName,
    Customers.CustLastName, Orders.OrderNumber,
    Orders.OrderDate, Order_Details.ProductNumber,
    Products.ProductName,
    Order_Details.QuantityOrdered
```

```
FROM ((Customers
INNER JOIN Orders
ON  Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON  Orders.OrderID = Order_Details.OrderID)
INNER JOIN Products
ON  Products.ProductNumber =
    Order_Details.ProductNumber
WHERE Orders.OrderDate =
    (SELECT MAX(OrderDate)
    FROM Orders AS O2
    WHERE O2.CustomerID = Customers.CustomerID)
```

Заметили ли вы, что второй ссылке на таблицу Orders было присвоено имя-псевдоним (т. е. таблице Orders в подзапросе)? Даже если имя-псевдоним пропущено, многие системы БД распознают, что подразумевается копия таблицы Orders в подзапросе. Фактически стандарт SQL указывает, что любая неуточненная ссылка должна быть разрешена, начиная с самого внутреннего запроса. Тем не менее мы добавили псевдоним-ссылку с целью сделать совершенно понятным, что копия таблицы Orders, на которую мы ссылаемся в условии WHERE подзапроса, является той же таблицей, что и в условии FROM подзапроса. Если соблюдать эту практику, то будет намного легче понять запрос и вам самим, в случае возвращения к нему несколько месяцев спустя, и кому-либо другому.

Специальные ключевые слова предиката для подзапросов

Стандарт SQL определяет несколько специальных ключевых слов предиката для использования в условии WHERE с подзапросом.

Принадлежность к множеству — IN

Ключевое слово IN используется в условии WHERE для сравнения столбца или выражения со списком значений. Каждое типизированное выражение в списке IN *может* быть скалярным подзапросом. А как насчет использования подзапроса для формирования всего списка? Как показано на рис. 11.7, это вполне можно сделать.

В этом случае можно использовать табличный подзапрос, который возвращает один столбец и столько строк, сколько необходимо для построения списка. Воспользуемся, например, базой данных Recipes (Рецепты). На рис. 11.8 показаны вовлеченные в это таблицы.

Предположим, что к обеду вы пригласили кого-то, кто обожает блюда из морепродуктов, Хотя у вас есть множество рецептов, включающих компоненты даров моря, вы не уверены в названиях всех компонентов в вашей базе данных. Вам точно известно, что в IngredientClassDescription имеются Seafood (Морепродукты), поэтому можно соединить все таблицы вместе и отфильтровать по IngredientClassDescription, или можно подойти творчески и воспользоваться вместо этого подзапросом и предикатом IN.

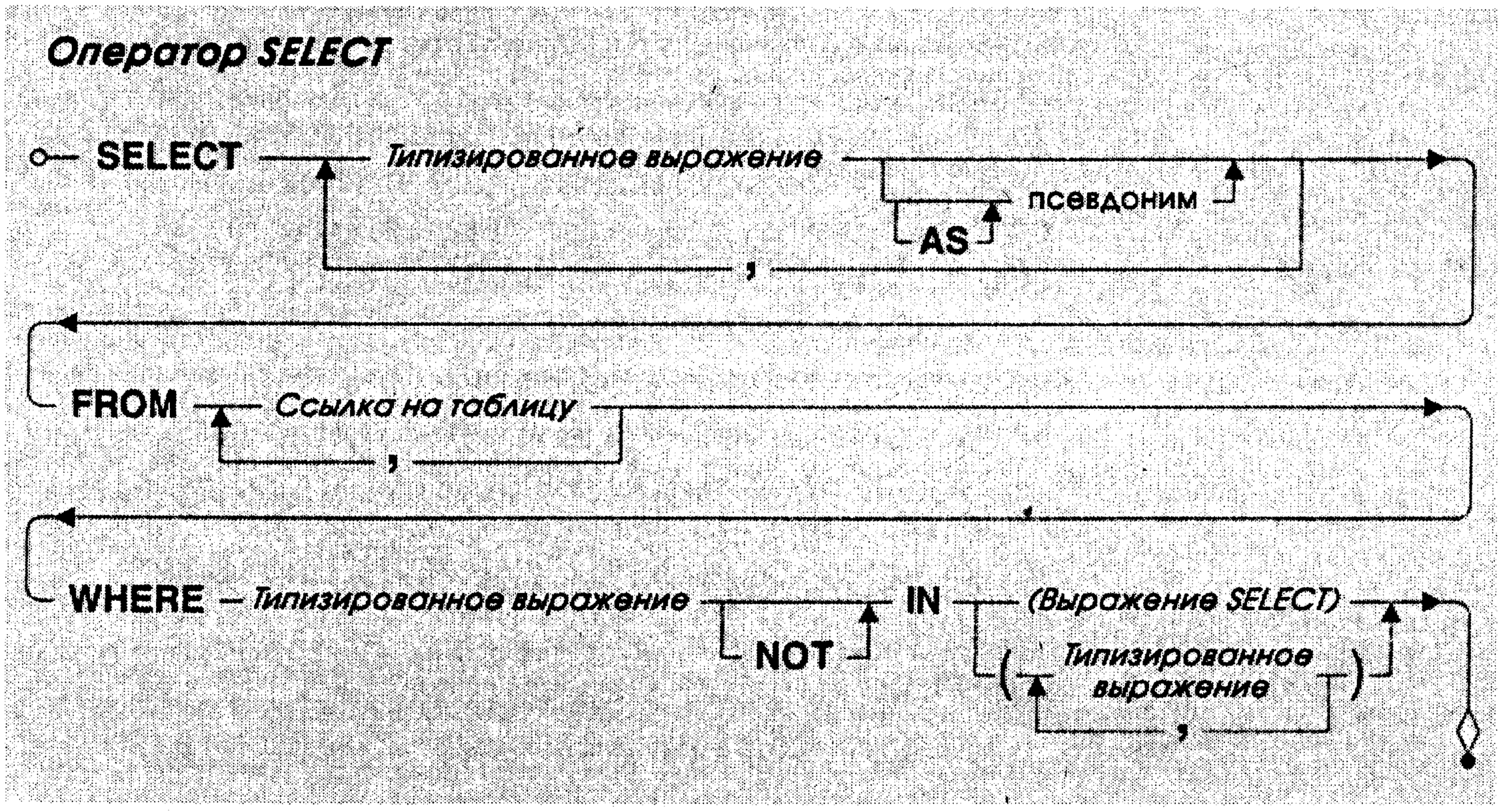


Рис. 11.7. *Использование подзапроса в предикате IN*

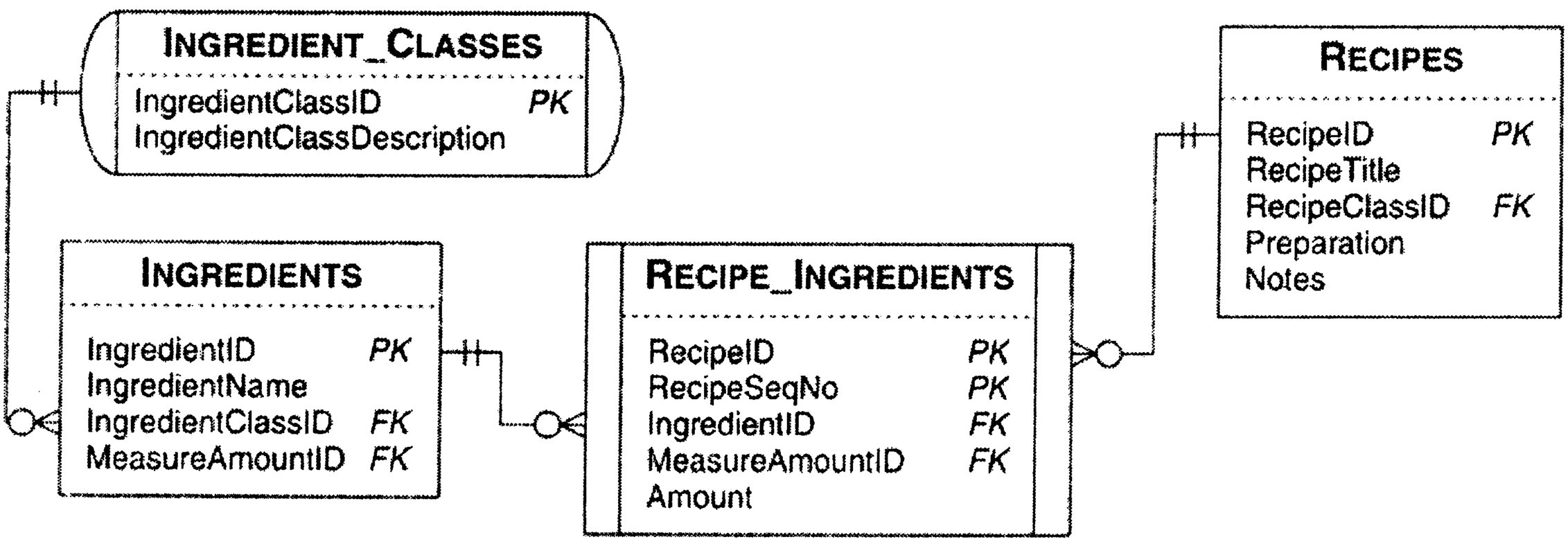


Рис. 11.8. *Таблицы, используемые для вывода рецептов и их компонентов*

"List all my recipes that have a Seafood ingredient".
("Предоставить список рецептов, содержащих компоненты из морепродуктов".)

Преобразование: Select recipe title from the recipes table where the recipe ID is in the selection of recipe IDs from the recipe ingredients table where the ingredient ID is in the selection of ingredient IDs from the ingredients table joined with the ingredient classes table on ingredient class ID where ingredient class description is "seafood"
(Выбрать заголовок рецепта из таблицы "Рецепты", где идентификатор рецепта выбирается в идентификаторах

рецептов из таблицы “Компоненты рецепта”, где идентификатор компонента выбирается в идентификаторах компонентов из таблицы “Компоненты”, соединенной с таблицей “Классы компонентов” по идентификатору класса компонента, где описание класса компонента — “морепродукты”)

Уточнение:

Select recipe title from ~~the~~ recipes ~~table~~ where ~~the~~ recipe ID ~~is~~ in ~~the~~ (select~~ion of~~ recipe ID~~s~~ from ~~the~~ recipe ingredients ~~table~~ where ~~the~~ ingredient ID ~~is~~ in ~~the~~ (select~~ion of~~ ingredient ID~~s~~ from ~~the~~ ingredients ~~table~~ join~~ed with the~~ ingredient classes table on ingredient class ID where ingredient class description ~~is~~ = “seafood”))
(Выбрать заголовок рецепта из “Рецепты”, где идентификатор рецепта в (Выбрать идентификатор рецептов из “Компоненты рецепта”, где идентификатор компонента (Выбрать идентификатор компонента из “Компоненты”, соединенной с “Классы компонентов” по идентификатору класса компонента, где описание класса компонента = “морепродукты”))

SQL

```
SELECT RecipeTitle
FROM Recipes
WHERE Recipes.RecipeID IN
    (SELECT RecipeID
    FROM Recipe_Ingredients
    WHERE Recipe_Ingredients.IngredientID IN
        (SELECT IngredientID
        FROM Ingredients
        INNER JOIN Ingredient_Classes
        ON  Ingredients.IngredientClassID =
            Ingredient_Classes.IngredientClassID
        WHERE
        Ingredient_Classes.IngredientClassDescription =
        'Seafood'))
```

Приходило ли вам на ум, что можно поместить подзапрос внутрь подзапроса? Мы действительно можем переместиться на уровень глубже, исключив INNER JOIN из второго подзапроса. Второй подзапрос можно записать, воспользовавшись следующим синтаксисом:

SQL

```
. . CSELECT IngredientID
FROM Ingredients
WHERE Ingredients.IngredientClassID IN
    (SELECT IngredientClassID
```

```
FROM Ingredient_Classes
WHERE
Ingredient_Classes.IngredientClassDescnption =
    'Seafood'))
```

Однако это было бы избыточным, потому что вложение условий IN в условия IN только затрудняет чтение запроса. Мы это сделали просто для того, чтобы показать, что это *возможно*. Тем не менее стоит повторно заявить, что *возможность* сделать что-то не означает *необходимость* это делать! Надеемся, вы согласитесь, что легче всего увидеть что произойдет, используя в подзапросе единственный предикат IN и более сложный оператор JOIN. Вот еще одно решение с использованием этого метода:

SQL

```
SELECT RecipeTitle
FROM Recipes
WHERE Recipes.RecipeID IN
    (SELECT RecipeID
    FROM (Recipe_Ingredients
    INNER JOIN Ingredients
    ON  Recipe_Ingredients.IngredientID =
        Ingredients.IngredientID)
INNER JOIN Ingredient_Classes
ON Ingredients.IngredientClassID =
    Ingredient_Classes.IngredientClassID
WHERE
Ingredient_Classes.IngredientClassDescription =
    'Seafood')
```

Зачем преодолевать все эти трудности? Почему просто не выполнить сложное соединение во внешнем запросе и закончить с этим? Причина в том, что будет получен неверный ответ! В действительности все возвращенные строки будут строками для рецептов с морепродуктами из таблицы “Рецепты”, но можно получить некоторые строки более одного раза. Попробуем решить эту задачу, не используя подзапрос, чтобы увидеть, почему будут получены повторяющиеся строки:

SQL

```
SELECT RecipeTitle
FROM ((Recipes
INNER JOIN Recipe_Ingredients
ON  Recipes.RecipeID = Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON  Recipe_Ingredients.IngredientID =
    Ingredients.IngredientID)
INNER JOIN Ingredient_Classes
ON  Ingredients.IngredientClassID =
Ingredient_Classes.IngredientClassID
```

```
WHERE
Ingredient_Classes.IngredientClassDescription =
    'Seafood')
```

На рис. 11.8 можно увидеть, что в таблице Recipe_Ingredients может быть несколько строк для каждой строки в таблице Recipes. Набор результатов, определенный условием FROM, будет содержать по крайней мере столько строк, сколько строк в Recipe_Ingredients, со значением столбца RecipeTitle, повторенным много раз. Даже когда добавляется фильтр для ограничения результата компонентами в классе Seafood, все еще будет получено более одной строки на рецепты в любом рецепте, который содержит более одного компонента морепродуктов.

Использование этой методики подзапроса действительно важно, когда требуется составить список более чем из одних заголовков рецептов. Предположим, что нужен также список *всех* компонентов из *каждого* рецепта, содержащего компонент морепродуктов. Если воспользоваться сложным соединением во внешнем запросе и отфильтровать класс компонентов Seafood, как это было сделано выше, то мы получим только компоненты морепродуктов, но не получим всех остальных компонентов рецепта. Зададим один дополнительный и немного более сложный запрос:

"List all my recipes and all ingredients for recipes that have a Seafood ingredient".
("Привести список всех рецептов и всех компонентов для рецептов, содержащих компонент морепродуктов".)

Преобразование: Select recipe title and ingredient name from the recipes table joined with the recipe_ingredients table on recipe ID, and then joined with the ingredients table on ingredient ID where the recipe ID is in the selection of recipe IDs from the recipe_ingredients table joined with the ingredients table on ingredient ID, and then joined with the ingredient_classes table on ingredient class ID where ingredient class description is "seafood"
(Выбрать заголовок рецепта и название компонента из таблицы "Рецепты", соединенной с таблицей recipe_ingredients по идентификатору рецепта, а затем соединенной с таблицей "Компоненты" по идентификатору компонента, где идентификатор рецепта находится в выборке идентификаторов рецептов из таблицы recipe_ingredients, соединенной с таблицей "Компоненты" по идентификатору компонента, а затем соединенной с таблицей ingredient_classes по идентификатору вида компонента, где описание вида компонента — "морепродукты")

Уточнение:

Select recipe title, ~~and~~ ingredient name from ~~the~~ recipes ~~table~~ join~~ed with the~~ recipe_ingredients ~~table~~ on recipe ID, ~~and then~~ join~~ed with the~~ ingredients ~~table~~ on ingredient ID where ~~the~~ recipe ID ~~is~~ in ~~the~~ (select~~ion of~~ recipe ID~~s~~ from ~~the~~ recipe_ingredients ~~table~~ join~~ed with the~~ ingredients ~~table~~ on ingredient ID, ~~and then~~ join~~ed with the~~ ingredient_classes ~~table~~ on ingredient class ID where ingredient class description ~~is~~ = “seafood”)

(Выбрать заголовок рецепта, название компонента
из “Рецепты”, соединенной с recipe_ingredients
по идентификатору рецепта, соединенной с “Компоненты”
по идентификатору компонента, где идентификатор
рецепта в (Выбрать идентификаторы рецепта
из recipe_ingredients, соединенной с “Компоненты”
по идентификатору компонента, соединенной
с ingredient_classes по идентификатору вида компонента,
где описание вида компонента = “морепродукты”))

SQL

```
SELECT Recipes.RecipeTitle,
    Ingredients.IngredientName
FROM (Recipes
INNER JOIN Recipe_Ingredients
ON  Recipes.RecipeID = Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID
WHERE Recipes.RecipeID IN
    (SELECT RecipeID
    FROM (Recipe_Ingredients
    INNER JOIN Ingredients
    ON  Recipe_Ingredients.IngredientID =
        Ingredients.IngredientID)
    INNER JOIN Ingredient_Classes
    ON  Ingredients.IngredientClassID =
    Ingredient_Classes.IngredientClassID
WHERE
Ingredient_Classes.IngredientClassDescription =
    'Seafood')
```

Основной момент здесь состоит в том, что сложное внешнее соединение в основной части запроса извлекает *все* компоненты выбранных рецептов, а сложный подзапрос возвращает список идентификаторов рецепта только для рецептов с морепродуктами. Это выглядит так, будто мы дважды выполняем сложное соединение, но именно в этом безрассудстве и состоит метод.

Определяя количество — ALL/SOME/ANY

Предикат IN позволяет сравнивать столбец или выражение со списком, чтобы определить, содержится ли этот столбец или выражение в списке (IN). Другими словами, столбец или выражение *равно* одному из элементов списка. Если нужно установить, является ли столбец или выражение больше или меньше, чем любое, все или некоторые из элементов в списке, то можно использовать *количественный* предикат. На рис. 11.9 представлен синтаксис.

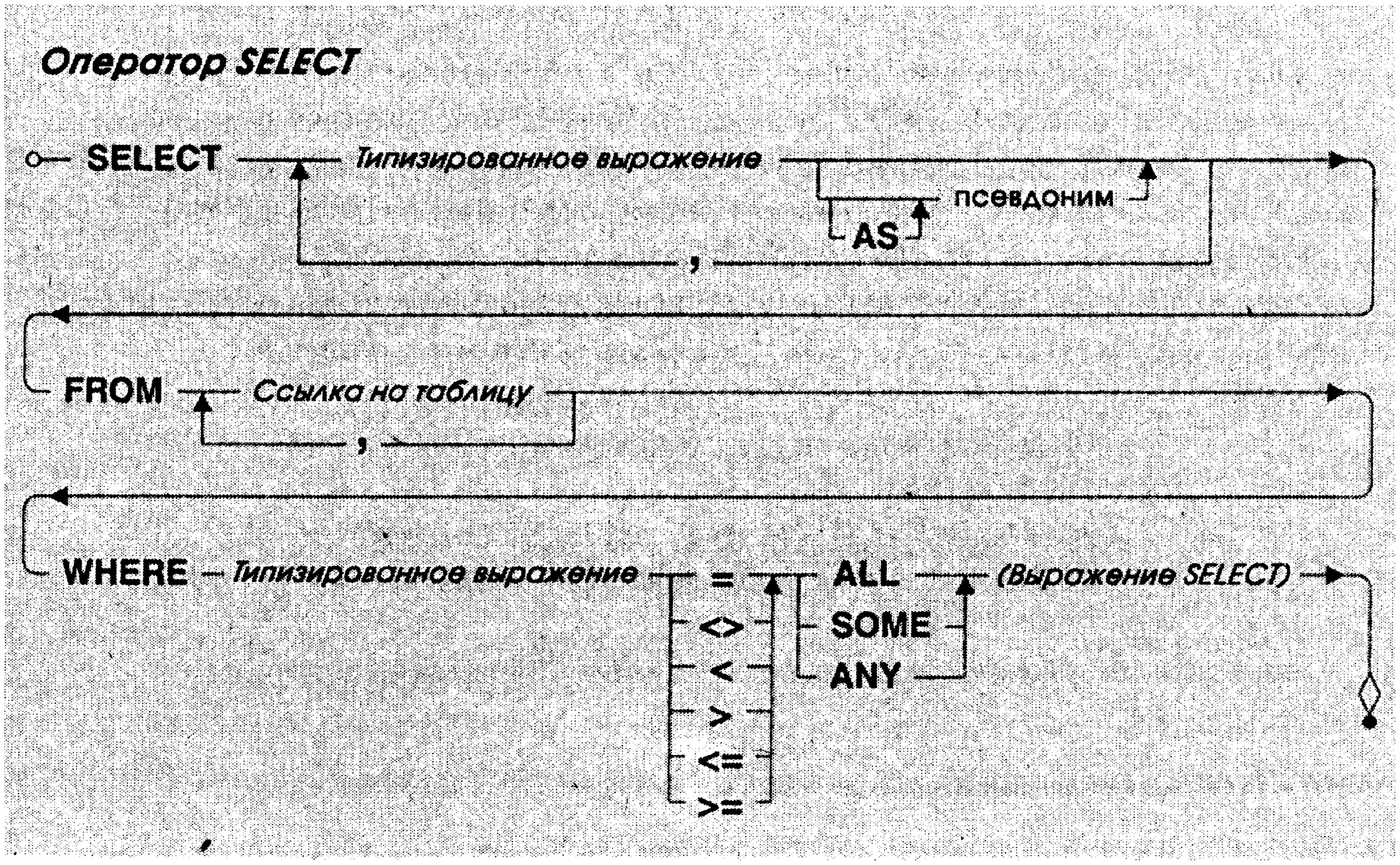


Рис. 11.9. *Использование количественного предиката в операторе SELECT*

В данном случае выражение SELECT должно быть табличным подзапросом, который возвращает только один столбец и ни одной или несколько строк. Когда подзапрос возвращает больше одной строки, значения в строках составляют список. Как можно видеть, этот предикат объединяет оператор сравнения с ключевым словом, которое указывает системе базы данных, как этот оператор применяется к элементам списка. При использовании ключевого слова ALL результат сравнения должен иметь значение True для всех значений, возвращенных подзапросом. При использовании ключевого слова SOME или ANY результат сравнения должен быть True хотя бы для одного значения в списке.

Когда подзапрос возвращает несколько строк, запрос для = ALL всегда будет False, если только все значения, возвращенные подзапросом, не являются одинаковыми и типизированное выражение в левой части сравнения равно каждому из них. Согласно этой же логике можно подумать, что < > ANY всегда будет False, если типизированное выражение в левой части *всегда* равно любому значению в списке.

На самом деле стандарт SQL интерпретирует SOME и ANY как одно и то же. Поэтому, если указать < > SOME или < > ANY, этот предикат будет True, если типизированное выражение слева не равно по крайней мере одному из значений в списке. Другим смущающим моментом является то, что если подзапрос не возвращает строк, то любой предикат сравнения с ключевым словом ALL является True и любой предикат сравнения с ключевыми словами SOME или ANY является False.

Внимание! Стандарт SQL определяет концепцию *конструктора строкового значения*. Если ваша база данных поддерживает эту часть стандарта, то выражение SELECT в количественном предикате может возвратить более одного столбца. В этом случае элемент в левой части сравнения должен представлять список типизированных выражений, разделенный запятыми и заключенный в скобки. Также выражение SELECT должно возвратить несколько столбцов, равных по количеству числу типизированных выражений слева. Например, можно построить оператор SQL, подобный следующему:

```
SELECT *
FROM MyTable
WHERE (MyTable.Column1, MyTable.Column2) > ALL
(SELECT ColumnA, ColumnB FROM OtherTable)
```

Проверьте по документации для базы данных, поддерживает ли она этот синтаксис.

Рассмотрим пару запросов, чтобы увидеть использование количественных предикатов. Вначале решим задачу в базе данных Recipes. Обратитесь к рис. 11.8, на котором представлены нужные нам таблицы.

"Show me the recipes that have beef or garlic".
("Показать рецепты, содержащие говядину и чеснок".)

Преобразование: Select recipe title from the recipes table where recipe ID is in the selection of recipe IDs from the recipe ingredients table where ingredient ID equals any of the selection of ingredient IDs from the ingredients table where ingredient name is "beef" or "garlic"
(Выбрать заголовок рецепта из таблицы "Рецепты", где идентификатор рецепта находится в выборке идентификаторов рецептов из таблицы "Компоненты рецептов", где идентификатор компонента равен любой выборке идентификаторов компонента из таблицы "Компоненты", где название компонента — "beef" или "garlic")

Уточнение: Select recipe title from ~~the~~ recipes ~~table~~ where recipe ID ~~is~~ in ~~the~~ (select~~ion of~~ recipe ID~~s~~ from ~~the~~ recipe ingredients ~~table~~ where ingredient ID ~~equals~~ = any ~~of the~~ (select~~ion of~~ ingredient ID~~s~~ from ~~the~~ ingredients ~~table~~ where ingredient name ~~is~~ “beef” ~~or~~ “garlic”))
(Выбрать заголовок рецепта из “Рецепты”, где идентификатор рецепта в (Выбрать идентификатор рецепта из “Компоненты рецептов”, где идентификатор компонента = любой (Выбрать идентификатор компонента из “Компоненты”, где название компонента — “beef”, “garlic””))

SQL
```
SELECT Recipes.RecipeTitle
FROM Recipes
WHERE Recipes.RecipeID IN
    (SELECT Recipe_Ingredients.RecipeID
    FROM Recipe_Ingredients
    WHERE Recipe_Ingredients.IngredientID = ANY
        (SELECT Ingredients.IngredientID
        FROM Ingredients
        WHERE Ingredients.IngredientName
        IN ('Beef, 'Garlic')))
```

Не возникло ли у вас ощущения, что можно также использовать IN вместо = ANY? Если возникло, то вы правы! Можно также создать JOIN между Recipe_Ingredients и Ingredients в первом подзапросе, чтобы возвратить нужный список идентификаторов рецептов RecipeID. В SQL всегда существует более одного способа решения конкретной задачи. Иногда использование количественного предиката может сделать запрос более понятным.

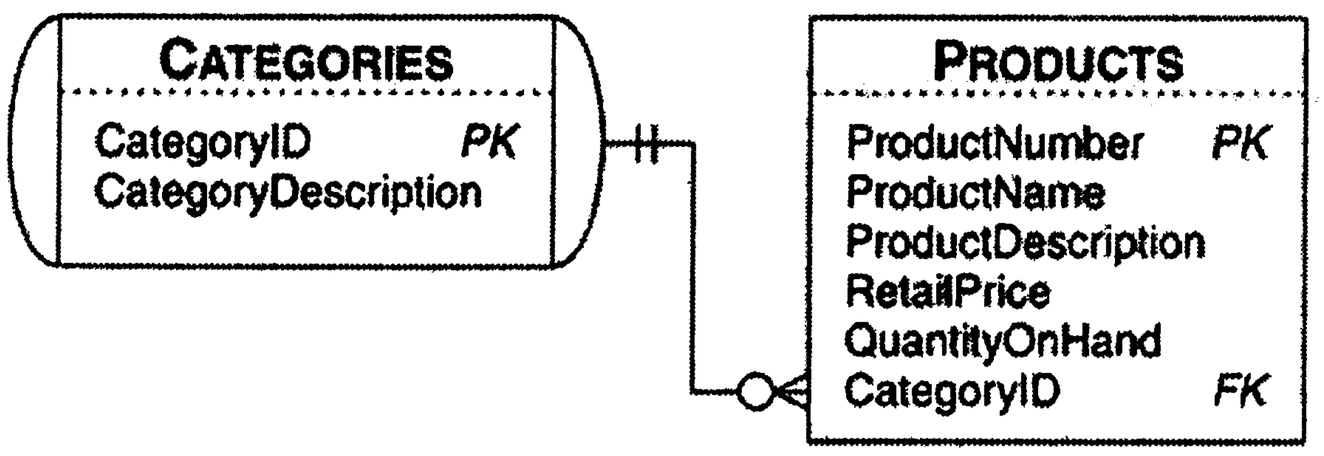


Рис. 11.10. *Таблица Categories и связанная с ней таблица Products*

Теперь решим немного более сложную задачу, чтобы показать реальную мощность количественных предикатов. В этом примере используется учебная база данных Sales Order (Заказы на закупку). На рис. 11.10 представлены используемые для этого таблицы.

“Find all accessories that are priced greater than any clothing item”.
(“Найти все аксессуары, цена которых выше, чем любой единицы одежды”.)

Преобразование: Select product name and retail price from the products table joined with the categories table on category ID

where category description is “accessories” and retail price
is greater than all of the selection of retail price
from the products table joined with the categories table
on category ID where category name is “clothing”
(Выбрать наименование товара и розничную цену
из таблицы “Товары”, соединенной с таблицей
“Категории” по идентификатору категории, где описание
категории — “аксессуары”, а розничная цена больше,
чем любая из выборки розничной цены из таблицы
“Товары”, соединенной с таблицей “Категории”
по идентификатору категории, где наименование
категории — “одежда”)

Уточнение:

Select product name ~~and~~ retail price from ~~the~~ products ~~table~~ join~~ed with the~~ categories ~~table~~ on category ID where category description ~~is~~ = “accessories” and retail price ~~is greater than~~ > all ~~of the~~ (select~~ion of~~ retail price from ~~the~~ products ~~table~~ join~~ed with the~~ categories ~~table~~ on category ID where category name ~~is~~ = “clothing”)
(Выбрать наименование товара, розничную цену
из “Товары”, соединенной с “Категории”
по идентификатору категории, где описание категории =
“аксессуары” и розничная цена > любой (Выбрать
розничную цену из “Товары”, соединенной
с “Категории” по идентификатору категории,
где наименование категории = “одежда”))

SQL

```
SELECT Products.ProductName, Products.RetailPrice
FROM Products
INNER JOIN Categories
ON Products.CategoryID = Categories.CategoryID
WHERE Categories.CategoryDescription = 'Accessories'
AND Products.RetailPrice > ALL
(SELECT Products.RetailPrice
FROM Products
INNER JOIN Categories
ON Products.CategoryID =
    Categories.CategoryID
WHERE Categories.CategoryDescription =
    'Clothing')
```

Что здесь происходит? Подзапрос извлекает все цены для единиц одежды. Затем внешний запрос выведет список всех аксессуаров, цена которых больше, чем цена единицы одежды из подзапроса.

Существование — EXISTS

Как принадлежность к множеству (IN), так и количественно определенные предикаты (SOME/ANY/ALL) выполняют сравнение с типизированным выражением — обычно это столбец из источника, определенного в условии FROM внешнего запроса. Иногда полезно просто знать, что интересующая строка EXISTS (существует) в наборе результатов, возвращенном подзапросом. В главе 8 показана методика решения задач "AND" с использованием сложных операторов INNER JOIN. EXISTS также можно использовать для решения задач подобного типа.

Рассмотрим еще раз задачу, решенную в главе 8.

"Find all the customers who ordered a bicycle and who also ordered a helmet".
("Найти всех клиентов, заказавших велосипед и также заказавших шлем".)

Преобразование: Select customer ID, customer first name, and customer last name from the customer table where there exists some row from the orders table joined with the order details table on order ID, and then joined with the products table on product ID where product name contains "Bike" and the orders table customer ID equals the customers table customer ID, and there also exists some row from the orders table joined with the order details table on order ID, and then joined with the products table on product ID where product name contains "Helmet" and the orders table customer ID equals the customers table customer ID
(Выбрать идентификатор клиента, имя клиента и фамилию клиента из таблицы "Клиент", в которой существует несколько строк из таблицы "Заказы", соединенной с таблицей "Детали заказа" по идентификатору заказа, а затем соединенной с таблицей "Товары" по идентификатору товара, где наименование товара содержит "Bike" и идентификатор клиента таблицы "Заказы" равен идентификатору клиента таблицы "Клиенты", а также существует несколько строк из таблицы "Заказы", соединенной с таблицей "Детали заказа" по идентификатору заказа, а затем соединенной с таблицей "Товары" по идентификатору товара, где наименование товара содержит "Helmet" и идентификатор клиента таблицы "Заказы" равен идентификатору клиента таблицы "Клиенты")

Уточнение: Select customer ID, customer first name, ~~and~~ customer last name from ~~the~~ customers ~~table~~ where ~~there~~ exists ~~some row~~ (SELECT * from ~~the~~ orders ~~table~~ join~~ed with the~~

order details ~~table~~ on order ID, ~~and then~~ join~~ed with the~~ products ~~table~~ on product ID where product name ~~contains~~ LIKE “Bike” and ~~the~~ orders ~~table~~ customer ID ~~equals~~ = ~~the~~ customers table customer ID), and ~~there also~~ exists ~~some row~~ (SELECT * from ~~the~~ orders ~~table~~ join~~ed with the~~ order details ~~table~~ on order ID, ~~and then~~ join~~ed with the~~ products ~~table~~ on product ID where product name ~~contains~~ LIKE “Helmet” and ~~the~~ orders ~~table~~ customer ID ~~equals~~ = ~~the~~ customers ~~table~~ customer ID)
(Выбрать идентификатор клиента, имя клиента, фамилию клиента из “Клиент”, в которой существуют (Выбрать * из “Заказы”, соединенной с “Детали заказа" по идентификатору заказа, соединенной с “Товары” по идентификатору товара, где наименование товара LIKE “Bike” и идентификатор клиента “Заказы” = идентификатору клиента “Клиенты”), и существует (Выбрать * из “Заказы”, соединенной с “Детали заказа” по идентификатору заказа, соединенной с “Товары” по идентификатору товара, где наименование товара LIKE “Helmet” и идентификатор клиента “Заказы” = идентификатору клиента “Клиенты”))

SQL

```
SELECT Customers.CustomerID,
    Customers.CustFirstName,
    Customers.CustLastName
FROM Customers
WHERE EXISTS
    (SELECT *
    FROM (Orders
    INNER JOIN Order_Details
    ON  Orders.OrderNumber =
        Order_Details,OrderNumber)
    INNER JOIN Products
    ON  Products.ProductNumber =
        Order_Details.ProductNumber
    WHERE Products.ProductName Like '%Helmet'
    AND Orders.CustomerID = Customers.CustomerID)
AND EXISTS
    (SELECT *
    FROM (Orders
    INNER JOIN Order_Details
    ON  Orders.OrderNumber =
        Order_Details.OrderNumber)
    INNER JOIN Products
```

```
ON  Products.ProductNumber =
    Order_Details.ProductNumber
WHERE Products.ProductName Like '%Bike'
AND Orders.CustomerID = Customers.CustomerID)
```

Обратите внимание, что можно использовать любое имя столбца из любой таблицы в условии FROM, как столбца, извлекаемого в условии SELECT подзапроса. Для всех столбцов выбрано использование краткого обозначения "*". Этот запрос можно представить по-другому, как "Указать клиентов, для которых существует несколько строк в деталях заказов на велосипеды и также существует несколько строк в деталях заказа на шлемы". Поскольку идентификатор заказа не сопоставляется, можно не беспокоиться о том, что клиент заказал велосипед в одном заказе, а шлем в другом.

Внимание! Это настолько интересный запрос, что его решение сохранено как "Cust_Bikes_And_Helmets_EXISTS" в учебной базе данных. Исходное решение с INNER JOIN можно найти под именем "Cust_Bike_And_Helmets_JOIN".

Использование подзапросов

На этот момент вы уже должны достаточно хорошо понимать концепцию использования подзапроса как для генерации выходных столбцов, так и для выполнения сложного сравнения в условии WHERE. Для лучшего представления о широких возможностях использования подзапросов мы покажем задачи, которые можно решить с их помощью.

Выражения из столбцов

Использование подзапроса для извлечения отдельного значения из связанной таблицы можно более эффективно выполнять в JOIN. Однако при рассмотрении обобщенных функций подзапросы для извлечения результатов вычислений функций делают эту мысль намного более интересной. Использование обобщенных функций мы рассмотрим в следующей главе, а пока — вот некоторые задачи, которые можно решить, используя подзапрос для формирования столбца вывода.

"Привести список поставщиков и подсчитайте товары, проданные ими нам".

"Вывести на экран товары и самую последнюю дату заказа товара".

"Показать эстрадных артистов и подсчитать количество ангажементов у каждого из них".

"Показать каждого клиента и дату последней заявки, зарегистрированной им".

"Привести список всего персонала и количество лекций у каждого преподавателя".

"Вывести на экран дисплея все предметы и количество лекций по каждому предмету в понедельник".

"Показать всех игроков в боулинг и количество сыгранных ими игр".

"Вывести на экран игроков в боулинг и самую важную сыгранную ими игру".

"Привести список всех видов мяса и число рецептов, в которых появляется каждый из них".

"Показать типы рецептов и количество рецептов в каждом из типов".

Фильтры

Теперь можно реально расширить наш набор инструментов для решения сложных проблем. В этой главе мы исследовали множество интересных способов использования подзапросов как фильтров в условии WHERE. Вот пример задач, которые можно решить, используя этот метод. Многие из этих задач уже были решены в предыдущих главах, а теперь мы попробуем решить их альтернативным способом.

Внимание! Ключевое слово (слова), которое можно использовать для решения задачи, стоит в скобках после формулировки запроса.

"Привести список клиентов, заказавших велосипеды". (IN)

"Вывести на экран дисплея клиентов, заказавших одежду или предметы первой необходимости". (= SOME)

"Вывести список всех клиентов, заказавших когда-либо велосипедный шлем". (IN)

"Найти всех клиентов, заказавших велосипед, но не заказавших шлем". (NOT EXIST)

"Какие товары никогда не заказывались?" (NOT IN)

"Вывести список клиентов, которые подали заявки на эстрадных артистов, исполняющих музыку в стиле "кантри" или "кантри-рок". (IN)

"Найти эстрадных артистов, отыгравших ангажементы для клиентов Бонниксен или Росалес". (= SOME)

"Вывести на экран дисплея агентов, которые не заказывали эстрадных артистов". (NOT IN)

"Вывести список эстрадных артистов, отыгравших ангажементы для клиентов Бонниксен или Росалес". (EXIST)

"Вывести на экран дисплея студентов, записавшихся на курс лекций во вторник". (IN)

"Показать студентов, которые имеют средний балл 85 и выше по курсу "Искусство" и также имеют средний балл 85 и выше по курсу "Вычислительная техника". (EXISTS)

"Вывести на экран дисплея студентов, которые никогда не отказывались от курса лекций". (NOT IN)

"Привести список предметов, предлагаемых в среду". (IN)

"Вывести на экран дисплея капитанов команд с текущим средним количеством очков более высоким, чем у любого из остальных участников их команды". (> ALL)

"Показать еще не сыгранные турниры". (NOT IN)

"Найти игроков, у которых предварительное количество очков 170 или выше как по Зандербирд Лэйнс, так и по Болеро Лэйнс". (EXISTS)

"Привести список всех игроков в боулинг, у которых текущее среднее количество очков ниже, чем у всех остальных игроков этой же команды". (< ALL)

"Показать рецепты, содержащие говядину и чеснок". (EXISTS)

"Вывести на экран дисплея все компоненты рецептов, содержащих морковь". (IN)

"Привести список всех компонентов, используемых в некотором рецепте, где единицы измерения количества не являются единицами измерения по умолчанию". (<> SOME)

"Привести список компонентов, не используемых еще ни в одном рецепте". (NOT IN)

Примеры операторов

Ознакомимся теперь с довольно постоянным множеством примеров, все из которых используют один или несколько подзапросов. Они взяты из учебных баз данных и иллюстрируют использование подзапросов либо для формирования столбцов вывода, либо в качестве фильтров.

Сюда также включены примеры наборов результатов, которые должны возвращать эти операции. Мы поместили их сразу после графического описания синтаксиса SQL в виде линии. Имя, которое появляется непосредственно над набором результатов, присвоено каждому запросу в учебной базе данных, которую можно найти на сайте издательства "Лори". Каждый запрос сохранен в соответствующем примере базы данных, во вложенной папке "Chapter 11". Чтобы загрузить примеры на свой компьютер и проверить их, следуйте указаниям, приведенным в начале книги.

Внимание! Все имена столбцов и таблиц, используемые в этих примерах, взяты из учебных структур баз данных, представленных в приложении В. Поскольку многие их этих примеров используют сложные соединения, СУБД может выбрать другой способ решения. Поэтому несколько первых строк, которые показаны здесь, могут не совпадать точно с результатом, полученным вами, но общее количество строк должно быть одинаково. Для упрощения процесса этапы преобразования и уточнения для всех примеров объединены.

Подзапросы в выражениях

База данных заказов на закупку

"List vendors and a count of the products they sell to us".
(Привести список поставщиков и подсчитать количество товаров, проданных ими нам".)

Преобразование/ Уточнение:

Select vendor name ~~and also~~ (select ~~the~~ count(*) ~~of products~~ from ~~the~~ product vendors ~~table for this vendor~~ WHERE vendors vendor ID = product vendors vendor ID) from ~~the~~ vendors ~~table~~
(Выбрать имя поставщика и (Выбрать count(*) из "Поставщики товаров", где идентификатор поставщика из "Поставщики" = идентификатору поставщика) из "Поставщики")

SQL

```
SELECT VendName,
    (SELECT COUNT (*)
    FROM Product_Vendors
    WHERE Product_Vendors.VendorID = Vendors.VendorID)
AS VendProductCount
FROM Vendors
```

Vendors_Product_Count (10 строк)

VendName	VendProductCount
Shinoman, Incorporated	3
Viscount	6
Nikoma of America	5
ProFormance	3
Kona, Incorporated	1
Big Sky Mountain Bikes	22
Dog Ear	9
<< остальные строки >>	

База данных агентства эстрадных мероприятий

"Display each customer and the date of the last booking they made". ("Вывести на экран дисплея каждого клиента и дату последнего сделанного им резервирования".)

Преобразование/ Уточнение:

Select customer first name, customer last name, ~~and also~~ (select ~~the highest~~ MAX(start date) from engagements ~~for this customer~~ WHERE engagements customer ID = customers customer ID) from ~~the~~ customers ~~table~~ (Выбрать имя клиента, фамилию клиента (Выбрать MAX(дата начала) из "Ангажементы", где идентификатор клиента из "Ангажементы" = идентификатору клиента из "Клиенты") из "Клиенты")

SQL

```
SELECT Customers.CustFirstName,
    Customers.CustLastName,
    (Select Max(StartDate)
    FROM Engagements
    WHERE Engagements.CustomerID =
        Customers.Customer ID)
AS LastBooking
FROM Customers
```

Внимание! Столбец LastBooking для некоторых клиентов пуст (Null), потому что у этих клиентов не имеется заявок.

Customers_Last_Booking (15 строк)

CustFirstName	CustLastName	LastBooking
Sally	Callahan	1999-12-23
Ann	Fuller	1999-12-17
James	Leverling	1999-12-26
Kenneth	Peacock	1999-12-24
Elizabeth	Hallmark	1999-12-19
Thomas	Fuller	1999-12-23
Amelia	Buchanan	1999-12-19
Samuel	Peacock	
Sarah	Thompson	1999-12-24
<< остальные строки >>		

База данных расписания занятий

"Display all subjects and count of classes for each subject on Monday". ("Вывести на экран дисплея все предметы и количество лекций по каждому предмету в понедельник".)

Преобразование/ Уточнение:

Select subject name ~~and also~~ (select ~~the~~ count(*) ~~of~~ ~~classes~~ from ~~the~~ classes ~~table~~ where Monday schedule ~~is~~ = true ~~for this subject~~ and classes subject ID = subjects subject ID) from ~~the~~ subjects ~~table~~
(Выбрать название предмета (Выбрать count(*) курсы лекций из "Курсы лекций", где расписание на понедельник = Истина и идентификатор предмета курса лекций = идентификатору предмета из "Предметы") из "Предметы")

SQL

```
SELECT Subjects.SubjectName,
    (SELECT Count (*)
    FROM Classes
    WHERE MondaySchedule = -1
    AND Classes.SubjectID = Subjects.SubjectID)
AS MondayCount
FROM Subjects
```

Внимание! Обязательно используйте проверку на "true", которую поддерживает система баз данных. Помните, что некоторые системы баз данных требуют проверки на ключевое слово "TRUE" или на целое значение 1.

Subjects_Monday_Count (56 строк)

SubjectName	MondayCount
Financial Accounting Fundamentals I	2
Financial Accounting Fundamentals II	1
Fundamentals of Managerial Accounting	1
Intermediate Accounting	1
Business Tax Accounting	1
Introduction to Business	0
Developing A Feasibility Plan	0
Introduction to Enterpreneurship	1
<< остальные строки >>	

Внимание! Вместо того чтобы возвратить значение Null, когда строки отсутствуют, агрегатная функция возвращает ноль.

База данных лиги игры в боулинг

"Display the bowlers and highest game each bowled".
("Вывести на экран дисплея игроков в боулинг и самую важную сыгранную ими игру".)

Преобразование/ Уточнение:

Select bowler first name, bowler last name, ~~and also~~ (select ~~the highest~~ MAX(raw score) from ~~the~~ bowler scores ~~table for this bowler~~ WHERE bowler scores bowler ID = bowlers bowler ID) from ~~the~~ bowlers ~~table~~
(Выбрать имя игрока в боулинг, фамилию игрока в боулинг (Выбрать MAX(предварительное количество очков) из "Очки игроков в боулинг", где идентификатор игрока в боулинг из "Очки игроков в боулинг" = идентификатору игрока в боулинг из "Игроки в боулинг") из "Игроки в боулинг")

SQL

```
SELECT Bowlers.BowlerFirstName,
    Bowlers.BowlerLastName,
    (SELECT MAX(RawScore)
    FROM Bowler_Scores
WHERE Bowler_Scores.BowlerID = Bowlers.BowlerID)
AS HighScore
FROM Bowlers
```

Bowler_High_Score (32 строки)

BowlerFirstName	BowlerLastName	HighScore
Barbara	Fournier	164
David	Fournier	178
John	Kennedy	191
Sara	Kennedy	149
Ann	Patterson	165
Neil	Patterson	179
Carol	Viescas	195
David	Viescas	150
<< остальные строки >>		

База данных рецептов

"List all the meats and the count of recipes each appears in".
("Предоставить список всех видов мяса и число рецептов, в которых появляется каждый из них".)

Преобразование/ Уточнение:

Select ingredient class description, ingredient name, ~~and also~~ (select ~~the~~ count(*) ~~of rows~~ from ~~the~~ recipe ingredients ~~table for this ingredient~~ where recipe ingredients ingredient ID = ingredients ingredient ID) from ~~the~~ ingredient classes ~~table~~ join~~ed with the~~ ingredients ~~table~~ on ingredient class ID where ingredient class description ~~is~~ = 'meat'
(Выбрать описание вида компонента, название компонента (Выбрать count(*) из "Компоненты рецепта", где идентификатор компонента из "Компоненты рецепта" = идентификатору компонента из "Компоненты") из "Виды компонентов", соединенной с "Компоненты" по идентификатору вида компонента, где описание вида компонента = 'мясо')

SQL

```
SELECT Ingredient_Classes.IngredientClassDescription,
    Ingredients.IngredientName,
    (SELECT COUNT (*)
    FROM Recipe_Ingredients
    WHERE Recipe_Ingredients.IngredientID =
        Ingredients.IngredientID)
AS RecipeCount
FROM Ingredient_Classes
```

Meat_Ingredient_Recipe_Count (11 строк)

IngredientClassDescription	IngredientName	RecipeCount
Meat	Beef	2
Meat	Chicken, Fryer	0
Meat	Bacon	0
Meat	Chicken, Pre-cut	0
Meat	T-bone Steak	0
Meat	Chicken Breast	0
Meat	Chicken Leg	1
Meat	Chicken Wing	0
Meat	Chicken Thigh	1
Meat	New York Steak	0
Meat	Ground Pork	1

```
INNER JOIN Ingredients
ON  Ingredient_Classes.IngredientClassID =
    Ingredients.IngredientClassID
WHERE Ingredient_Classes.IngredientClassDescription =
    'Meat'
```

Подзапросы в фильтрах

База данных заказов на закупку

"Display customers who ordered Clothing or Accessories".
("Вывести на экран дисплея клиентов, заказавших одежду или аксессуары".)

Преобразование/ Уточнение:

Select customer ID, customer first name, customer last name from ~~the~~ customers ~~table~~ where customer ID ~~is~~ in ~~the~~ (select~~ion of~~ customer ID from ~~the~~ orders ~~table~~ join~~ed with the~~ order details ~~table~~ on order number, ~~then~~ join~~ed with the~~ products ~~table~~ on product number, ~~and then~~ join~~ed with the~~ categories ~~table~~ on category ID where category description ~~is~~ = 'clothing' or category description ~~is~~ = 'accessories'

(Выбрать идентификатор клиента, имя клиента, фамилию клиента из "Клиенты", где идентификатор клиента в (Выбрать идентификатор клиента из "Заказы", соединенной с "Детали заказа" по номеру заказа, соединенной с "Товары" по номеру товара, соединенной с "Категории" по идентификатору категории, где описание категории = 'одежда' или описание категории = 'аксессуары')

SQL

```
SELECT Customers.CustomerID,
    Customers.CustFirstName, Customers.CustLastName
FROM Customers
WHERE Customers.CustomerID = ANY
(SELECT Orders.CustomerID
FROM ((Orders
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber)
INNER JOIN Products
ON  Products.ProductNumber =
    Order_Details.ProductNumber)
INNER JOIN Categories
ON  Categories.CategoryID = Products.CategoryID
WHERE Categories.CategoryDescription = 'Clothing'
OR  Categories.CategoryDescription = 'Accessories')
```

Customers_Clothing_OR_Accessories (27 строк)

CustomerID	CustFirstName	CustLastName
1001	Suzanne	Viescas
1002	Will	Thompson
1003	Gary	Hallmark
1004	Michael	Davolio
1005	Kenneth	Peacock
1006	John	Viescas
1007	Laura	Callahan
1008	Neil	Patterson
<< остальные строки >>		

База данных агентства эстрадных мероприятий

"List the entertainers who played engagements for customers Bonnicksen and Rosales".
("Вывести список эстрадных артистов, отыгравших ангажементы для клиентов Бонниксен или Росалес".)

Внимание! Эта задача решена в главе 8 с использованием JOIN и двух сложных табличных подзапросов. В этот раз будет использоваться EXISTS.

Преобразование/ Уточнение:

Select entertainer ID, ~~and~~ entertainer stage name from ~~the~~ entertainers ~~table~~ where ~~there~~ exists (SELECT * ~~some row~~ from ~~the~~ customers ~~table~~ join~~ed with the~~ engagements ~~table~~ on customerID where customer last name ~~is~~ = 'Rosales' and ~~the~~ entertainers ~~table~~ entertainer ID ~~equals~~ = ~~the~~ engagements ~~table~~ entertainer ID), and ~~there also~~ exists (SELECT * ~~some row~~ from ~~the~~ customers ~~table~~ join~~ed with the~~ engagements ~~table~~ on customerID where customer last name ~~is~~ = 'Bonnicksen' and ~~the~~ entertainers ~~table~~ entertainer ID ~~equals~~ = ~~the~~ engagements ~~table~~ entertainer ID)
(Выбрать идентификатор эстрадного артиста, псевдоним эстрадного артиста из "Эстрадные артисты", где существует (Выбрать * из "Клиенты", соединенной

с “Ангажементы” по идентификатору клиента,
где фамилия клиента = ‘Росалес’ и идентификатор
эстрадного артиста в “Эстрадные артисты” =
идентификатору эстрадного артиста в “Ангажементы”),
и существует (Выбрать * из “Клиенты”, соединенной
с “Ангажементы” по идентификатору клиента,
где фамилия клиента = ‘Бонниксен’ и идентификатор
эстрадного артиста в “Эстрадные артисты =
идентификатору эстрадного артиста в ”Ангажементы”)) .

SQL

```
SELECT Entertainers.EntertainerID,
    Entertainers.EntStageName
FROM Entertainers
WHERE EXISTS
    (SELECT *
    FROM Customers
    INNER JOIN Engagements
    ON Customers.CustomerID =
        Engagements.CustomerID
    WHERE Customers.CustLastName = ‘Rosales’
    AND Engagements.EntertainerID =
        Entertatners.EntertainerID)
AND EXISTS
    (SELECT *
    FROM Customers
    INNER JOIN Engagements
    ON Customers.CustomerID =
        Engagements.CustomerID
    WHERE Customers.CustLastName='Bonnicksen'
    AND Engagements.EntertainerID =
        Entertainers.EntertainerID)
```

Entertainers_Bonnicksen_AND_Rosales_EXISTS

(4 строки)

EntertainerID	EntStageName
1008	Country Feeling
1009	Katherine Ehrlich
1010	Saturday Revue
1011	Julia Schnebly

База данных расписания занятий

"Display students who have never withdrawn from a class".
("Вывести на экран дисплея студентов, которые никогда не отказывались от курса лекций".)

Преобразование/ Уточнение:

Select student ID, student first name, ~~and~~ student last name from ~~the~~ students ~~table~~ where ~~the~~ student ID ~~is~~ not in ~~the~~ (selecti~~on of~~ student ID from ~~the~~ student schedules ~~table~~ join~~ed with the~~ student class status ~~table~~ on class status where class status description ~~is~~ = 'withdrew'
(Выбрать идентификатор студента, имя студента, фамилию студента из "Студенты", где идентификатора студента нет в (Выбрать идентификатор студента из "Расписание студента", соединенной с "Состояние курса лекций студента" по состоянию курса лекций, где описание состояния курса лекций = 'отказался'))

SQL

```
SELECT Students.StudentID,
    Students.StudFirstName, Students.StudLastName
FROM Students
WHERE Students.StudentID NOT IN
(SELECT Student_Schedules.StudentID
FROM Student_Schedules
INNER JOIN Student_Class_Status
ON Student_Schedules.ClassStatus =
    Student_Class_Status.ClassStatus
WHERE Student_Class_Status.ClassStatusDescription
    = 'Withdrew')
```

Students_Never_Withdrawn (15 строк)

StudentID	StudFirstName	StudLastName
1002	Andrew	Fuller
1003	Sarah	Leverling
1004	Carol	Peacock
1005	Sally	Callahan
1006	Steven	Buchanan
1007	Elizabeth	Hallmark
1008	Sara	Kennedy
1010	Mary	Fuller
<< остальные строки >>		

Внимание! Это достаточно простой запрос, который в подзапросе находит всех студентов, когда-либо отказывавшихся от курса лекций, а затем запрашивает всех студентов, не содержащихся (NOT IN) в этом списке. Можете ли вы представить, как это сделать с OUTER JOIN?

База данных лиги игры в боулинг

"Display team captains with a current average higher than all other members on their team".
("Вывести на экран дисплея капитанов команд с текущим средним количеством очков более высоким, чем у любого из остальных участников их команд".)

Преобразование/ Уточнение:

Select team name, bowler ID, bowler first name, bowler last name, ~~and~~ current average from ~~the~~ bowlers ~~table~~ join~~ed with the~~ teams ~~table~~ on bowler ID matches = captain ID where ~~the~~ current average ~~is greater than~~ > all ~~of the~~ (select~~ion of~~ current average from bowlers where ~~the~~ bowler ID ~~is not equal~~ <> ~~the current~~ bowler ID and ~~the~~ team ID ~~is equal~~ = ~~to the current~~ team ID
(Выбрать название команды, идентификатор игрока в боулинг, имя, фамилию игрока в боулинг, текущее среднее из "Игроки в боулинг", соединенной с "Команды" по идентификатору игрока в боулинг = идентификатору капитана, где текущее среднее > всех (Выбрать текущее среднее из "Игроки в боулинг", где идентификатор игрока в боулинг <> идентификатор игрока в боулинг и идентификатор команды = идентификатору команды))

SQL

```
SELECT Teams.TeamName, Bowlers.BowlerID,
    Bowlers.BowlerFirstName, Bowlers.BowlerLastName,
    Bowlers.CurrentAverage
FROM Bowlers
INNER JOIN Teams
ON  Bowlers.BowlerID = Teams.CaptainID
WHERE Bowlers.CurrentAverage > ALL
    (SELECT B2.CurrentAverage
    FROM Bowlers AS B2
    WHERE B2.BowlerID <> Bowlers.BowlerID
    AND B2.TeamID = Bowlers.TeamID)
```

Team_Captains_High_Average (2 строки)

TeamName	BowlerID	BowlerFirstName	BowlerLastName	CurrentAverage
Sharks	5	Ann	Patterson	170.00
Barracudas	16	Richard	Sheskey	165.00

Внимание! Мы явно присвоили второй копии таблицы Bowlers (Игроки в боулинг) в подзапросе псевдоним, чтобы стало совершенно понятно, о чем идет речь. Специально не хотелось сравнивать со средним значением текущего игрока, что вызвало бы сбой при выполнении предиката > ALL. И сравнение хотелось выполнить только с другими игроками этой же команды.

База данных рецептов

"Display all the ingredients for recipes that contain carrots".
("Вывести на экран все компоненты рецептов, содержащих морковь".)

Преобразование/ Уточнение:

Select recipe title and ingredient name from ~~the~~ recipes ~~table~~ join~~ed with the~~ recipe ingredients ~~table~~ on recipe ID, ~~and then~~ join~~ed with the~~ ingredients ~~table~~ on ingredient ID where recipe ID ~~is~~ in ~~the~~ (select~~ion of~~ recipe ID from ~~the~~ ingredients ~~table~~ join~~ed with the~~ recipe ingredients ~~table~~ on ingredient ID where ingredient name ~~is~~ = 'carrot')
(Выбрать заголовок рецепта, название компонента из "Рецепты", соединенной с "Компоненты рецепта" по идентификатору рецепта, соединенной с "Компоненты" по идентификатору компонента, где идентификатор рецепта в (Выбрать идентификатор рецепта из "Компоненты", соединенной с "Компоненты рецепта" по идентификатору компонента, где название компонента = 'морковь'))

SQL

```
SELECT Recipes.RecipeTitle,
    Ingredients.IngredientName
FROM (Recipes
INNER JOIN Recipe_Ingredients
ON  Recipes.RecipeID =
    Recipe_Ingredients.RecipeID)
INNER JOIN Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID
WHERE Recipes.RecipeID
IN
    (SELECT Recipe_Ingredients.RecipeID
    FROM Ingredients
    INNER JOIN Recipe_Ingredients
    ON  Ingredients.IngredientID =
        Recipe_Ingredients.IngredientID
    WHERE Ingredients.IngredientName = 'carrot')
```

Внимание! Если поместить фильтр для 'carrot' во внешнем соединении, то в выводе будет получен только компонент "морковь". В этой задаче нам хотелось показать *все* компоненты из всех рецептов, в которых используется морковь, поэтому подзапрос является хорошим способом решения.

Recipes_Ingredients_With_Carrots (16 строк)

RecipeTitle	IngredientName
Irish Stew	Beef
Irish Stew	Onion
Irish Stew	Potato
Irish Stew	Carrot
Irish Stew	Water
Irish Stew	Guinness Beer
Salmon Filets in Parchment Paper	Salmon
Salmon Filets in Parchment Paper	Carrot
Salmon Filets in Parchment Paper	Leek
<< остальные строки >>	

Итоги

В данной главе было приведено описание трех типов подзапросов, определенных в стандарте SQL,— скалярного, строкового и табличного. Мы также кратко обсудили использование строковых подзапросов, которые поддерживает пока еще не очень много коммерческих реализаций.

Мы рассмотрели использование подзапроса для формирования выражения со столбцами в условии SELECT и обсудили простой пример, а затем ввели две агрегатные (обобщенные) функции, которые полезны для извлечения соответствующей суммарной информации из другой таблицы.

Подзапросы можно использовать для создания сложных фильтров в условии WHERE. Мы рассмотрели простые сравнения, а затем ввели специальные ключевые слова для сравнений — IN, SOME, ANY, ALL и EXISTS,— которые нужны для построения предикатов в подзапросах.

Подзапросы полезны для решения многих задач. В приведенных нами примерах подзапросы использовались для выражений со столбцами и для создания фильтров

Задачи для самостоятельного решения

Ниже приведены формулировки запросов и имена решений этих запросов в учебных базах данных. Попрактикуйтесь немного и разработайте SQL для каждого запроса, а затем сверьте свой ответ с решением, которое сохранено нами в этих базах данных. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов,— важно, чтобы набор результатов был тем же.

База данных заказов на закупки

1. *"Display products and the latest date the product was ordered"*.
   ("Вывести на экран товары и самую последнюю дату, когда товар был заказан".)
   (*Совет*: Используйте агрегатную функцию MAX.)
   Решение можно найти в Products_Last_Date (40 строк)

2. *"List customers who ordered bikes"*.
   ("Привести список клиентов, заказавших велосипеды".)
   (*Совет*: Постройте фильтр, используя IN.)
   Решение можно найти в Customers_Order_Bikes (23 строки).

3. *"Find all customers who ordered a bicycle but who did not ordered a helmet"*.
   ("Найдите всех клиентов, заказавших велосипед, но не заказавших шлем".)
   (*Совет*: Начните с предыдущего запроса и добавьте фильтр, используя NOT EXIST.)
   Решение можно найти в Customer_Bikes_No_Helmets (2 строки).

4. *"What products have never been ordered?"*
   ("Какие товары никогда не заказывали?")
   (*Совет*: Постройте фильтр, используя NOT IN.)
   Решение можно найти в Products_Not_Ordered (2 строки).

База данных агентства эстрадных мероприятий

1. *"Show me entertainers and count of each entertainer's engagements"*.
   ("Показать эстрадных артистов и количество ангажементов у каждого из них".)
   (*Совет*: Воспользуйтесь агрегатной функцией COUNT.)
   Решение можно найти в Entertainers_Engagement_Count (13 строк).

2. *"List customers who have booked entertainers who play country or country rock"*.
   ("Вывести список клиентов, которые подали заявки на эстрадных артистов, играющих музыку в стиле "кантри" или "кантри-рок".)
   (*Совет*: Постройте фильтр, используя IN.)
   Решение можно найти в Customers_Who_Like_Country (15 строк).

3. *"Find the entertainers who played engagements for customers Bonnicksen or Rosales"*.
("Найдите эстрадных артистов, имевших ангажементы для клиентов Бонниксен или Росалес".)
(*Совет*: Постройте фильтр, используя = SOME.)
Решение можно найти в Entertainers_Bonnicksen_OR_Rosales_SOME (9 строк)

4. *"Display agents who haven't booked an entertainer"*.
("Вывести на экран дисплея агентов, которые никогда не подавали заявок на эстрадного артиста".)
(*Совет*: Постройте фильтр, используя NOT IN.)
Решение можно найти в Bad_Agents (1 строка).

База данных расписания занятий

1. *"List all staff members and the count of classes each teaches"*.
("Привести список всех штатных сотрудников и количество лекций, читаемых каждым из преподавателей".)
(*Совет*: Воспользуйтесь агрегатной функцией COUNT.)
Решение можно найти в Staff_Class_Count (27 строк).

2. *"Display students enrolled in a class on Tuesday"*.
("Вывести на экран студентов, записавшихся на лекции во вторник".)
(*Совет*: Постройте фильтр, используя IN.)
Решение можно найти в Students_In_Class_Tuesdays (18 строк).

3. *"Show me the students who have the average score of 85 or better in Art and who also have an average score 85 or better in Computer Science"*.
("Показать студентов, у которых по курсу "Искусство" средний бал 85 и выше и по курсу "Вычислительная техника" средний бал также 85 и выше".)
(*Совет*: Постройте фильтр, используя EXISTS.)
Решение можно найти в Good_Art_CS_Students_EXISTS (1 строка).

4. *"List the subjects taught on Wednesday"*.
("Привести список дисциплин, преподаваемых в среду".)
(*Совет*: Постройте фильтр, используя IN.)
Решение можно найти в Subjects_On_Wednesday (45 строк).

База данных лиги игры в боулинг

1. *"Show me all the bowlers and a count of games each bowled"*.
("Показать всех игроков в боулинг и количество игр, сыгранных каждым из них".)
(*Совет*: Воспользуйтесь агрегатной функцией COUNT.)
Решение можно найти в Bowlers_And_Count_Games (52 строки).

2. *"Show me tournaments that haven't been played yet"*.
("Показать турниры, которые еще не разыгрывались".)
(*Совет*: Воспользуйтесь фильтром NOT IN.)
Решение можно найти в Tourneys Not_Played (6 строк).

3. *"Find the bowlers who had a raw score of 170 or better at both Thunderbird Lanes and Bolero Lanes"*.
("Найти игроков в боулинг, у которых предварительное количество очков 170 или выше как на Зандербирд Лэйнс, так и на Болеро Лэйнс".)
(*Совет*: Постройте фильтр, используя EXISTS.)
Решение можно найти в Good_Bowlers_TBird_And_Bolero_EXISTS (10 строк).

4. *"List all the bowlers who have a current average that's less than all of the other bowlers on the same team"*.
("Привести список всех игроков в боулинг, текущее среднее количество очков у которых меньше, чем у всех остальных игроков в боулинг из той же команды".)
(*Совет*: Постройте фильтр, используя < ALL.)
Решение можно найти в Bowlers_Low_Average (8 строк.)

База данных рецептов

1. *"Show me the types of recipes and the count of recipes in each type"*.
("Показать типы рецептов и количество рецептов в каждом типе".)
(*Совет*: Воспользуйтесь агрегатной функцией COUNT.)
Решение можно найти в Count_Of_Recipe_Types (7 строк.)

2. *"Show me the recipes that have beef and garlic"*.
("Показать рецепты, содержащие говядину и чеснок".)
(*Совет*: Постройте фильтр, используя EXISTS.)
Решение можно найти в Recipes_Beef_And_Garlic (1 строка).

3. *"List the ingredients that are used in some recipe where the measurement amount in the recipe is not the default measurement amount"*.
("Привести список компонентов, используемых в некотором рецепте, где единицы измерения количества не являются единицами измерения по умолчанию".)
(*Совет*: Постройте фильтр, используя <> SOME.)
Решение можно найти в Ingredients_Using_NonStandard_Measure (21 строка).

4. *"List ingredients not used in any recipe yet"*.
("Привести список компонентов, не используемых пока еще ни в одном из рецептов".)
(*Совет*: Постройте фильтр, используя NOT IN.)
Решение можно найти в Ingredients_No_Recipe (20 строк).

Часть IV

Суммирование данных и объединение в группы

12

Простая сумма

"Существует два вида статистики:
та, которую вы ищете,
и та, которую вы выдумываете".

— Рекс Стаут,
Смерть нищенки
(из цикла о Ниро Вульфе)

Вопросы, рассматриваемые в данной главе:

- Агрегатные функции
- Использование агрегатных функций в фильтрах
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

Теперь вам известно как выбрать нужные столбцы в запросе, определить выражения, добавляющие дополнительные уровни подробностей, соединить соответствующие таблицы, которые предоставляют нужные столбцы, и определить условия для фильтрации данных, отправленных в набор результатов. Все эти методы были показаны для того, чтобы научить вас извлекать подробную информацию из одной или нескольких таблиц в базе данных. В этой и последующих главах показано, как вернуться на шаг обратно и увидеть данные "с птичьего полета".

Из этой главы можно узнать, как использовать агрегатные функции, чтобы получить основную суммарную информацию. В главе 13 будет показано, как организовать данные в группы в условии GROUP BY оператора SELECT, а в главе 14 будут представлены различные методы фильтрации, которые можно применить к данным после их группирования.

Агрегатные функции

Запросы, с которыми работали до сих пор, требовали ответов, включающих значения отдельных столбцов, из строк возвращенных условиями FROM и WHERE. Однако часто встречаются запросы, которые для получения ответа требуют вычисления значений по нескольким строкам.

"Сколько наших клиентов живет в Сиэтле?"

"Какова наименьшая и наибольшая цена, установленная для шлема в вашей инвентарной ведомости?"

"Сколько курсов лекций читает Майкл Хернандес?"

"Во сколько начинается самая первая лекция?"

"Какова средняя продолжительность лекции ?"

"Какова общая сумма для заказа номер 12?"

Стандарт SQL обеспечивает совокупность *агрегатных функций*, которая позволяет вычислить отдельное значение для строк из набора результата или из значений, возвращенных типизированным выражением. Агрегатную функцию можно применить ко всем строкам или значениям или с помощью условия WHERE применить ее к конкретному множеству строк или значений. Например, можно применить агрегатную функцию для определения наибольшего или наименьшего значения типизированного выражения, подсчета количества строк в наборе результата или вычисления общей суммы, используя только определенные значения из типизированного выражения. На рис. 12.1 представлен синтаксис для всех агрегатных функций.

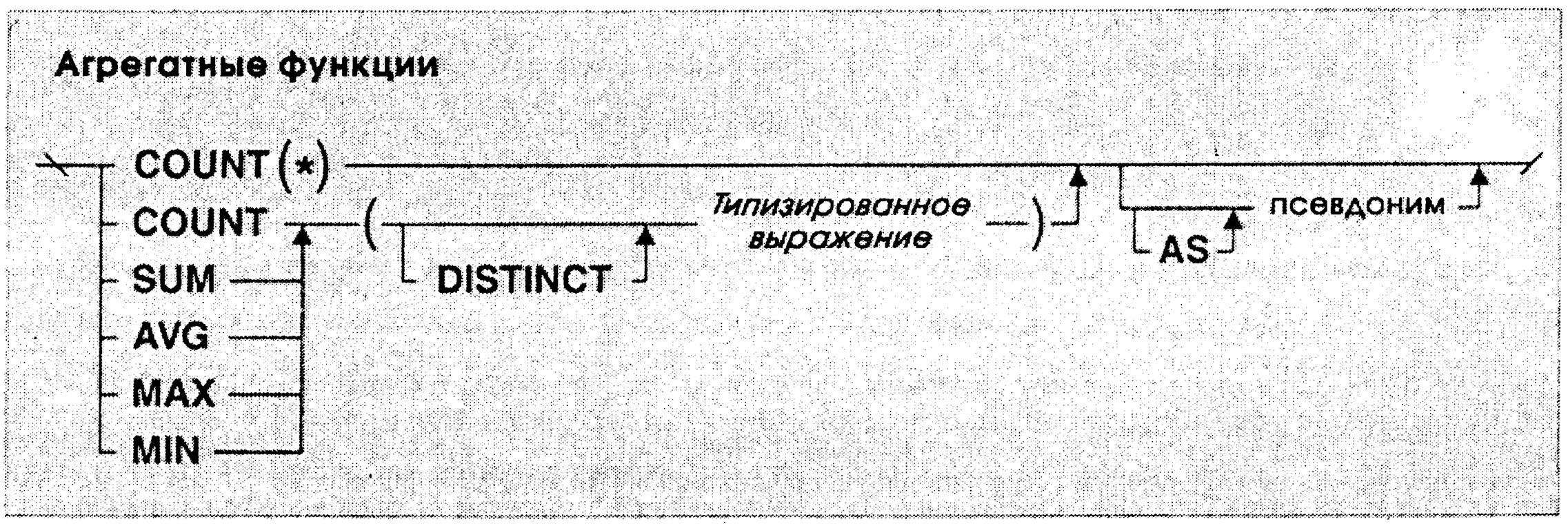


Рис. 12.1. *Синтаксическая диаграмма для агрегатных функций*

Агрегатные функции имеют очень простой и непосредственный синтаксис. В данной главе показано, как использовать эти функции в условии SELECT, а в главе 14 изучается использование этой функции в условии HAVING.

Каждая агрегатная функция возвращает единственное значение независимо от того, обрабатываются ли строки из набора результата или значения, возвращенные типизированным выражением. За исключением функции COUNT(*), все агрегатные функции автоматически игнорируют значения Null. В условии SELECT одновременно можно использовать несколько агрегатных функций и даже можно смешивать типизированные выражения, содержащие агрегатные функции, с типизированными выражениями, содержащими *литеральные значения*. Однако *не допускается*

смешивание типизированных выражений, содержащих агрегатные функции, с типизированными выражениями, содержащими *ссылки на столбец*, за исключением случая, если эти последние также появляются в спецификациях объединения в группы (см. главу 13).

Рассмотрим каждую из этих агрегатных функций и их использование для ответа на запрос.

Подсчет строк и значений в COUNT

Стандарт SQL определяет два варианта функции COUNT. COUNT(*) обрабатывает строки в наборе результатов, а COUNT (*типизированное выражение*), обрабатывает значения, возвращенные типизированным выражением.

COUNT(*) используется для определения количества строк в наборе результатов. Функция COUNT(*) подсчитывает *все* строки в набор результатов, включая избыточные строки, содержащие значения Null. Вот простой пример вопросов, на которые можно получить ответ с помощью этой функции.

> **Внимание!** В этой главе используется метод "Запрос/Преобразование/Уточнение/SQL", введенный в главе 4. Во всех примерах предполагается, что концепции, рассмотренные в предыдущих главах, тщательно изучены и поняты, особенно главы по JOIN и подзапросам.

"Show me the total number of employees we have in our company".
("Показать общее количество сотрудников в нашей компании".)

Преобразование: Select the count of employees from the employees table
(Выбрать количество сотрудников из таблицы "Сотрудники")

Уточнение: Select ~~the~~ count ~~of employees~~ (*) from ~~the~~ employees ~~table~~
(Выбрать количество (*) из "Сотрудники")

SQL
```
SELECT COUNT (*)
FROM Employees
```

Здесь в уточнении используется "*" для указания того, что требуется подсчитать все строки в таблице Employees. Эта практика полезна при работе с запросами подобного типа, потому что помогает гарантировать, что используется правильная функция COUNT. Оператор SELECT в этом примере формирует набор результатов, состоящий из строки из одного столбца, содержащей цифровое значение, которое представляет общее количество строк в таблице Employees.

Ограничения на количество строк, обрабатываемых функцией COUNT(*), отсутствует. Строки, которые должны подсчитываться функцией COUNT(*), указываются использованием условия WHERE. Например, вот как определяется оператор

SELECT, который подсчитывает все строки в таблице Employees для сотрудников, проживающих в штате Вашингтон:

```
SQL                SELECT COUNT(*)
                   FROM Employees
                   WHERE EmpState = 'WA'
```

Условие WHERE можно использовать, чтобы отфильтровать строки или значения, обработанные агрегатной функцией.

Когда агрегатная функция используется в операторе SELECT, можно увидеть или не увидеть имя столбца в наборе результата для возвращаемого значения функции. Некоторые системы базы данных обеспечивают имя столбца по умолчанию, а другие нет. Но можно воспользоваться опцией AS синтаксиса функции, чтобы обеспечить значащее имя столбца для набора результата. Вот как можно применить эту опцию для предыдущего примера:

```
SQL                SELECT COUNT(*) AS TotalWashingtonEmployees
                   FROM Employees
                   WHERE EmpState = 'WA'
```

Теперь набор результатов состоит из столбца TotalWashingtonEmployees, который содержит возвращаемое значение функции COUNT(*). Как показывает синтаксическая диаграмма на рис. 12.1, этот метод можно применить к любой агрегатной функции.

Функция COUNT (*Типизированное выражение*) используется для подсчета общего количества *не-Null* значений, возвращаемых типизированным выражением. (Это выражение более широко известно как COUNT, и это имя мы будем использовать в остальной части книги.) Она подсчитывает все значения, возвращенные типизированным выражением, независимо от того, является ли оно уникальным или повторяющимся, и автоматически исключает все значения Null из окончательного подсчета. Для ответа на запросы такого типа можно использовать COUNT.

> *"How many customers were able to indicate which county they live in?"*
> *("Сколько клиентов смогли указать, в каком округе они проживают?")*

Здесь необходимо определить, сколько реальных значений существует в столбце указания округа. Поскольку COUNT(*) *включает* также значения Null, она не обеспечит правильный ответ. Вместо нее используйте функцию COUNT и преобразуйте запрос следующим образом:

Преобразование: Select the count of non-null county values
as NumberOfKnownCounties from the customers table
(Выбрать количество не-Null-значений для округа
как NumberOfKnownCounties из таблицы "Клиенты")

Уточнение: Select ~~the~~ count ~~of non-null~~ (county) ~~values~~
as NumberOfKnownCounties from ~~the~~ customers ~~table~~

(Выбрать количество (округ)
как NumberOfKnownCounties из “Клиенты”)

SQL

```
SELECT COUNT(CustCounty)
    AS NumberOfKnownCounties
FROM Customers
```

Обратите внимание, что “Перевод” и “Уточнение” явно запрашивают значения не-Null. Хотя уже известно, что эта функция обрабатывает только не-Null значения, следует добавить этот запрос к обоим операторам, чтобы быть уверенным, что используется правильная функция COUNT. Оператор SELECT, определенный здесь, будет генерировать набор результатов, который содержит числовое значение, представляющее общее количество наименований округов, обнаруженных в столбце CustCounty.

Вспомните, что функция COUNT интерпретирует повторяющиеся наименования округа, как если бы они были уникальными, и включает каждый из них в окончательный подсчет. Однако можно воспользоваться опцией DISTINCT функции для исключения повторяющихся значений из итоговой суммы. В следующем примере показано, как можно ее применить:

“How many unique county names are there in the customers table?”
(“Сколько уникальных названий округов имеется в таблице Клиенты?”)

Преобразование:
Select the count of unique county names
as NumberOfUniqueCounties from the customers table
(Выбрать количество уникальных названий округов,
как NumberOfUniqueCounties из таблицы “Клиенты”)

Уточнение:
Select ~~the~~ count ~~of unique~~ (distinct county) names
as NumberOfUniqueCounties from ~~the~~ customers ~~table~~
(Выбрать подсчет (уникальных округов)
как NumberOfUniqueCounties из “Клиенты”)

SQL

```
SELECT COUNT (DISTINCT CustCounty)
    AS NumberOfUniqueCounties
FROM Customers
```

Когда используется опция DISTINCT, база данных извлекает все не-Null-значения из столбца для округа, исключает повторы, а *затем* подсчитывает оставшиеся значения. База данных тщательно разбирается в большинстве этих же процессов, когда DISTINCT используется с функциями SUM, AVG, MIN или MAX.

В следующем примере используется немного измененная версия предыдущего запроса, чтобы показать, что к функции COUNT можно применять фильтр:

“How many unique county names are there in the customers table for the state of Oregon?”

("Сколько уникальных названий округов имеется в таблице "Клиенты" для штата Орегон?")

Преобразование: Select the count of unique county names as NumberOfUniqueOregonCounties from the customers table where the state is 'OR'
(Выбрать количество уникальных названий округов как NumberOfUniqueOregonCounties из таблицы "Клиенты", где штатом является 'OR')

Уточнение: Select ~~the~~ count ~~of unique~~ (distinct county) as NumberOfUniqueOregonCounties ~~names~~ from ~~the~~ customers ~~table~~ where ~~the~~ state ~~is~~ = 'OR'
(Выбрать подсчет (округ по одному) как NumberOfUniqueOregonCounties из "Клиенты", где штат = 'OR')

SQL
```
SELECT COUNT (DISTINCT CustCounty)
    AS NumberOfUniqueOregonCounties
FROM Customers
WHERE CustState = 'OR'
```

Использование DISTINCT с COUNT(*) *не допускается*. Это — разумное ограничение, поскольку COUNT(*) подсчитывает *все* строки в таблице независимо от того, являются ли они избыточными или содержат значения Null.

Подсчет общей суммы в SUM

С помощью функции SUM можно вычислить общую сумму *числовых* типизированных выражений. Она обрабатывает все не-Null-значения типизированного выражения и возвращает окончательную итоговую сумму для набора результатов. Если типизированное выражение содержит во *всех* строках значение Null или условия FROM и WHERE вместе возвращают пустой набор результатов, то SUM возвращает Null. Вот пример запроса, на который можно ответить, используя SUM:

"What is the total amount we pay in salaries to our employees in California?"
("Какая общая сумма выплачивается в виде зарплаты нашим сотрудникам в Калифорнии?")

Преобразование: Select the sum of salary as TotalSalaryAmount from the employees table where the state is 'CA'
(Выбрать сумму заработной платы как TotalSalaryAmount из таблицы "Сотрудники", где штатом является 'CA')

Уточнение: Select ~~the~~ sum ~~of~~ (salary) as TotalSalaryAmount from ~~the~~ employee ~~table~~ where ~~the~~ state ~~is~~ = 'CA'
(Выбрать сумму (заработная плата) как TotalSalaryAmount из "Сотрудники", где штат = 'CA')

SQL
```
SELECT SUM(Salary) AS TotalSalaryAmount
FROM Employees
Where EmpState = 'CA'
```

Используемое здесь типизированное выражение было простой ссылкой на столбец. Однако также можно использовать SUM к типизированному выражению, состоящему из числового выражения:

"How match is our current inventory worth?"
("В какую сумму оцениваются запасы наших товаров?")

Преобразование: Select the sum of wholesale price times quantity on hand as TotalInventoryValue from the products table
(Выбрать сумму оптовой цены, умноженной на имеющееся количество, как TotalInventoryValue из таблицы "Товары")

Уточнение: Select ~~the~~ sum ~~of~~ (wholesale price ~~times~~ * quantity on hand) as TotalInventoryValue from ~~the~~ products ~~table~~
(Выбрать сумму (оптовая цена * имеющееся количество) как TotalInventoryValue из "Товары")

SQL
```
SELECT SUM(WholesalePrice * QuantityOnHand)
    AS TotalInventoryValue
FROM Products
```

Как известно, строка должна содержать реальные значения в столбцах WholesalePrice и QuantityOnHand, чтобы их можно было обрабатывать в функции SUM. В данном случае база данных обрабатывает выражение для всех уточняющих строк в таблице Products, подсчитывая общую сумму результатов в функции SUM, а затем отправляя итоговую сумму в набор результата.

Вот пример использования SUM для вычисления итоговой суммы для уникального множества числовых значений:

"Calculate a total of all unique wholesale costs for the products we sell".
("Вычислить общую сумму всех уникальных оптовых затрат на проданные товары".)

Преобразование: Select the sum of unique wholesale costs as SumOfUniqueWholesaleCosts from the products table
(Выбрать сумму уникальных оптовых затрат как SumOfUniqueWholesaleCosts из таблицы "Товары")

Уточнение: Select ~~the~~ sum ~~of unique~~ (distinct wholesale costs) as SumOfUniqueWholesaleCosts from ~~the~~ products ~~table~~ (Выбрать сумму (неповторяющихся оптовых затрат) как SumOfUniqueWholesaleCosts из "Товары")

SQL
```
SELECT SUM(DISTINCT WholesaleCost)
    AS SumOfUniqueWholesaleCosts
FROM Products
```

Вычисление среднего значения в AVG

Другой функцией, которую можно использовать с *числовыми* значениями, является AVG. Она вычисляет арифметические среднее всех не-Null-значений, возвращенных типизированным выражением. AVG можно использовать для ответа на запросы следующего типа:

"What is the average contract amount for vendor number 10014?"
("Какова средняя сумма контракта для поставщика номер 10014?")

Преобразование: Select the average of contract price as AverageContractPrice from the vendor contracts table where the vendor ID is 10014 (Выбрать среднее значение цены контракта как AverageContractPrice из таблицы "Контракты поставщика", где идентификатор поставщика — 10014)

Уточнение: Select ~~the average of~~ avg (contract price) as AverageContractPrice from ~~the~~ vendor contracts ~~table~~ where the vendor ID is = 10014 (Выбрать avg (цена контракта) как AverageContractPrice из "Контракты поставщика", где идентификатор поставщика = 10014)

SQL
```
SELECT AVG(ContractPrice)
    AS AverageContractPrice
FROM Vendor_Contracts
WHERE VendorID = 10014
```

При работе с уточнением обязательно вычеркните слово "average" и замените его на "avg". Это позволит воздержаться от случайного использования "Average" в условии SELECT. Average не является допустимым ключевым словом SQL, поэтому оператор SELECT даст сбой при попытке его использования.

Функция AVG может использоваться для обработки нескольких выражений точно так же, как функция SUM. Помните, что невозможно использовать AVG в типизированном выражении, которое не является числовым. Большинство систем БД выдают ошибку при попытке использования этих функций с символьной строкой или данными типа дата/время.

"What is the average item total for order 64?"
("Какова средняя стоимость товара в заказе 64?")

Преобразование: Select the average of price times quantity ordered as AverageItemTotal from the order details table where order ID is 64
(Выбрать среднюю цену, умноженную на количество, заказанное как AverageItemTotal из таблицы "Детали заказа", где идентификатор заказа равняется 64)

Уточнение: Select ~~the average of~~ avg (price ~~times~~ * quantity ordered) as AverageItemTotal from ~~the~~ order details ~~table~~ where order ID is = 64
(Выбрать avg (цена * заказанное количество) как AverageItemTotal из "Детали заказа", где идентификатор заказа = 64)

SQL
```
SELECT AVG(Price * QuantityOrdered)
    AS AverageItemTotal
FROM Order_Details
WHERE OrderID = 64
```

Имейте в виду, что строка должна содержать реальные значения в столбцах Price (Цена) и QuantityOrdered (Заказанное количество), чтобы строки обрабатывались функцией AVG.

В противном случае числовое выражение оценивается в Null, а функция AVG игнорирует всю строку целиком. Как и с SUM, если типизированное выражение во всех строках является Null или условия FROM и WHERE вместе возвращают пустой набор результата, то AVG возвращает значение Null.

В следующем примере опция DISTINCT используется для усреднения уникального множества числовых значений:

"Calculate an average of all unique product prices".
("Подсчитать среднее значение цен отдельных товаров".)

Преобразование: Select the average of unique prices as UniqueProductPrices from the products table
(Выбрать среднее значение уникальных цен как UniqueProductPrices из таблицы "Товары")

Уточнение: Select ~~the average of unique~~ avg (distinct prices) as UniqueProductPrices from ~~the~~ products ~~table~~
(Выбрать avg (уникальных цен) как UniqueProductPrices из "Товары")

SQL
```
SELECT AVG(DISTINCT Price) AS UniqueProductPrices
FROM Products
```

Поиск наибольшего значения в MAX

Наибольшее значение, возвращенное типизированным выражением, можно определить с помощью функции MAX. Функция MAX может обрабатывать любые типы данных. Возвращаемое значение зависит от обрабатываемых данных.

Символьные строки	Значение, возвращаемое MAX, основывается на последовательности сортировки, используемой системой базы данных или компьютером. Например, если база данных использует множество символов ASCII и нечувствительна к регистру, то имена компаний будут отсортированы следующим образом: "...4th Dimension Productions...Al's Auto Shop...allegheny & associates...Zercon Productions...zorn credit services". В этом случае MAX возвратит "zorn credit services" как значение MAX.
Числа	MAX возвращает наибольшее число.
Дата/Время	MAX оценивает даты и время в хронологическом порядке и возвращает *самую недавнюю* (или самую давнюю) дату и время.

Вот примеры того, как следует использовать MAX для ответа на запрос:

"What is the largest amount we've paid on a contract?"
("Какая наибольшая сумма была уплачена по контракту?")

Преобразование: Select the maximum contract price
as LargestContractPrice from the engagements table
(Выбрать максимальную цену контракта
как LargestContractPrice из таблицы "Ангажементы")

Уточнение: Select ~~the~~ max~~imum~~ (contract price)
as LargestContractPrice from ~~the~~ engagements ~~table~~
(Выбрать максимум (цена контракта)
как LargestContractPrice из "Ангажементы")

SQL

```
SELECT MAX(ContractPrice)
    AS LargestContracrPrice
FROM Engagements
```

"What was the largest line item total for order 3314?"
("Какова была наибольшая сумма для элемента строки в заказе 3314?")

Преобразование: Select the maximum price times quantity ordered as LargestItemTotal from the order details table where the order ID is 3314
(Выбрать максимум цены, умноженной на заказанное количество, как LargestItemTotal из таблицы "Детали заказа", где идентификатор заказа равен 3314)

Уточнение: Select ~~the~~ max~~imum~~ (price ~~times~~ * quantity ordered) as LargestItemTotal from ~~the~~ order details ~~table~~ where ~~the~~ order ID ~~is~~ = 3314
(Выбрать максимум (цена * заказанное количество) как LargestItemTotal из "Детали заказа", где идентификатор заказа = 3314)

SQL
```
SELECT MAX(Price * QuantityOrdered)
    AS LargestItemTotal
FROM Order_Details
WHERE OrderID = 3314
```

В следующем примере используется опция DISTINCT для возврата неповторяющихся экземпляров самой недавней даты проверки в таблице Staff. В этом случае проверка двух или нескольких штатных сотрудников могла быть проведена не позже прошлого четверга, но нам необходимо получить только одно появление этой даты.

"What is the most recent date that we reviewed any of our stuff?"
("Указать самую последнюю дату проверки любого из наших штатных сотрудников".)

Преобразование: Select the maximum unique review date as MostRecentReviewDate from the staff table
(Выбрать максимальную уникальную дату проверки как MostRecentReviewDate из таблицы "Персонал")

Уточнение: Select ~~the~~ max~~imum unique~~ (distinct review date) as MostRecentReviewDate from ~~the~~ staff ~~table~~
(Выбрать max (неповторяющуюся дату проверки) как MostRecentReviewDate из "Персонал")

SQL
```
SELECT MAX(DISTINCT ReviewDate)
    AS MostRecentReviewDate
FROM Staff
```

Хотя стандарт SQL определяет DISTINCT как опцию для функции MAX, DISTINCT *не оказывает никакого влияния* на функцию MAX. Максимальное значение может быть только одно, независимо от того, отличается оно от других или нет. Например, оба приведенные ниже выражения возвращают одно и то же значение:

```
SELECT MAX(HireDate) FROM Agents
SELECT MAX(DISTINCT HireDate) FROM Agents
```

Мы представили оба варианта функции, поскольку они являются частью текущего стандарта SQL, но рекомендуем использовать только функцию MAX без опции DISTINCT.

Поиск наименьшего значения с использованием MIN

Функция MIN, позволяет определить *наименьшее* значение, возвращенное типизированным выражением. Она работает подобно функции MAX, но возвращает противоположное значение: первый символ в строке (на основании последовательности сортировки), наименьшее число и самую недавнюю дату или время.

Используя функцию MIN, можно ответить на запросы, подобные приведенным ниже:

"What is the lowest price we charge for a product?"
("Указать наименьшую цену, уплаченную за товар?")

Преобразование: Select the minimum price as LowestProductPrice
from the products table
(Выбрать минимальную цену как LowestProductPrice
из таблицы "Товары")

Уточнение: Select ~~the~~ mini~~mum~~ (price) as LowestProductPrice
from ~~the~~ products ~~table~~
(Выбрать min(цена) как LowestProductPrice
из "Товары")

SQL

```
SELECT MIN(Price)
    AS LowestProductPrice
FROM Products
```

"What was the lowest line item total for order 3314?"
("Какова была наименьшая сумма для элемента строки в заказе 3314?")

Преобразование: Select the minimum price times quantity ordered
as LowestItemTotal from the order details table
where the order ID is 3314
(Выбрать минимум цены, умноженной на заказанное
количество, как LowestItemTotal из таблицы "Детали
заказа", где идентификатор заказа равен 3314)

Уточнение: Select ~~the~~ mini~~mum~~ (price ~~times~~ * quantity ordered) as LowestItemTotal from ~~the~~ order details ~~table~~ where ~~the~~ order ID is = 3314
(Выбрать минимум (цена * заказанное количество) как LowestItemTotal из "Детали заказа", где идентификатор заказа = 3314)

SQL
```
SELECT MIN(Price * QuantityOrdered)
    AS LowestItemTotal
FROM Order_Details
WERE OrderID = 3314
```

В следующем примере используется опция DISTINCT для возвращения уникального экземпляра самой ранней даты приема на работу в таблице Employee. В этом случае два или несколько сотрудников могут быть приняты на работу 16 мая 1977 г., но нам нужно видеть только один экземпляр этих данных:

"When did we hire our first employees?"
("Когда были приняты на работу самые первые сотрудники?")

Преобразование: Select the minimum unique hire date as EarliestHireDate from the employees table
(Выбрать минимальную уникальную дату приема на работу как EarliestHireDate из таблицы "Сотрудники")

Уточнение: Select ~~the~~ mini~~mum unique~~ (distinct hire date) as EarliestHireDate from ~~the~~ employees ~~table~~
(Выбрать min (уникальную дату приема на работу) как EarliestHireDate из "Сотрудники")

SQL
```
SELECT MIN(DISTINCT HireDate)
    AS EarliestHireDate
FROM Employees
```

Важно отметить, что опция DISTINCT *не оказывает какого-либо влияния* на функцию MIN, как и на функцию MAX. Может существовать только одно минимальное значение независимо от того, повторяется оно или нет. Например, оба следующих выражения возвращают одинаковое значение:

```
SELECT MIN(ReviewDate) FROM Agents
SELECT MIN(DISTINCT ReviewDate) FROM Agents
```

Мы представили оба варианта функции, потому что они являются частью текущего стандарта SQL, но рекомендуем использовать только функцию MIN без опции DISTINCT.

Использование нескольких функций

Можно использовать несколько агрегатных функций одновременно. Это позволяет показать сравниваемую информацию, используя один оператор SELECT. Например, можно использовать функции MIN и MAX, чтобы показать самую давнюю и самую последнюю дату заказов для конкретного клиента, или функции MAX, MIN и AVG, показывающие наибольший, наименьший и средний баллы для данного студента. Вот другие примеры использования двух или более агрегатных функций:

"Show me the earliest and most recent review dates for the employees in the advertising department".
("Показать самую давнюю и самую последнюю дату проверок сотрудников рекламного отдела".)

Преобразование: Select the minimum review date as EarliestReviewDate and the maximum review date as RecentReviewDate from the employees table where the department is "Advertising"
(Выбрать минимальную дату проверки как EarliestReviewDate и максимальную дату проверки как RecentReviewDate из таблицы "Сотрудники", где отдел — "Рекламный")

Уточнение: Select ~~the~~ min~~imum~~ review date as EarliestReviewDate ~~and the~~ max~~imum~~ review date as RecentReviewDate from ~~the~~ employees ~~table~~ where ~~the~~ department ~~is~~ = "Advertising"
(Выбрать min дату проверки как EarliestReviewDate и max дату проверки как RecentReviewDate из "Сотрудники", где отдел = "Рекламный")

SQL

```
SELECT MIN(ReviewDate) AS EarliestReviewDate,
    MAX(ReviewDate) AS RecentReviewDate
FROM Employees
WHERE Department = "Advertising"
```

"How many different products where ordered on order number 553 and what was the total cost of that order?"
("Сколько различных товаров было заказано в заказе номер 553 и какова общая стоимость этого заказа?")

Преобразование: Select the count of product ID as TotalProductsPurchased and the sum of price times quantity ordered as OrderAmount from the order details table where the order number is 553
(Выбрать количество идентификаторов товара как TotalProductsPurchased и сумму цены, умноженной на количество, заказанное как OrderAmount, из таблицы "Детали заказа", где номер заказа — 553)

Уточнение: Select the count of (product ID) as TotalProductsPurchased and the sum of (price times * quantity ordered) as OrderAmount from the order details table where the order number is = 553
(Выбрать количество (идентификатор товара) как TotalProductsPurchased и сумму (цена * заказанное количество) как OrderAmount из "Детали заказа", где номер заказа = 553)

SQL
```
SELECT COUNT(ProductID) AS
    Total ProductsPurchased, SUM(Price *
    QuantityOrdered) AS OrderAmount
FROM Order_Details
WHERE OrderNumber = 553
```

Существуют два ограничения, которые необходимо учитывать при работе с двумя или более агрегатных функций. Первое состоит в том, что не допускается вложение одной агрегатной функции в другую. Это ограничение делает недействительным следующее выражение:

```
SUM(AVG(LineItemTotal))
```

Второе ограничение не допускает использования подзапроса как типизированного выражения для агрегатной функции. Например, в соответствии с этим ограничением следующее выражение является недействительным:

```
AVG((SELECT Price FROM Products WHERE Category = 'Bikes'))
```

Теперь рассмотрим, как можно использовать агрегатную функцию для фильтрации информации в наборе результата.

Использование агрегатных функций в фильтрах

Поскольку агрегатная функция возвращает отдельное значение, ее можно использовать как часть предиката сравнения в условии поиска. Однако ее следует помещать в подзапрос, а затем использовать этот подзапрос как часть предиката сравнения. Мы уже изучали использование подзапроса как части условия поиска в условии WHERE, а агрегатной функции — в подзапросе. Теперь расширим эти знания.

Использование агрегатной функции как части предиката сравнения позволяет сравнивать значение типизированного выражения с отдельным статистическим значением. Хотя для задачи можно использовать литеральное значение, подзапрос предоставляет большую гибкость и обеспечивает более динамический подход к условию. Предположим, например, что к базе данных обращаются со следующим запросом:

"List the engagement numbers that have a contract price greater than or equal to the overall average contract price".
("Привести список номеров ангажементов, для которых цена контракта больше или равна общей средней цене контракта".)

Один из методов, который можно применить для ответа на этот запрос, состоит в вычислении общей средней цены контракта вручную, а затем использовании этого значения в предикате сравнения.

Преобразование: Select the engagement number from the engagements table where the contract price is greater than or equal to $24,887.00
(Выбрать номер ангажемента из таблицы "Ангажементы", где цена контракта больше или равна 24887,00 долларов)

Уточнение: Select ~~the~~ engagement number from ~~the~~ engagements ~~table~~ where ~~the~~ contract price ~~is greater than or equal to~~ >= $24,887.00
(Выбрать номер ангажемента из "Ангажементы", где цена контракта >= 24 887,00)

SQL
```
SELECT EngagementNumber
FROM Engagements
WHERE ContractPrice >= 24,887.00
```

Другой способ решения этой задачи — использование агрегатной функции в подзапросе и предоставление системе базы данных возможности выполнить всю работу за вас.

Преобразование: Select the engagement number from the engagements table where the contract price is greater than or equal to the overall average contract price in the engagements table
(Выбрать номер ангажемента из таблицы "Ангажементы", где цена контракта больше или равна общему среднему значению цены контракта в таблице "Ангажементы")

Уточнение: Select ~~the~~ engagement number from ~~the~~ engagements ~~table~~ where ~~the~~ contract price ~~is greater than or equal to the~~ >= ~~overall average~~ (select avg contract price ~~in the~~ from engagements ~~table~~)
(Выбрать номер ангажемента из "Ангажементы", где цена контракта >= (Выбрать avg цены контракта из "Ангажементы"))

```
SQL                SELECT EngagementNumber
                   FROM Engagements
                   WHERE ContractPrice >=
                       (SELECT AVG(ContractPrice)
                       FROM Engagements)
```

Использование подзапросов в агрегатной функции является наилучшим образом действий. Если используется литеральное значение, следует быть уверенным, что средняя цена контракта всегда пересчитывается до выполнения оператора SELECT в случае изменения любых существующих цен контрактов. Затем следует убедиться, что значение в предикат сравнения вводится правильно. Но, если используется подзапрос, то ни о чем из сказанного не следует беспокоиться. Функция AVG всегда оценивается при выполнении оператора SELECT и всегда возвращает правильное значение независимо от того, изменились ли какие-либо цены контрактов (это справедливо для любой агрегатной функции, используемой в подзапросе).

Используя в подзапросе условие WHERE, можно ограничить строки, оцениваемые агрегатной функцией. Это позволяет сузить область статистического значения, возвращаемого агрегатной функцией. Рассмотрим пример применения этой методики.

"List the engagement number and contract price of all engagements that have a contract price larger than the total amount of all contract prices for the entire month of May in 1999".
("Привести список номеров ангажементов и цену контрактов всех ангажементов, у которых цена контракта больше общей суммы всех цен контрактов за месяц май 1999 г.".)

Преобразование: Select engagement number and contract price from the engagements table where the contract price is greater than the sum of all contract prices of engagements dated between May 1, 1999 and May 31, 1999
(Выбрать номер ангажемента и цену контракта из таблицы "Ангажементы", где цена контракта больше суммы всех цен контрактов для ангажементов на период с 1 мая 1999 по 31 мая 1999)

Уточнение: Select engagement number, ~~and~~ contract price from ~~the~~ engagements ~~table~~ where ~~the~~ contract price ~~is greater than~~ > ~~the~~ (select sum ~~of all~~ (contract prices) ~~of~~ from engagements where ~~dated~~ start date between ~~May 1, 1999~~ '1999-05-01' and '1999-05-31') ~~May 31,1999~~
(Выбрать номер ангажемента, цену контракта из "Ангажементы", где цена контракта > (Выбрать сумму (цена контрактов) из "Ангажементы", где дата начала между '1999-05-01' и '1999-05-31')

```
SQL                SELECT EngagementNumber, ContractPrice
                   FROM Engagements
                   WHERE ContractPrice >
                      (SELECT SUM(ContractPrice) FROM Engagements
                         WHERE StartDate BETWEEN '1999-05-01'
                         AND '1999-05-31')
```

Необходимость использовать агрегатные функции в фильтрах возникает редко, но они определенно подходят, когда требуется ответить на случайные, "вне рамок", запросы.

Примеры операторов

Рассмотрим некоторые примеры применения агрегатных функций, используя таблицы из каждого примера баз данных.

> **Внимание!** Все имена столбцов и таблиц, используемых в этих примерах, взяты из структур учебных баз данных, представленных в приложении В. Для упрощения процесса во всех последующих примерах этапы преобразования и уточнения объединены.

База данных заказов на закупки

"How many customers do we have in the state of California?"
("Сколько у нас клиентов в штате Калифорния?")

Преобразование/ Уточнение:
Select ~~the~~ count(*) ~~of all customers~~ from ~~the~~ customers ~~table~~ where the state ~~is~~ = 'CA'
(Выбрать count(*) из "Клиенты", где штат = 'CA')

```
SQL                SELECT COUNT(*) AS NumberOfCACustomers
                   FROM Customers
                   WHERE CustState = 'CA'
```

Number_Of_California_Customers (1 строка)

NumberOfCACustomers
7

"List the product names and numbers that have a quoted price greater than or equal to the overall average retail price in the products table".
("Привести список наименований и номеров товаров, объявленная цена которых больше или равна общей средней розничной цене в таблице "Товары".)

Преобразование/ Уточнение:

Select ~~the~~ product name, ~~and the~~ product number from ~~the~~ products ~~table~~ join~~ed with the~~ order_details ~~table~~ on product number where ~~the~~ quoted price >= ~~is greater than or equal to the average~~ (select avg(retail price) from ~~in the~~ products ~~table~~)
(Выбрать наименование товара, номер товара из “Товары”, соединенной с order_details по номеру товара, где объявленная цена >= (Выбрать avg(розничная цена) из “Товары”)

SQL

```
SELECT DISTINCT Products.ProductName,
    Order_Details.ProductNumber
FROM Products
INNER JOIN Order_Details
ON  Products.ProductNumber =
    Order_Details.ProductNumber
WHERE Order.Details.QuotedPrice >=
    (SELECT AVG(RetailPrice)
    FROM Products)
```

Внимание! Мы предпочли запросить DISTINCT (неповторяющиеся) товары, потому что конкретный товар может быть заказан более одного раза, а нам нужно показать наименование и номер каждого товара только один раз.

Quoted_Price_vs_Average_Retail_Price (4 строки)

ProductName	ProductNumber
Eagle FS-3 Mountain Bike	2
GT RTS-2 Mountain Bike	11
Trek 9000 Mountain Bike	1
Viscount Mountain Bike	6

База данных эстрадных мероприятий

“List the engagement number and contract price of our earliest contracts”.
(“Привести номер ангажемента и цену контракта из наших самых первых контрактов”.)

Преобразование/ Уточнение:

Select engagement number, ~~and~~ contract price from ~~the~~ engagements ~~table~~ where ~~the~~ start date = ~~is equal to the earliest~~ (select min(start date) ~~in the~~ from engagements ~~table~~)
(Выбрать номер ангажемента, цену контракта из “Ангажементы”, где дата начала = (Выбрать min (дата начала) из “Ангажементы”)

SQL

```
SELECT EngagementNumber, ContractPrice
FROM Engagements
WHERE StartDate =
    (SELECT MIN(StartDate) FROM Engagements)
```

Earliest_Contracts (2 строки)

EngagementNumber	ContractPrice
2	$200.00
8	$1,850.00

"What was the total value of all engagements booked in August of 1999?" ("Какова общая стоимость всех ангажементов, заявленных на август 1999?")

Преобразование/ Уточнение:

Select ~~the~~ sum ~~of~~ (contract price) as TotalBookedValue
from ~~the~~ engagements ~~table~~ where ~~the~~ start date ~~is~~
between '1999-08-01' and '1999-08-31' ~~August 1, 1999 and August 31, 1999~~
(Выбрать sum (цена контракта) как TotalBookedValue
из "Ангажементы", где дата начала между '1999-08-01'
и '1999-08-31')

SQL

```
SELECT SUM(ContractPrice)
    AS TotalBookedValue
FROM Engagements
WHERE StartDate
    BETWEEN '1999-08-01'
    AND '1999-08-31'
```

Total_Booked_Value_For_August_1999 (1 строка)

TotalBookedValue
$ 30,005.00

База данных расписания занятий

"What is the largest salary we pay to any staff member?" ("Указать наибольшую заработную плату, уплаченную любому штатному сотруднику".)

Преобразование/ Уточнение:

Select ~~the~~ max~~imum~~ (salary) as LargestStaffSalary
from ~~the~~ staff ~~table~~
(Выбрать max (зарплата) как LargestStaffSalary
из "Персонал")

SQL

```
SELECT Max(Salary)
    AS LargestStaffSalary
FROM Staff
```

Largest_Staff_Salary (1 строка)

LargestStaffSalary
$ 60,000.00

"What is the total salary amount paid to our staff in California?"
("Указать общую сумму зарплаты, выплаченную нашему персоналу в Калифорнии".)

Преобразование/ Уточнение:
Select ~~the~~ sum ~~of~~ (salary) as TotalSalaryAmount from ~~the~~ staff ~~table for all our California staff~~ where state = 'CA'
(Выбрать max (зарплата) как TotalSalaryAmount из "Персонал", где штат = 'CA')

SQL

```
SELECT SUM(Salary)
    AS TotalSalaryAmount
FROM Staff
WHERE StfState = 'CA'
```

Total_Salary_Paid_To-California_Staff (1 строка)

TotalAmountPaid
$209,000.00

База данных лиги игры в боулинг

"How many tournaments have been played at Red Rooster Lanes?"
("Сколько турниров сыграно на Ред Рустер Лейнс?")

Преобразование/ Уточнение:
Select ~~the~~ count ~~of~~ (tourney location)s as NumberOfTournaments from ~~the~~ tournaments ~~table~~ where ~~the~~ tourney location ~~is~~ = 'Red Rooster Lanes'
(Выбрать count(место проведения турнира) как NumberOfTournaments из "Турниры", где место проведения турнира = 'Red Rooster Lanes')

SQL

```
SELECT COUNT(TourneyLocation)
    AS NumberOfTournaments
FROM Tournaments
WHERE TourneyLocation = 'Red Rooster Lanes'
```

Number_Of_Tournaments_At_Red_Rooster_Lanes (1 строка)

NumberOfTournaments
2

"List the last name and first name, in alphabetical order, of every bowler whose personal average score is greater than or equal to the overall current average score".
("Привести список фамилий и имен, в алфавитном порядке, всех игроков в боулинг, у которых личное среднее количество очков больше или равно общему текущему среднему количеству очков".)

Преобразование/ Уточнение:

Select ~~the~~ last name, ~~and~~ first name from ~~the~~ bowlers ~~table~~ where ~~the~~ current average >= ~~is greater than or equal to the overall~~ (select avg(current average) ~~score in the~~ from bowlers) ~~table~~
(Выбрать фамилию, имя из “Игроки в боулинг”, где текущее среднее >+ (Выбрать avg(текущее среднее) из “Игроки в боулинг”))

SQL

```
SELECT BowlerLastName,
    BowlerFirstName
FROM Bowlers
WHERE CurrentAverage >=
    (SELECT AVG(CurrentAverage) FROM Bowlers)
ORDER BY BowlerLastName, BowlerFirstName
```

Better_Than_Overall_Average (16 строк)

BowlerLastName	BowlerFirstName
Cunningham	David
Fournier	David
Hallmark	Alaina
Hallmark	Gary
Hernandez	Michael
Kennedy	John
McLain	Susan
Patterson	Kathryn
Patterson	Neil
Patterson	Rachel
Piercy	Greg
Pundt	Steve
Thompson	Mary
Thompson	Will
Viescas	Carol
Viescas	John

База данных рецептов

"How many recipes contain a beef ingredient?"
("Сколько рецептов содержат говядину?")

Преобразование/ Уточнение:

Select ~~the~~ count(*) ~~of recipes~~ as NumberOfRecipes from ~~the~~ recipes ~~table~~ where ~~the~~ recipe ID ~~is~~ in ~~the~~ (selecti~~on of~~ recipe ID~~s in the~~ from recipe ingredients ~~table~~ joine~~d with~~ ~~the~~ inner join ingredients ~~table~~ on ingredient ID where ~~the~~ ingredient name ~~is~~ like 'Beef')
(Выбрать count(*) как NumberOfRecipes из "Рецепты", где идентификатор рецепта в (Выбрать идентификатор рецепта из "Компоненты рецепта" с внутренним соединением с "Компоненты" по идентификатору компонента, где название компонента совпадает с шаблоном 'говядина')

SQL

```
SELECT COUNT(*) AS NumberOfRecipes
FROM Recipes
WHERE Recipes.RecipeID IN
    (SELECT RecipeID
     FROM Recipe_Ingredients
   INNER JOIN Ingredients ON
     Recipe_Ingredients.IngredientID =
     Ingredients.Ingredient ID
 WHERE Ingredients.IngredientName
     LIKE 'Beef%')
```

Recipes_With_Beef_Ingredient (1 строка)

NumberOfRecipes
3

"How many ingredients are measured by the cup?"
("Сколько компонентов измеряются чашкой?")

Преобразование/ Уточнение:

Select ~~the~~ count(*) ~~of ingredients~~ as NumberOfIngredients from ~~the~~ ingredients ~~table~~ joine~~d with the~~ inner join measurements ~~table~~ on measureamount ID where ~~the~~ measurement description ~~is~~ ='Cup'
(Выбрать count(*) как NumberOfIngredients, из "Компоненты", внутренне соединенной с "Единицы измерения" по идентификатору единиц измерения, где описание единиц измерения = 'чашка')

SQL

```
SELECT COUNT (*) AS NumberOfIngredients
FROM Ingredients
INNER JOIN Measurements ON
    Ingredients.MeasureAmountID =
    Measurements.MeasureAmountID
WHERE MeasurementsDescription = 'Cup'
```

Number_of_Ingredients_Measured_by_the_Cup (1 строка)

NumberOfIngredients
12

Итоги

В данной главе были представлены агрегатные функции. Существует шесть различных функций, и их можно использовать в условиях SELECT и WHERE оператора SELECT. Все агрегатные функции — за исключением COUNT(*) — при выполнении своих операций игнорируют все значения Null.

Подсчет элементов выполняется с использованием функции COUNT, самые наибольшие и самые наименьшие значения находят, используя функции MAX и MIN, среднее значение вычисляется с помощью функции AVG, общая сумма множества значений определяется в функции SUM. В каждой функции можно использовать опцию DISTINCT, но она не оказывает влияния на функции MAX и MIN.

В конце главы показано применение агрегатных функций в фильтрах. Агрегатную функцию следует включить в подзапрос, а затем использовать этот подзапрос как часть фильтра. Фильтр можно применять к подзапросу, а значение агрегатной функции определяется конкретным множеством данных.

Задачи для самостоятельного решения

Ниже приведены формулировки запросов и имена решений этих запросов в учебных базах данных. Попрактикуйтесь немного и разработайте SQL для каждого запроса, а затем проверьте свой ответ по запросу, который сохранен нами в этих БД. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов,— важно, чтобы набор результатов был тем же.

База данных заказов на закупку

1. *"What is the average retail price of a mountain bike?"*
   ("Какова средняя розничная цена горного велосипеда?")
   Решение можно найти в Average_Price_Of_A_Mountain_Bike (1 строка)

2. *"What was the date of our most recent order?"*
   ("Указать дату самого последнего заказа".)
   Решение можно найти в Most_Recent_Order_Date (1 строка)

3. *"What was the total amount for order number 8?"*
("Какова общая сумма для заказа номер 8?")
Решение можно найти в Total_Amount_For_Order_Number_8 (1 строка)

База данных агентства эстрадных мероприятий

1. *"What is the average salary of a booking agent?"*
("Каков размер средней заработной платы агента по приему заявок?")
Решение можно найти в Average_Agent_Salary (1 строка)

2. *"Show me the engagement numbers for all engagements that have a contract price greater than or equal to the overall average contract price"*.
("Показать номера ангажементов для всех ангажементов, контрактная цена которых больше или равна общей средней цене контракта".)
(*Совет*: Для ответа на этот запрос следует использовать подзапрос.)
Решение можно найти в Contract_Price_>=_Average_Contract_Price (52 строки).

3. *"Which vendors do we work with that don't have a Web site?"*
("У каких поставщиков, с которыми мы работаем, нет Web-сайта?")
Решение можно найти в Number_Of_Bellevue_Entertainers (1 строка)

База данных расписания занятий

1. *"Which is the current average class duration?"*
("Какова средняя продолжительность лекции?")
Решение можно найти в Average_Class_Duration (1 строка).

2. *"List the last name and first name of each staff member who has been with us the longest amount of time"*.
("Привести список фамилий и имен каждого штатного сотрудника, которые работают у нас наиболее давно".)
(*Совет*: Необходимо использовать подзапрос, содержащий агрегатную функцию, которая оценивает столбец DateHired (Дата приема на работу).)
Решение можно найти в Most_Senior_Staff_Members (3 строки).

3. *"How many classes are held in room 3346?"*
("Сколько лекций проводится в аудитории 3346?")
Решение можно найти в Number_Of_Classes_Held_In_Room_3346 (1 строка).

База данных лиги игры в боулинг

1. *"What is the largest handicap held by any bowler at the current time?"*
("Какой наибольший гандикап, который имеет любой игрок в боулинг в настоящее время?")
Решение можно найти в Current_Highest_Handicap (1 строка).

2. *"Which locations hosted the very first tournaments?"*
("В каких местах будут проводиться самые первые турниры?")
(*Совет*: Необходимо определить дату самого первого турнира.)
Решение можно найти в Tourney_Locations_For_Earliest_Dates (2 строки).

3. *"What is the most recent tournament date we have in our schedule?"*
("Какая самая последняя дата турнира в нашем расписании?")
Решение можно найти в Most_Recent_Tournev_Date (1 строка).

База данных рецептов

1. *"Which recipe requires the most cloves of garlic?"*
("В каком рецепте требуется наибольшее количество зубков чеснока?")
(*Совет*: Для ответа на этот запрос необходимо воспользоваться операциями INNER JOIN и подзапросом.)
Решение можно найти в Recipe_With_Most_Cloves_of_Garlic (1 строка).

2. *"Count the number of main course recipes"*.
("Подсчитать количество рецептов основных блюд".)
(*Совет*: Здесь требуется операция JOIN между Recipe_Classes и Recipes.)
Решение можно найти в Number_Of_Main_Course_Recipes (1 строка).

3. *"Calculate the total number of teaspoons of salt in all recipes"*.
("Подсчитать общее количество чайных ложек соли во всех рецептах".)
Решение можно найти в Total_Salt_Used (1 строка).

13

Группирование данных

"Не погрязайте в деталях. Смотрите в целом".

— Маршал Фердинанд Фош,
Главнокомандующий союзных армий во Франции

Вопросы, рассматриваемые в данной главе:

- Зачем нужно группировать данные
- Условие GROUP BY
- Некоторые ограничения
- Применение GROUP BY
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

В главе 12 мы объяснили, как в SQL использовать агрегатные функции (COUNT, MIN, MAX, AND и SUM) для запроса вычисления значения по всем строкам в таблице, определенной в условиях FROM и WHERE. Однако если в условие SELECT включается любое типизированное выражение, содержащее агрегатную функцию, то *все* типизированные выражения должны быть либо литеральными константами, либо содержать агрегатную функцию. Эта особенность полезна, если нужно получить только одну строку итоговых сумм по набору результатов, но что делать, если требуется получить несколько подсумм? В этой главе мы покажем, как составить запрос для вычисления подсумм посредством объединения данных в группы.

Зачем нужно группировать данные

При работе с базой данных Sales Orders (Заказы на закупки) является полезным поиск количества заказов (COUNT), общих сумм продаж (SUM), среднего объема продаж (AVG), наименьшего (MIN) или наибольшего (MAX) заказа. А когда нужно вычислить любое из этих значений по клиенту, дате заказа или товару, можно добавить фильтр (WHERE) для извлечения строк для некоторого конкретного клиента или товара. Но что если нужно вычислить подсуммы для *всех* клиентов, отображая имя клиента вместе с подсуммами? Для этого необходимо затребовать от системы базы данных выполнения объединения строк в *группы*.

Подобным образом в базе данных Entertainment Agency (Агентства эстрадных мероприятий) легко найти количество контрактов, полную цену контракта, наименьшую или наибольшую цену контракта для всех контрактов. Можно даже отфильтровать строки таким образом, чтобы видеть эти вычисления для отдельного эстрадного артиста, конкретного клиента или для определенного диапазона дат. Но если нужно увидеть только строку общих сумм для каждого клиента или эстрадного артиста, следует сгруппировать строки.

EntStageName	ContractPrice
Albert Buchanan	$200.00
Albert Buchanan	$500.00
Albert Buchanan	$185.00
Albert Buchanan	$200.00
Albert Buchanan	$110.00
Albert Buchanan	$770.00
Albert Buchanan	$230.00
Albert Buchanan	$365.00
Albert Buchanan	$470.00
Carol Peacock Trio	$1,670.00
Carol Peacock Trio	$1,670.00
Carol Peacock Trio	$1,670.00
Carol Peacock Trio	$320.00
Carol Peacock Trio	$1,400.00
Carol Peacock Trio	$410.00
Carol Peacock Trio	$140.00
Carol Peacock Trio	$410.00
Carol Peacock Trio	$1,940.00
Carol Peacock Trio	$770.00
Carol Peacock Trio	$680.00
Caroline Coie Cuartet	$650.00
<< остальные строки >>	

Улавливаете идею? Когда от системы базы данных запрашивается группирование строк по значениям столбца или выражений, она формирует подмножества строк, основываясь на совпадающих значениях. Можно затем запросить базу данных подсчитать общие (агрегатные) значения *для каждой группы*. Рассмотрим это на простом примере для базы данных Entertainment Agency. Сначала нужно записать запрос, который извлекает интересующие нас столбцы — имена эстрадных артистов и цену контракта:

```
SQL  SELECT
        Entertainers.EntStageName,
        Engagements.ContractPrice
     FROM Entertainers
     INNER JOIN Engagements
     ON Entertainers.EnrertainerID =
        Engagements.EntertainerID
     ORDER BY EntStageName
```

Результат будет похож на приведенную слева таблицу. (В учебной БД этот запрос сохранен как Entertainers_And_ContractPrices.)

Можно подсчитать количество всех строк или найти наименьшее значение, наибольшее значение, сумму или среднее значение для столбца ContractPrice, если исключить из рассмотрения столбец с псевдонимами эстрадных артистов. Можно сохранить этот столбец,

если запросить базу данных сгруппировать по нему строки. При получении запроса на объединение в группы по псевдонимам база данных сформирует первую группу, содержащую первые девять строк ("Albert Buchanan"), вторую группу, содержащую следующие одиннадцать строк ("Carol Peacock Trio"), и т. д. по всей таблице. Теперь можно запросить COUNT (Подсчет) по строкам или SUM, MIN, MAX или AVG по столбцу с ценой контракта и получить совокупную строку по каждой эстрадной группе. Результат будет похож на представленный в следующей таблице.

EntStageName	NumContracts	TotPrice	MinPrice	MaxPrice	AvgPrice
Albert Buchanan	9	$3,030.00	$110.00	$770.00	$336.67
Carol Peacock Trio	11	$11,080.00	$140.00	$1,940.00	$1,007.27
Caroline Coie Cuartet	11	$15,070.00	$290.00	$2,450.00	$1,370.00
Coldwater Cattle Company	10	$19,100.00	$350.00	$3,800.00	$1,910.00
Country Feeling	16	$36,230.00	$275.00	$14,105.00	$2,264.38
Jazz Persuasion	8	$7,780.00	$500.00	$2,300.00	$972.50
Julia Schnebly	9	$4,665.00	$275.00	$875.00	$518.33
JV & the Deep Six	11	$18,820.00	$950.00	$3,650.00	$1,710.91
<< остальные строки >>					

Интересно выглядит, не так ли? Спорим, вам хотелось бы узнать, как это было сделано! Все детали подробно излагаются в следующих разделах.

Условие GROUP BY

Используя агрегатные функции, можно найти интересующую информацию любого типа. Во всех приведенных нами примерах агрегатные функции применялись ко *всем* строкам, возвращенным условиями FROM и WHERE. Используя условие WHERE, можно отфильтровать набор результатов по одной группе, но в действительности не был показан способ просмотра результатов из нескольких групп в одном запросе. Чтобы выполнить это суммирование по группам в одном запросе, требуется добавить еще одно основное условие к нашему словарю языка SQL — GROUP BY.

Синтаксис

Внимательно рассмотрим условие GROUP BY. На рис. 13.1 представлена основная диаграмма для оператора SELECT с добавленным условием GROUP BY.

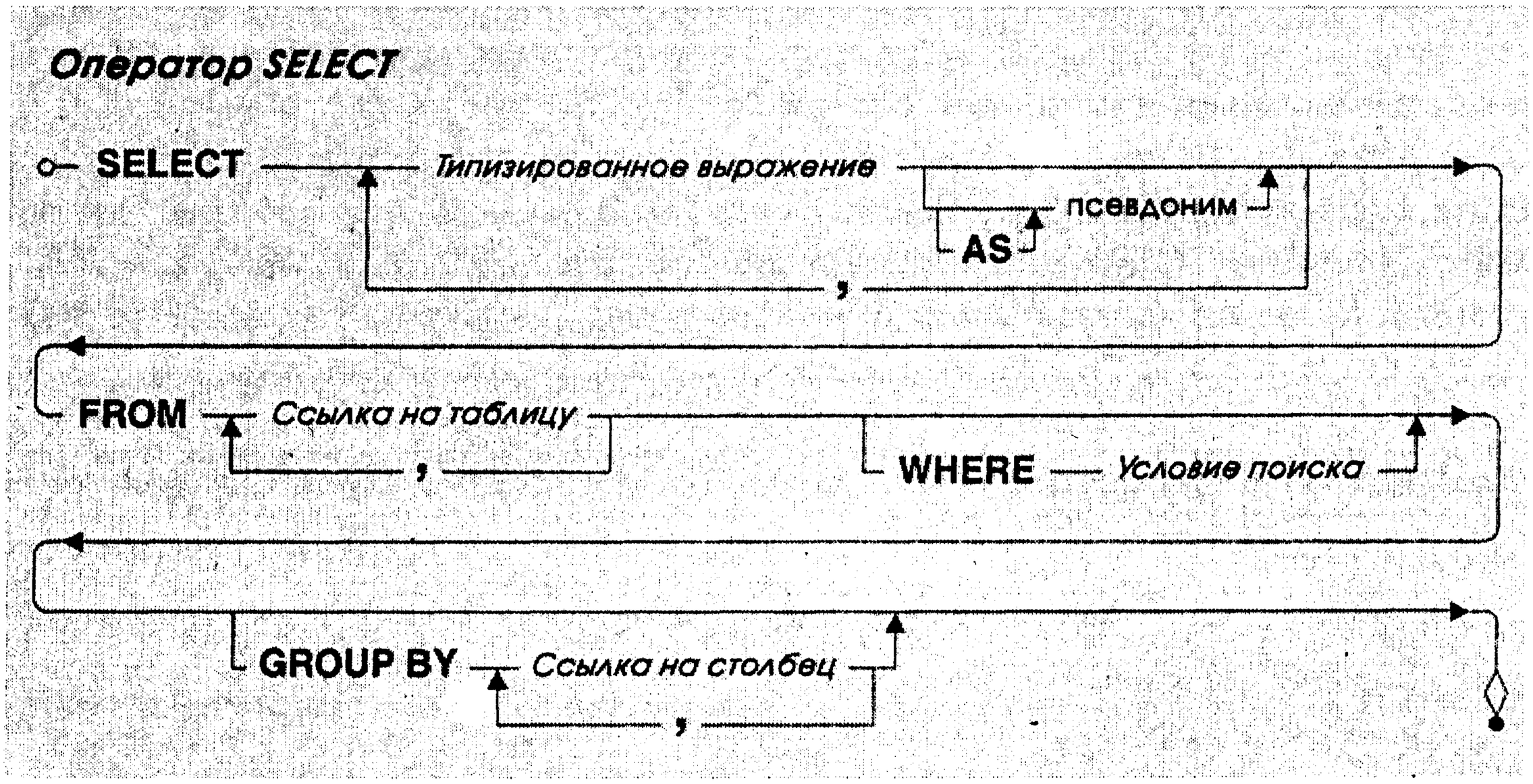


Рис. 13.1. *Оператор SELECT с условием GROUP BY*

Таблицы, которые являются источником данных, определяются в условии FROM. Оно может быть совсем простым, включая только имя одной таблицы, или достаточно сложным, состоящим из JOIN нескольких таблиц. Можно даже вложить весь табличный подзапрос (оператор SELECT) как ссылку на таблицу. Затем при необходимости можно указать условие WHERE, чтобы включить или исключить определенные строки, получаемые из условия FROM (см. главу 6).

Посредством добавления условия GROUP BY определяются столбцы в логической таблице, сформированной условиями FROM и WHERE, которые СУБД должна использовать как определение для групп срок. Строки, имеющие одинаковое значение в столбцах, указанных в списке, будут собраны вместе в некоторую группу; столбцы, которые указаны в списке условия GROUP BY, могут использоваться в типизированном выражении условия SELECT. Для выполнения вычисления для каждой из групп могут также использоваться любые агрегатные функции.

Попробуем применить условие GROUP BY, чтобы увидеть, какие вычисления можно выполнять с информацией о ценах контрактов для групп эстрадных артистов. На рис. 13.2 представлены таблицы, необходимые для решения этой задачи.

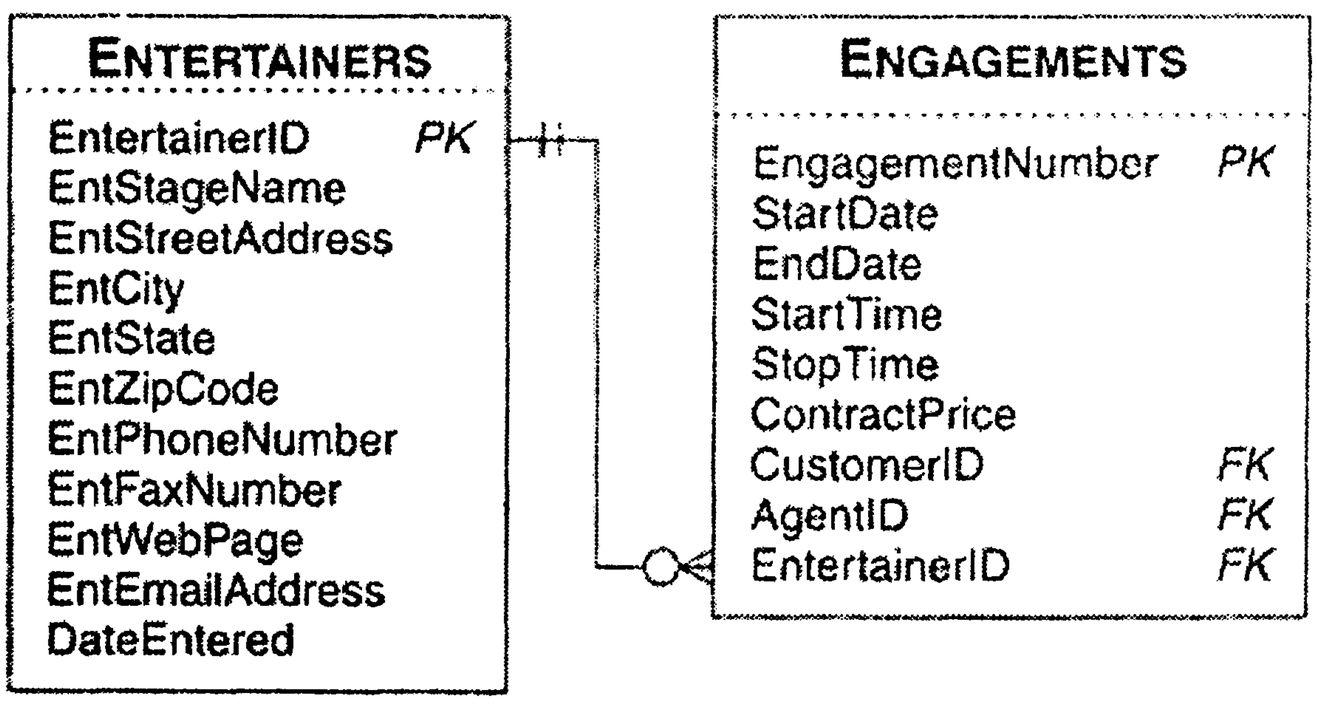


Рис. 13.2. *Связь между таблицами Entertainers и Engagements*

> **Внимание!** В этой главе используется метод "Запрос/Преобразование/Уточнение/SQL", введенный в главе 4.

"Show me for each entertainment group the group name, the count of contracts for the group, the total price of all the contracts, the lowest contract price, the highest contract price, and the average price of all the contracts".
("Показать, для каждой группы эстрадных артистов, название группы, количество контрактов группы, общую сумму стоимости всех контрактов, наименьшую стоимость контракта, наибольшую стоимость контракта и среднюю стоимость всех контрактов)

(*Совет*: Когда в запросе требуется количество, а также наименьшая, наибольшая или средняя величина значений на уровне деталей (контракты) для каждого значения на более высоком уровне (эстрадные артисты), необходимо использовать агрегатные функции и группирование. Помните, что у большинства эстрадных артистов не один контракт.)

Преобразование: Select entertainer name, the count of contracts, the sum of the contract price, the lowest contract price, the highest contract price, and the average contract price from the entertainers table joined with the engagements table on entertainer ID, grouped by entertainer name
(Выбрать имя эстрадного артиста, количество контрактов, сумму цен контракта, наименьшую стоимость контракта, наибольшую стоимость контракта и среднюю стоимость контракта из таблицы "Эстрадные артисты", соединенной с таблицей "Ангажементы" по идентификатору эстрадного артиста, сгруппированные по имени эстрадного артиста)

Уточнение: Select entertainer name, ~~the~~ count ~~of~~ (*) ~~contracts~~, ~~the~~ sum ~~of the~~ (contract price), ~~the lowest~~ min(contract price), ~~the highest~~ max(contract price), ~~and the average~~ avg(contract price) from ~~the~~ entertainers ~~table~~ join~~ed with the~~ engagements ~~table~~ on entertainer ID, group~~ed~~ by entertainer name
(Выбрать имя эстрадного артиста, count (*), sum(цена контракта), min(стоимость контракта), max(стоимость контракта) и avg(стоимость контракта) из "Эстрадные артисты", соединенной с "Ангажементы" по идентификатору эстрадного артиста, сгруппированные по имени эстрадного артиста)

SQL
```
SELECT Entertainers.EntStageName,
    COUNT (*) AS NumContracts,
    SUM (Engagements.ContractPrice) AS TotPrice,
    MIN (Engagements.ContractPrice) AS MinPrice,
```

```
    MAX (Engagements.ContractPrice) AS MaxPrice,
    AVG (Engagements.ContractPrice) AS AvgPrice
FROM Entertainers
INNER JOIN Engagements
ON  Entertainers.EntertainerID =
    Engagements.EntertainerID
GROUP BY Entertainers.EntStageName
```

Обратите внимание на сделанную подстановку MIN для наименьшего, MAX для наибольшего и AVG для среднего значения. Также в запрос включена функция COUNT(*), поскольку нужно подсчитать все строки с ангажементами (контрактами) невзирая на какие-либо значения Null. Добавление условия GROUP BY предоставляет агрегатные вычисления *по каждой группе эстрадных артистов в отдельности* и, кроме того, позволяет включить имя эстрадного артиста в условие SELECT. (Этот запрос сохранен в учебной базе данных как Aggregate_Contract_Info_By_Entertainer.)

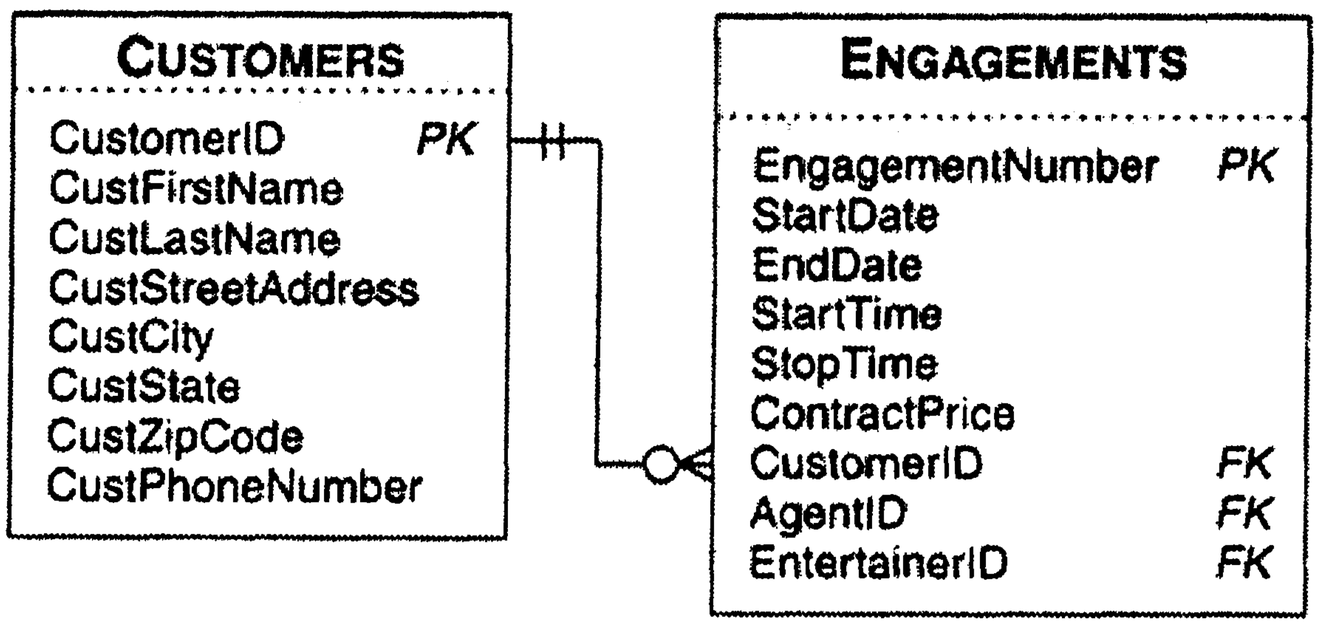


Рис. 13.3. *Связь между таблицами Customer и Engagements*

А что делать, если нужно выполнить объединение в группу более чем по одному значению? Рассмотрим эту проблему, но с точки зрения клиентов, а не артистов, и предположим, что нужно отобразить в наборе результатов как фамилию клиента, так и его имя. На рис. 13.3 представлены нужные таблицы.

"Show me for each customer the customer first and last name, the count of contracts for the customer, the total price of all the contracts, the lowest contract price, the highest contract price, and the average price of all the contracts".
("Показать для каждого клиента имя и фамилию, количество контрактов клиента, общую сумму всех контрактов, наименьшую стоимость контракта, наибольшую стоимость контракта и среднюю стоимость всех контрактов".)

Преобразование: Select customer last name, customer first name, the count of contracts, the sum of the contract price, the lowest contract price, the highest contract price, and the average contract price from the customers table joined with the engagements table on customer ID, grouped by customer last name and customer first name
(Выбрать фамилию клиента, имя клиента, количество контрактов, сумму цен контракта, наименьшую стоимость

контракта, наибольшую стоимость контракта и среднюю стоимость контракта из таблицы “Клиенты”, соединенной с таблицей “Ангажементы” по идентификатору клиента, сгруппированные по фамилии и по имени клиента)

Уточнение: Select customer last name, customer first name, ~~the~~ count ~~of~~ (*) ~~contracts~~, ~~the~~ sum ~~of the~~ (contract price), ~~the lowest~~ min(contract price), ~~the highest~~ max(contract price), ~~and the average~~ avg(contract price) from ~~the~~ customers ~~table~~ join~~ed with the~~ engagements ~~table~~ on customer ID, group~~ed~~ by customer last name ~~and~~ customer first name (Выбрать фамилию клиента, имя клиента, count (*), sum(цена контракта), min(стоимость контракта), max(стоимость контракта) и avg(стоимость контракта) из “Клиенты”, соединенной с “Ангажементы” по идентификатору клиента, сгруппированные по фамилии клиента, по имени клиента)

SQL
```
SELECT Customers.CustLastName,
    Customers.CustFirstName,
    COUNT(*) AS NumContracts,
    SUM (Engagements.ContractPrice) AS TotPrice,
    MIN (Engagements.ContractPrice) AS MinPrice,
    MAX (Engagements.ContractPrice) AS MaxPrice,
    AVG (Engagements.ContractPrice) AS AvgPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomersID = Engagements.CustomersID
GROUP BY Customers.CustLastName,
    Customers.CustFirstName
```

Результат будет похож на приведенную ниже таблицу. (Этот запрос сохранен в учебной базе данных как Aggregate_Contract_Info_By_Customer.)

CustLastName	CustFirstName	NumContracts	TotPrice	MinPrice	MaxPrice	AvgPrice
Bonnicksen	Joyce	8	$7,000.00	$320.00	$2,150.00	$875.00
Buchanan	Amelia	7	$4,685.00	$200.00	$1,490.00	$669.29
Callahan	Sally	9	$11,520.00	$140.00	$2,750.00	$1,280.00
Ehrlich	Mel	8	$7,015.00	$110.00	$2,930.00	$876.88
Ehrlich	Zachary	13	$12,455.00	$230.00	$1,550.00	$958.08
Fuller	Ann	10	$12,320.00	$350.00	$2,450.00	$1,232.00
Fuller	Thomas	10	$13,745.00	$200.00	$2,675.00	$1,374.50
Hallmark	Elizabeth	8	$25,585.00	$410.00	$14,105.00	$3,198.13
<< остальные строки >>						

Поскольку для отображения полного имени клиента требуется два столбца, *оба* они должны быть включены в условие GROUP BY. Запомните, что если некоторый столбец, который не является результатом агрегатного вычисления, нужно включить в вывод, то он обязательно должен также быть включен в условие GROUP BY. Столбец ContractPrice не был включен в условие GROUP BY, потому что этот столбец используется во многих выражениях в агрегатных функциях. Если бы ContractPrice был включен, то были бы получены отдельные группы клиентов и цен. MIN, MAX и AVG — все возвратят эту сгруппированную цену. COUNT будет больше единицы, только если у данного клиента имеется более одного контракта с той же самой ценой. Тем не менее группирование по клиенту и цене и запрос функции COUNT будет хорошим способом поиска клиентов, у которых есть несколько контрактов с одинаковой ценой.

Смешивание столбцов и выражений

Предположим, нам нужен список имен клиентов как один столбец вывода, полный адрес клиента как второй столбец вывода, дата последнего ангажемента и сумма цен контрактов ангажемента. Полное имя клиента находится в двух столбцах: CustFirstName и CustLastName. Для задания полного адреса требуются столбцы CustStreetAddress, CustState, CustCity и CustZipCode. Рассмотрим, как построить SQL для этого запроса (он сохранен в учебной базе данных под именем Customers_Last_Booking).

"Show me for each customer the customer full name, the customer full address, the latest contract date for the customer, and the total price of all the contracts".
("Показать для каждого клиента полное имя клиента, полный адрес клиента, дату последнего контракта клиента и общую сумму всех контрактов".)

Преобразование: Select customer last name and customer first name as customer full name; street address, city, state, and zipcode as customer full address; the latest contract start date; and the sum of the contract price from the customers table joined with the engagements table on customer ID, grouped by customer last name, customer first name, customer street address, customer city, customer state, and customer zip code
(Выбрать фамилию и имя клиента как полное имя клиента; улицу, город, штат и почтовый индекс как полный адрес клиента; дату начала последнего контракта; сумму цены контракта из таблицы "Клиенты", соединенной с таблицей "Ангажементы" по идентификатору клиента, сгруппированные по фамилии клиента, имени клиента, улице клиента, городу клиента, штату клиента и почтовому коду клиента)

Уточнение: Select customer last name ~~and~~ || ',' || customer first name as customer full name; street address, || ',' || city, || ',' || state, ~~and~~ || ',' || zipcode as customer full address; ~~the latest~~ MAX(~~contract~~ start date) as latest date;, ~~and the~~ sum ~~of~~ ~~the~~ (contract price) from ~~the~~ customers ~~table~~ join~~ed with~~ ~~the~~ engagements ~~table~~ on customer ID, group~~ed~~ by customer last name, customer first name, customer street address, customer city, customer state, ~~and~~ customer zip code (Выбрать фамилию клиента ||','||, имя клиента как полное имя клиента, улицу ||','||, город ||','||, штат ||','||, почтовый индекс как полный адрес клиента, MAX(дата начала) как последнюю дату, сумму (цена контракта) как общую цену контракта из "Клиенты", соединенной с "Ангажементы" по идентификатору клиента, сгруппированные по фамилии клиента, имени клиента, улице клиента, городу клиента, штату клиента, почтовому коду клиента)

SQL

```
SELECT Customers.CustLastName || ', ' ||
    Customers.CustFirstName AS CustomerFullName
    Customers.CustStreetAddress || ', ' ||
    Customers.CustSity || ', ' ||
    Customers.CustState || ' ' ||
    Customers.CustZipCode AS CustomerFullAddress
    MAX(Engagements.StartDate) AS LatestDate
    SUM(Engagements.ContractPrice),
    AS TotalContractPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID = Engagements.CustomerID
GROUP BY Customers.CustLastName,
    Customers.CustFirstName,
    Customers.CustStreetAddress,
    Customers.CustSity, Customers.CustState
    Customers.CustZipCode
```

Здесь требуется перечислить все и каждый из столбцов, используемых в выходном выражении, которое не включает агрегатную функцию. Столбцы StartDate и ContractPrice использовались в составных выражениях, поэтому не требуется перечислять их в условии GROUP BY. Фактически не имеет смысла группировать их как по StartDate, так и по ContractPrice, потому что мы хотим использовать их в составных вычислениях по нескольким клиентам. Если, например, выполнить объединение в группу по StartDate, то MAX(StartDate) возвратит значение группирования, а SUM(ContractPrice) возвратит только сумму цен контрактов для клиента по любой указанной дате. Вы получите сумму лишь одного контракта, если только у клиента не имеется более одного контракта на указанную дату — что маловероятно.

Использование GROUP BY в подзапросе условия WHERE

Рассмотрим запрос, который требует использования как подзапроса с агрегатной функцией, так и условия GROUP BY в этом подзапросе:

> *"Display the engagement contract whose price is greater than the sum of all contracts for any other customer".*
> *("Вывести на экран контракт ангажемента, цена которого больше суммы всех контрактов для любого другого клиента".)*

Преобразование: Select customer first name, customer last name, engagement start date, and engagement contract price from the customers table joined with the engagements table on customer ID where the contract price is greater than the sum of all contract prices for customers other than the current customer, grouped by customer ID
(Выбрать имя клиента, фамилию клиента, дату начала ангажемента и стоимость контракта ангажемента из таблицы "Клиенты", соединенной с таблицей "Ангажементы" по идентификатору клиента, где цена контракта больше суммы всех цен контрактов для других клиентов, иных, чем текущий клиент, сгруппированные по идентификатору клиента)

Уточнение: Select customer first name, customer last name, engagement start date, ~~and~~ engagement contract price from ~~the~~ customers ~~table~~ join~~ed with the~~ engagements ~~table~~ on customer ID where ~~the~~ contract price ~~is greater than~~ > ALL (SELECT ~~the~~ sum ~~of all~~ contract prices ~~for~~ customers ID <> ~~other than the current~~ customer ID, group~~ed~~ by customer ID)
(Выбрать имя клиента, фамилию клиента, дату начала ангажемента, стоимость контракта ангажемента из "Клиенты", соединенной с "Ангажементы" по идентификатору клиента, где цена контракта > всех (Выбрать сумму цен контрактов по идентификатору клиентов <> идентификатор иного клиента, сгруппированные по идентификатору клиента))

SQL
```
SELECT Customers.CustFirstName,
    Customers.CustLastName, Engagements.StartDate,
    Engagements.ContractPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID = Engagements.CustomerID
```

```
WHERE Engagements.ContractPrice > ALL
    (Select SUM(ContractPrice)
    FROM Engagements AS E2
    WHERE E2.CustomerID <> Customers.CustomerID
    GROUP BY E2.CustomerID)
```

Проанализируем то, что выполняется в подзапросе. Для каждого ангажемента, который рассматривается запросом в JOIN клиентов и ангажементов, подзапрос вычисляет SUM всех цен контрактов для всех *других* клиентов и объединяет их в группы по идентификатору клиента. Поскольку в базе данных имеется несколько клиентов, подзапрос возвращает несколько значений SUM — по одному для каждого из других клиентов. По этой причине нельзя использовать условие сравнения просто на больше (>). Однако можно воспользоваться количественным условием сравнения "больше, чем все" (> ALL) для проверки множества значений. Если выполнить этот запрос для примера базы данных Entertainment Agency (сохраненного как "Biggest_Big_Contract"), то обнаружится, что только один контракт удовлетворяет всем требованиям.

CustFirstName	CustLastName	StartDate	ContractPrice
Elizabeth	Hallmark	1999-11-21	$14,105.00

Моделирование оператора SELECT DISTINCT

Можно ли использовать условие GROUP BY, не включая каких-либо агрегатных функций в условие SELECT? Конечно можно! При этом будет получен тот же эффект, что и при использовании ключевого слова DISTINCT (см. главу 4).

Рассмотрим простой запрос, требующий уникальных значений, и разрешим его, используя оба метода:

"Show me the unique city names from the customers table".
("Показать уникальные названия городов из таблицы "Клиенты".)

Преобразование 1: Select the distinct city names from the customers table
(Выбрать неповторяющиеся названия городов
из таблицы "Клиенты")

Уточнение: Select ~~the~~ distinct city names from ~~the~~ customers ~~table~~
(Выбрать неповторяющиеся названия городов
из "Клиенты")

SQL

```
SELECT DISTINCT Customers.CustCityName
FROM Customers
```

Преобразование 2/ Select city name from ~~the~~ customers ~~table~~, group~~ed~~ by
Уточнение: city name
(Выбрать названия городов из таблицы “Клиенты”, сгруппированные по названию города)

```
SQL     SELECT Customers.CustCityName
        FROM Customers
        GROUP BY Customers.CustCityName
```

GROUP BY объединяет в группу все строки по определенному вами столбцу группирования и возвращает одну строку для каждой группы. Это немного другой способ получения того же результата, который был получен с ключевым словом DISTINCT. Какой из них лучше? Нам кажется, что DISTINCT более понятное утверждение того, что нужно, но ваша СУБД может решить задачу быстрее при использовании GROUP BY.

Наложение некоторых ограничений

Добавление условия GROUP BY налагает определенные ограничения на построение запроса. Рассмотрим эти ограничения, чтобы оградить вас от попадания в обычные ловушки.

Ограничения на использование столбцов

Когда добавляется условие GROUP BY, это означает, что от системы базы данных запрашивается формирование уникальных групп строк из строк, возвращенных таблицами, определенными в условии FROM и отфильтрованными условием WHERE. В условии SELECT можно использовать столько составных выражений, сколько требуется, и эти выражения могут использовать любые столбцы таблицы, определенной условиями FROM и WHERE. Ссылка на столбец в составном выражении, а также включение этого столбца в спецификацию группирования, вероятно, не имеет смысла.

Если выбрать включение выражений со ссылкой на столбцы, но без агрегатной функции, то потребуется перечислить в списке *все* столбцы, которые используются таким образом, в условии GROUP BY. Одна из наиболее обычных ошибок состоит в предположении, что, если столбцы исходят из уникальных строк, то в выражениях, которые не являются составными, могут быть ссылки на эти столбцы. Рассмотрим, например, некорректный запрос, который включает значение первичного ключа, т. е. то, что по определению является уникальным.

> *“Display the customer ID, customer full name, and the total of all engagement contract prices”.*
> *(“Вывести на экран идентификатор клиента, полное имя клиента и общую сумму всех цен контрактов ангажемента”.)*

Преобразование: Select customer ID, customer first name, and customer last name as customer full name, and the sum of contract prices from the customer table joined with the engagements table on customer ID, grouped by customer ID
(Выбрать идентификатор клиента, имя клиента и фамилию клиента как полное имя клиента, и сумму цен контрактов из таблицы "Клиенты", соединенной с таблицей "Ангажементы" по идентификатору клиента, сгруппированные по идентификатору клиента)

Уточнение: Select customer ID, customer first name, ~~and~~ ||''|| customer last name as customer full name, ~~and the~~ sum ~~of~~ (contract price)~~s~~ from ~~the~~ customer ~~table~~ join~~ed with the~~ engagements ~~table~~ on customer ID, group~~ed~~ by customer ID
(Выбрать идентификатор клиента, имя клиента ||''|| фамилию клиента как полное имя клиента, sum (цена контракта) из "Клиенты", соединенной с "Ангажементы" по идентификатору клиента, сгруппированные по идентификатору клиента)

SQL
```
SELECT Customers.CustomerID,
    Customers.CustFirstName || ' ' ||
    Customers.CustLastName AS CustFullName,
    SUM(Engagements.ContractPrice) AS TotalPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID = Engagements.CustomerID
GROUP BY Customers.CustomerID
```

Нам *известно*, что CustomerID (идентификатор клиента) является уникальным для клиента. Группирования по CustomerID должно быть достаточно для извлечения уникальной информации об имени и фамилии клиента в пределах групп, сформированных по CustomerID. Однако SQL — это язык, основанный на синтаксисе, а не на семантике. Иначе говоря, SQL не принимает во внимание любую информацию, которая может подразумеваться структурой таблиц БД, включая и то, какие столбцы являются первичным ключом. SQL требует синтаксической "чистоты" запроса и возможности его преобразования без учета какой-либо информации о структуре таблиц, лежащих в его основе. Таким образом, показанный выше оператор SQL приведет к ошибке в системе базы данных, что полностью соответствует стандарту SQL, потому что в условие SELECT включены столбцы, которые отсутствуют в агрегатной функции, а также и в условии GROUP BY (CustFirstName и CustLastName). Правильный SQL-запрос выглядит следующим образом:

SQL
```
SELECT Customers. CustomerID,
    Customers.CustFirstName || ' ' ||
```

```
    Customers.CustLastName AS CustFullName,
    SUM(Engagements.ContractPrice) AS TotalPrice
FROM Customers
INNER JOIN Engagements
ON Customers.CustomerID = Engagements.CustomerID
GROUP BY Customers.CustomerID,
    Customers.CustFirstName, Customers.CustLastName
```

Может показаться избыточным, но это — правильный способ решения!

Внимание! В некоторых системах баз данных требуется точно повторить *выражения*, используемые в условии SELECT, в условии GROUP BY. В качестве примеров можно привести Oracle и Microsoft Access. В приведенном выше запросе можно было бы выполнить:

```
GROUP BY Customers.CustomerID
    Customers.CustFirstName ]| ' ' ||
    Customers.CustLastName
```

вместо перечисления отдельных столбцов. Это не соответствует стандарту SQL, но может оказаться, что это будет единственным способом заставить вашу систему выполнить данный запрос.

Группирование выражений

Мы уже приводили несколько правильных примеров создания выражений, которые не включали агрегатные функции. Одной из наиболее обычных ошибок является попытка объединения в группу по выражению, созданному в условии SELECT, вместо объединения по отдельным столбцам. Вспомните, что условие GROUP BY должно ссылаться на столбцы, созданные условиями FROM и WHERE. В нем нельзя использовать выражение, созданное в условии SELECT.

Рассмотрим еще ранее решенный пример, чтобы показать вам, что имеется в виду, но в этот раз намеренно сделаем ошибку (а также пропустим этапы "Преобразование" и "Уточнение", поскольку они уже рассматривались):

> *"Show me for each customer in the state of Washington the customer full name, the customer full address, the latest contract date for the customer, and the total price of all the contracts".*
> *("Показать для каждого клиента из штата Вашингтон полное имя клиента, полный адрес клиента, дату самого последнего контракта клиента и общую стоимость всех контрактов".)*

```
SQL             SELECT Customers.CustLastName || ', ' ||
                    Customers.CustFirstName AS CustomerFullName,
```

```
    Customers.CustStreetAddress || ', ' ||
    Customers.CustCity || ', ' ||
    Customers.CustState || ' ' ||
    Customers.CustZip AS CustomerFullAddress
    MAX(Engagements.StartDate) AS LatestDate,
    SUM (Engagements.ContractPrice)
    AS TotalContractPrice
FROM Customers
INNER JOIN Engagements
    ON Customers.CustomerID = Engagements.CustomerID
WHERE Customers.CustState = 'WA'
GROUP BY CustomerFullame, CustomerFullAddress
```

Некоторые СУБД позволяют обойти это, но это неправильно. Столбцы CustomerFullName и CustomerFullAddress не существуют *до тех пор*, пока СУБД не вычислит условия FROM, WHERE и GROUP BY. Условие GROUP BY не обнаружит эти столбцы в результате, созданном в условиях FROM и WHERE, поэтому в системе базы данных, которая строго придерживается стандарта SQL, будет получена синтаксическая ошибка.

Для корректного способа решения необходимо перечислить в списке все столбцы, используемые в выражениях CustomerFullName и CustomerFullAddress. Другой способ — заставить условие FROM сформировать вычисленные столбцы, используя вложенный табличный подзапрос. Вот как он может выглядеть:

SQL

```
SELECT CE.CustomerFullName, CE.CustomerFullAddress,
    MAX(CE.StartDate) AS LatestDate,
    SUM(CE.ContractPrice)
    AS TotalContractPrice
FROM
    (SELECT Customers.CustLastName || ', ' ||
     Customers.CustFirstName AS CustomerFullName,
     Customers.CustStreetAddress || ', ' ||
     Customers.CustCity || ', ' ||
     Customers.CustState || ' ' ||
     Customers.CustZip AS CustomerFullAddress,
     Engagements.StartDate, Engagements.ContractPrice
    FROM Customers
    INNER JOIN Engagements
    ON Customers.CustomerID = Engagements.CustomerID
    WHERE Customers.CustState = 'WA')
AS CE
GROUP BY CE.CustomerFullName,
    CE.CustomerFullAddress
```

В данном случае это работает, поскольку столбцы CustomerFullName и CustomerFullAddress сформированы как вывод в условии FROM. Следует признать, тем не менее, что это очень усложняет запрос. По правде говоря, лучше просто перечислить в списке все отдельные столбцы, которые планируется использовать в не являющихся составными выражениях, чем пытаться сгенерировать выражения как столбцы в условии FROM.

Использование GROUP BY

К настоящему моменту вы уже должны хорошо понимать того, как запрашивать подсуммы для групп, используя агрегатные функции и условие GROUP BY. Самый лучший способ продемонстрировать широкий диапазон использования GROUP BY — это показать задачи, которые можно решать, используя это новое условие.

"Показать каждого поставщика и среднее для поставщика количество дней на доставку товара".

"Вывести на экран для каждого товара наименование товара и общий объем продаж".

"Привести для каждого клиента и даты заказа полное имя клиента и общую стоимость единиц товара, заказанных на каждую дату".

"Вывести на экран, для каждого идентификатора эстрадной группы, участника эстрадной группы и сумму выплаты для каждого участника, основанной на общей сумме контракта, разделенной на число участников в группе".

"Показать имя каждого агента, сумму цен контрактов для заявленных ангажементов и общую сумму комиссионного вознаграждения агента".

"Для закончившихся курсов лекций привести список по категории и по студенту для наименования категории, имени студента и среднего балла студента за все курсы лекций, взятые из этой категории".

"Привести список категорий с наименованием категории и количеством предложенных курсов лекций".

"Привести список всех штатных сотрудников и количество курсов лекций, которые каждый из них запланировал прочитать".

"Показать, для всех турниров и матчей, идентификатор турнира, место проведения турнира, номер турнира, название каждой команды и общее количество очков гандикапа для каждой команды".

"Вывести на экран дисплея, для каждого игрока в боулинг, имя игрока и среднее количество очков за предварительные игры".

"Показать, сколько существует рецептов для каждого вида компонентов".

"Какое количество каждого из компонентов потребуется иметь в запасе, если нужно будет приготовить все рецепты из поваренной книги?"

Примеры операторов

Теперь ознакомимся с множеством примеров, все из которых требуют группирования информации. Они взяты из учебных баз данных.

Сюда также включены примеры наборов результатов, которые должны возвращать эти операции. Мы поместили их сразу же после графического описания синтаксиса SQL в виде линии. Имя, которое появляется непосредственно над набором результатов, присвоено каждому запросу в примере базы данных. Каждый запрос сохранен в соответствующем примере базы данных во вложенной папке "Chapter 13" на сайте издательства "Лори". Чтобы загрузить примеры на свой компьютер и проверить их, следуйте указаниям, приведенным в начале этой книги.

Внимание! Все имена столбцов и таблиц, используемые в этих примерах, взяты из учебных структур баз данных, представленных в приложении B. Для упрощения процесса этапы преобразования и уточнения объединены.

Эти примеры предполагают, что тщательно изучены и поняты концепции, рассмотренные в предыдущих главах, особенно в главах по JOIN и подзапросам.

База данных заказов на закупку

"List for each customer and order date the customer full name and the total cost of items ordered on each date".
("Привести список для каждого клиента и даты заказа, указав полное имя клиента и общую стоимость единиц товара, заказанных на каждую дату".)

Преобразование/ Уточнение:

Select customer first name ~~and~~ || '' || customer last name as customer full name, order date, ~~and the~~ sum ~~of~~ (quoted price ~~time~~ * quantity ordered) as total cost from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~and then~~ join~~ed with the~~ order details ~~table~~ on order number, group~~ed~~ by customer first name, customer last name, ~~and~~ order date

(Выбрать имя клиента || ‘’ ||, фамилию клиента как полное имя клиента, дату заказа, sum(объявленная цена * заказанное количество) как общую стоимость из “Клиенты”, соединенной с “Заказы” по идентификатору клиента, соединенной с “Детали заказа” по номеру заказа, сгруппированные по имени клиента, фамилии клиента, дате заказа)

SQL

```
SELECT Customers.CustFirstName || ‘ ‘ ||
    Customers.CustLastName AS CustFullName,
    Orders.OrderDate,
    SUM(Order_Details.QuotedPrice *
    Order_Details.QuantityOrdered) AS TotalCost
FROM (Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber
GROUP BY Customers.CustFirstName,
    Customers.CustLastName, Orders.OrderDate
```

Order_Totals_By_Customer_And_Date (847 строк)

CustFullName	OrderDate	TotalCost
Alaina Hallmark	1999-07-02	$4,699.98
Alaina Hallmark	1999-07-14	$4,433.95
Alaina Hallmark	1999-07-18	$353.25
Alaina Hallmark	1999-07-21	$3,951.90
Alaina Hallmark	1999-07-22	$10,388.68
Alaina Hallmark	1999-07-30	$3,088.00
Alaina Hallmark	1999-08-11	$6,775.06
Alaina Hallmark	1999-08-21	$15,781.10
<< остальные строки >>		

База данных агентства эстрадных мероприятий

“Display each entertainment group ID, entertainment group member, and the amount of pay for each member based on the total contract price divided by the number of member in the group”.

("Вывести на экран дисплея, для каждого идентификатора эстрадной группы, участника эстрадной группы и сумму выплаты для каждого участника, основанную на общей сумме контракта, разделенной на число участников в группе".)

Внимание! Это действительно сложная задача, потому что каждый участник может принадлежать более чем к одной группе эстрадных артистов. Требуется вычислить сумму цен контрактов для каждого эстрадного артиста, а затем разделить на число участников в этой группе (предполагая, что каждый участник получает одинаковую оплату). Извлечение итоговой суммы требует фильтрации подзапроса по текущему идентификатору эстрадного артиста (по идентификатору группы, а не участника), что означает, что требуется объединение в группы по идентификатору эстрадного артиста. И не забудьте исключить тех участников, которые не активны (Состояние = 3).

Преобразование/ Уточнение:

Select entertainer ID, member first name, member last name, ~~and the~~ sum ~~of~~ (contract price)~~s divided by~~ / ~~the~~ (SELECT count(*) ~~of active members~~ FROM entertainer_members AS EM2 ~~in the current entertainer group~~ WHERE ~~the~~ EM2 entertainer ID = ~~the~~ entertainer members entertainer ID) from ~~the~~ members ~~table~~ join~~ed with the~~ entertainer members ~~table~~ on member ID, ~~then~~ join~~ed with the~~ entertainers ~~table~~ on entertainer ID, ~~and finally~~ join~~ed with the~~ engagements ~~table~~ on entertainer ID, where member status ~~is not equal to~~ <> 3, group~~ed~~ by entertainer ID, member first name, ~~and~~ member last name, ~~sorted~~ ORDER by member last name
(Выбрать идентификатор эстрадного артиста,
имя участника, фамилию участника,
sum(цена контракта)/(выбрать count(*)
FROM entertainer_members AS EM2
WHERE EM2 — идентификатор эстрадного артиста =
идентификатор эстрадной группы) из "Участники",
соединенной с "Участники эстрадной группы"
по идентификатору участника, соединенной
с "Эстрадные артисты" по идентификатору
эстрадного артиста, соединенной с "Ангажементы"
по идентификатору эстрадного артиста, где статус
участника <> 3, сгруппированные по идентификатору
эстрадного артиста, имени участника, фамилии участника,
упорядоченные по фамилии участника)

SQL

```
SELECT Entertainers.EntertainerID,
    Members.MbrFirstName, Members.MbrLastName,
    SUM (Engagements.ContractPrice)/
        (SELECT COUNT(*)
         FROM Entertainer_Members AS EM2
         WHERE EM2.Status <> 3
         AND EM2.EntertainerID =
         Entertainers.EntertainerID)
    AS MemberPay
FROM ((Members
INNER JOIN Entertainer_Members
ON Members.MemberID =
    Entertainer_Members.MemberID)
INNER JOIN Entertainers
ON Entertainers.EntertainerID =
    Entertainer_Members.EntertainerID)
INNER JOIN Engagements
ON Entertainers.EntertainerID =
    Engagements.EntertainerID
WHERE Entertainer_Members.Status <> 3
GROUP BY Entertainers.EntertainerID,
    Members.MbrFirstName, Members.MbrLastName
ORDER BY Members.MbrLastName
```

Member_Pay (40 строк)

EntertainerID	MbrFirstName	MbrLastName	MemberPay
1010	Kendra	Bonnicksen	$3,675.00
1013	Kendra	Bonnicksen	$3,767.50
1004	Albert	Buchanan	$3,030.00
1007	Andrea	Buchanan	$3,820.00
1001	Laura	Callahan	$3,693.33
1008	George	Chavez	$7,246.00
1013	George	Chavez	$3,767.50
1010	Caroline	Coie	$3,675.00
<<остальные строки >>			

База данных лиги игры в боулинг

"Show me for each tournament and match the tournament ID, the tournament location, the match number, the name of each team, and the total of the handicap score for each team".
("Показать, для всех турниров и матчей, идентификатор турнира, место проведения турнира, номер турнира, название каждой команды и общее количество очков гандикапа для каждой команды".)

Преобразование/ Уточнение:

Select tourney ID, tourney location, match ID, team name ~~and the~~ sum ~~of~~ (handicap score) as TotHandicapScore from ~~the~~ tournaments ~~table~~ join~~ed with the~~ tourney matches ~~table~~ on tournament ID, ~~then~~ join~~ed with the~~ match games ~~table~~ on match ID, ~~then~~ join~~ed with the~~ bowler scores ~~table~~ on match ID and game number, ~~then~~ join~~ed with the~~ bowlers ~~table~~ on bowler ID, ~~and finally~~ join~~ed with the~~ teams ~~table~~ on team ID, group~~ed~~ by tourney ID, tourney location, match ID, ~~and~~ team name
(Выбрать идентификатор турнира, место проведения турнира, идентификатор матча, название команды, sum(очки гандикапа) как TotHandicapScore из "Турниры", соединенной с "Игры матча" по идентификатору матча, соединенной с "Игры" по идентификатору матча, соединенной с "Очки игрока в боулинг" по идентификатору матча и номеру игры, соединенной с "Игроки в боулинг" по идентификатору игрока в боулинг, соединенной с "Команды" по идентификатору команды, сгруппированные по идентификатору турнира, месту проведения турнира, идентификатору матча, названию команды)

SQL

```
SELECT Tournaments.TourneyID,
    Tournaments.TourneyLocation,
    Tourney_Matches.MatchID, Teams.TeamName,
    Sum(Bowler_Scores.HandiCapScore)
    AS TotHandiCapScore
FROM ((((Tournaments
INNER JOIN Tourney_Matches
ON Tournaments.TourneyID =
    Tourney_Matches.Tourney ID)
INNER JOIN Match_Games
ON Tourney_Matches.MatchID =
    Match_Games.MatchID)
INNER JOIN Bowler_Scores
```

```
ON (Match_Games.MatchID =
    Bowler_Scores.MatchID) AND
    (Match_Games.GameNumber =
    Bowler_Scores.GameNumber))
INNER JOIN Bowlers
ON Bowlers.BowlerID = Bowler_Scores.BowlerID)
INNER JOIN Teams
ON Teams.TeamID = Bowlers.TeamID
GROUP BY Tournaments.TourneyID,
    Tournaments.TourneyLocation,
    Tourney_Matches.MatchID, Teams.TeamName
```

Как можно видеть, трудная часть этого запроса состоит в сборе сложных условий JOIN для связи всех таблиц корректным способом.

Tournament_Match_Team_Results (112 строк)

TourneyID	TourneyLocation	MatchID	TeamName	TotHandiCapScore
1	Red Rooster Lanes	1	Marlins	2351
1	Red Rooster Lanes	1	Sharks	2348
1	Red Rooster Lanes	2	Barracudas	2289
1	Red Rooster Lanes	2	Terrapins	2391
1	Red Rooster Lanes	3	Dolphins	2389
1	Red Rooster Lanes	3	Orcas	2395
1	Red Rooster Lanes	4	Manatees	2292
1	Red Rooster Lanes	4	Swordfish	2353
2	Thunderbird Lanes	5	Marlins	2297
2	Thunderbird Lanes	5	Terrapins	2279
<< остальные строки >>				

База данных расписания занятий

"For completed classes, list by category and student the category name, the student name, and the student's average grade of all classes taken in that category".
("Для закончившихся курсов лекций привести список по категории и по студенту для наименования категории, имени студента и среднего балла студента за все курсы лекций, взятые из этой категории".)

Преобразование/ Уточнение:

Select category description, student first name, student last name, and the average AVG(of grade) as AvgOfGrade from the categories table joined with the subjects table on category ID then joined with the classes table on subject ID, then joined with the student schedules table on class ID, then joined with the student class status table on class status, and finally joined with the students table on student ID where class status description is = 'Completed', grouped by category description, student first name, and student last name
(Выбрать описание категории, имя студента, фамилию студента, AVG(балл) как AvgOfGrade из "Категории", соединенной с "Предметы" по идентификатору категории, соединенной с "Курсы лекций" по идентификатору предмета, соединенной с "Расписание занятий студента" по идентификатору курса лекций, соединенной с "Состояние курсов лекций студента" по состоянию курса лекций, соединенной со "Студенты" по идентификатору студента, где описание состояния курса лекций = 'Завершен', сгруппированные по описанию категории, имени студента, фамилии студента)

SQL

```
SELECT Categories.CategoryDescription,
    Students.StudFirstName,
    Students.StudLastName,
    Avg(Student_Schedules.Grade) AS AvgOfGrade
FROM ((((Categories
INNER JOIN Subjects
ON Categories.CategoryID = Subjects.CategoryID)
INNER JOIN Classes
ON Subjects.SubjectID = Classes.SubjectID)
INNER JOIN Student_Schedules
ON Classes.ClassID = Student_Schedules.ClassID)
INNER JOIN Student_Class_Status
ON Student_Class_Status.ClassStatus =
    Student_Schedules.ClassStatus)
INNER JOIN Students
ON Students.StudentID =
    Student_Schedules.StudentID
WHERE Student_Class_Status.ClassStatusDescription =
    'Completed'
GROUP BY Categories.CategoryDescription,
    Students.StudFirstName,
    Students.StudLastName
```

Student_GradeAverage_By_Category (45 строк)

CategoryDescription	StudFirstName	StudLastName	AvgOfGrade
Accounting	Andrew	Fuller	79.43
Accounting	Elizabeth	Hallmark	90.24
Accounting	John	Kennedy	71.45
Accounting	Michael	Viescas	90.01
Accounting	Sara	Kennedy	89.92
Accounting	Sarah	Leverling	90.67
Accounting	Steven	Buchanan	87.82
Accounting	Steven	Pundt	84.37
Art	Carol	Peacock	80.78
<< остальные строки >>			

База данных рецептов

"Show me how many recipes exist for each class of ingredient". ("Показать, сколько существует рецептов для каждого вида компонентов".)

Внимание! Трудность данной задачи состоит в том, что мы не хотим выполнять подсчет по конкретному виду рецептов более одного раза на рецепт. Например, если рецепт содержит несколько растительных и молочных компонентов, то он должен подсчитываться только один раз для конкретного вида. Похоже, настало самое время воспользоваться функцией COUNT(DISTINCT *Value Expression*), не так ли?

Преобразование/ Уточнение:

Select ingredient class description, ~~and the unique~~ count ~~of~~ (DISTINCT recipe ID) from ~~the~~ ingredient classes ~~table~~ join~~ed with the~~ ingredients ~~table~~ on ingredient class ID, ~~and then~~ join~~ed with the~~ recipe ingredients ~~table~~ on ingredient ID, group~~ed~~ by ingredient class description
(Выбрать описание вида компонента, подсчитать (DISTINCT идентификатор рецепта) из "Виды компонентов", соединенной с "Компоненты" по идентификатору вида компонента, соединенной с "Компоненты рецепта" по идентификатору компонента, сгруппированные по описанию класса компонента)

SQL

```
SELECT
    Ingredient_Classes.IngredientClassDescription,
    Count(DISTINCT RecipeID) AS CountOfRecipeID
FROM (Ingredient_Classes
INNER JOIN Ingredients
ON  Ingredient_Classes.IngredientClassID =
    Ingredients.IngredientClassID)
INNER JOIN Recipe_Ingredients
ON  Ingredients.IngredientID =
    Recipe_Ingredients.IngredientID
GROUP BY
    Ingredient_Classes.IngredientClassDescription
```

IngredientClass_Distinct_Recipe_Count (18 строк)

IngredientClassDescription	CountOfRecipeID
Butter	3
Cheese	2
Chips	1
Condiment	3
Dairy	2
Fruit	1
Grain	2
Herb	1
<< остальные строки >>	

Внимание! Поскольку Microsoft Access 2000 не поддерживает COUNT DISTINCT, запрос для примера базы данных на Access вначале выбирает DISTINCT значения для Recipe ID (идентификатора рецепта), используя табличный подзапрос в условии FROM, а затем подсчитывает строки, полученные в результате.

Итоги

Из данной главы вы узнали, зачем требуется объединять данные в группы, чтобы получить несколько итоговых подсумм для набора результата. После танталовых мучений с примером мы перешли к показу использования условия GROUP BY для решения этого примера и нескольких других. Также показали, как смешивать выражения со столбцами с агрегатными функциями.

Были проанализированы интересные примеры использования GROUP BY в подзапросе, который играет роль фильтра в условии WHERE. Построение запроса с GROUP BY и без использования агрегатных функций является тем же, что и использование DISTINCT в условии SELECT. Необходимо внимательно строить условия GROUP BY, чтобы включить столбцы, а не выражения.

Помните, что SQL не принимает во внимание какой-либо информации о первичных ключах. Мы также предупредили об ошибках, которые можно допустить при использовании выражений со столбцами в условии SELECT.

Условие GROUP BY является полезным, и мы представили примеры задач, которые можно решить с использованием GROUP BY. Затем были представлены примеры построения запросов, требующих использования условия GROUP BY.

Задачи для самостоятельного решения

Ниже приведены формулировки запросов и имена решений этих запросов в учебных базах данных. Попрактикуйтесь немного и разработайте SQL для каждого запроса, а затем сверьте свой ответ с запросом, который сохранен нами в этих базах данных. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов,— важно, чтобы набор результатов был тем же.

База данных заказов на закупку

1. *"Show me each vendor and the average by vendor of the number of days to deliver products"*.
   ("Показать каждого поставщика и среднее количество дней поставки товаров для поставщика".)
   (*Совет*: Используйте агрегатную функцию AVG и объединение в группу по поставщику.)
   Решение можно найти в Vendor_Avg_Delivery (10 строк).
2. *"Display for each product the product name and the total sales"*.
   ("Вывести на экран для каждого товара наименование товара и общий объем продаж".)
   (*Совет*: Используйте функцию SUM с вычислением количества, умноженного на цену и сгруппированного по наименованию товара.)
   Решение можно найти в Sales_By_Product (38 строк).

База данных агентства эстрадных мероприятий.

1. *"Show each agent's name, the sum of the contract price for the engagements booked, and the agent's total commission"*.
   ("Показать имя каждого агента, сумму цен контрактов для заявленных ангажементов и общую сумму комиссионного вознаграждения агента".)
   (*Совет*: Следует перемножить сумму цен контракта и комиссионное вознаграждение агента. Убедитесь, что группирование выполнено по ставке комиссионного вознаграждения!)
   Решение можно найти в Agent_Sales_And_Commissions (8 строк).

База данных лиги игры в боулинг

1. *"Display for each bowler the bowler name and the average of their raw game scores"*.
   ("Вывести на экран, для каждого игрока в боулинг, имя игрока и среднее количество очков за предварительные игры".)
   (*Совет*: Используйте агрегатную функцию AVG и объединение в группу по имени игрока в боулинг.)
   Решение можно найти в Bowler_Average (32 строки).

База данных расписания занятий

1. *"Display by category the category name and the count of classes offered"*.
   ("Вывести на экран дисплея список категорий с наименованием категории и количеством предложенных курсов лекций".)
   (*Совет*: Используйте функцию COUNT и объединение в группы по наименованию категории.)
   Решение можно найти в Category_Class_Count (16 строк).

2. *"List each staff member and the count of classes each is scheduled to teach"*.
   ("Привести список всех штатных сотрудников и количество курсов лекций, которые каждый из них запланировал прочитать".)
   (*Совет*: Используйте функцию COUNT и объединение в группы по имени преподавателя.)
   Решение можно найти в Staff_Class_Count (23 строки).

База данных рецептов

1. *"If I want to cook all the recipes in my cookbook, how much of each ingredient must I have on hand?"*
   ("Если попытаться приготовить все рецепты из поваренной книги, то какое количество каждого компонента необходимо иметь в запасе?")
   (*Совет*: Воспользуйтесь функцией SUM и выполните объединение в группы по названию компонента и описанию единиц измерения количества.)
   Решение можно найти в Total_Ingredients._Needed (65 строк).

14

Фильтрация сгруппированных данных

"Пусть педагоги ум терзают свой,
Грамматик правилами, вздором и ученьем;
Уверен я — от выпивки хорошей
Лишь глубже станет прозорливость духа".

— Оливер Голдсмит

Вопросы, рассматриваемые в данной главе:

- Сужение групп
- Фильтр: Почувствуйте разницу
- Использование HAVING
- Примеры операторов
- Итоги
- Задачи для самостоятельного решения

В главе 12 были представлены детали всех агрегатных функций, определенных в стандарте SQL. В главе 13 Мы показали, как запросить систему БД сгруппировать множества строк вместе, а затем высчитать агрегатные значения по каждой группе. Одно из преимуществ объединения в группу состоит в том, что можно также отобразить типизированное выражение на основе группирования столбцов для идентификации каждой группы.

В этой главе будет уложена на место последняя часть мозаики. После группирования строк и вычисления агрегатных функций часто бывает полезно выполнить дополнительную фильтрацию окончательного результата, используя предикат, по составным вычислениям. Для этого нам потребуется последняя деталь мозаики — условие HAVING.

Сужение групп

Собрав информацию в группы строк, можно запросить MIN, MAX, AVG, SUM или COUNT для всех значений в каждой группе. Предположим, что нужно еще более уточнить окончательный набор результатов — "сузить" группы,— проверяя одно из агрегатных значений. Рассмотрим простой запрос:

"Show me the entertainer groups who play in a jazz style and who have more than three members".
("Показать группы эстрадных артистов, играющих в джазовом стиле и состоящих более чем из трех человек".)

Выглядит не слишком сложно, не так ли? На рис. 14.1 представлены таблицы, необходимые для решения этого запроса.

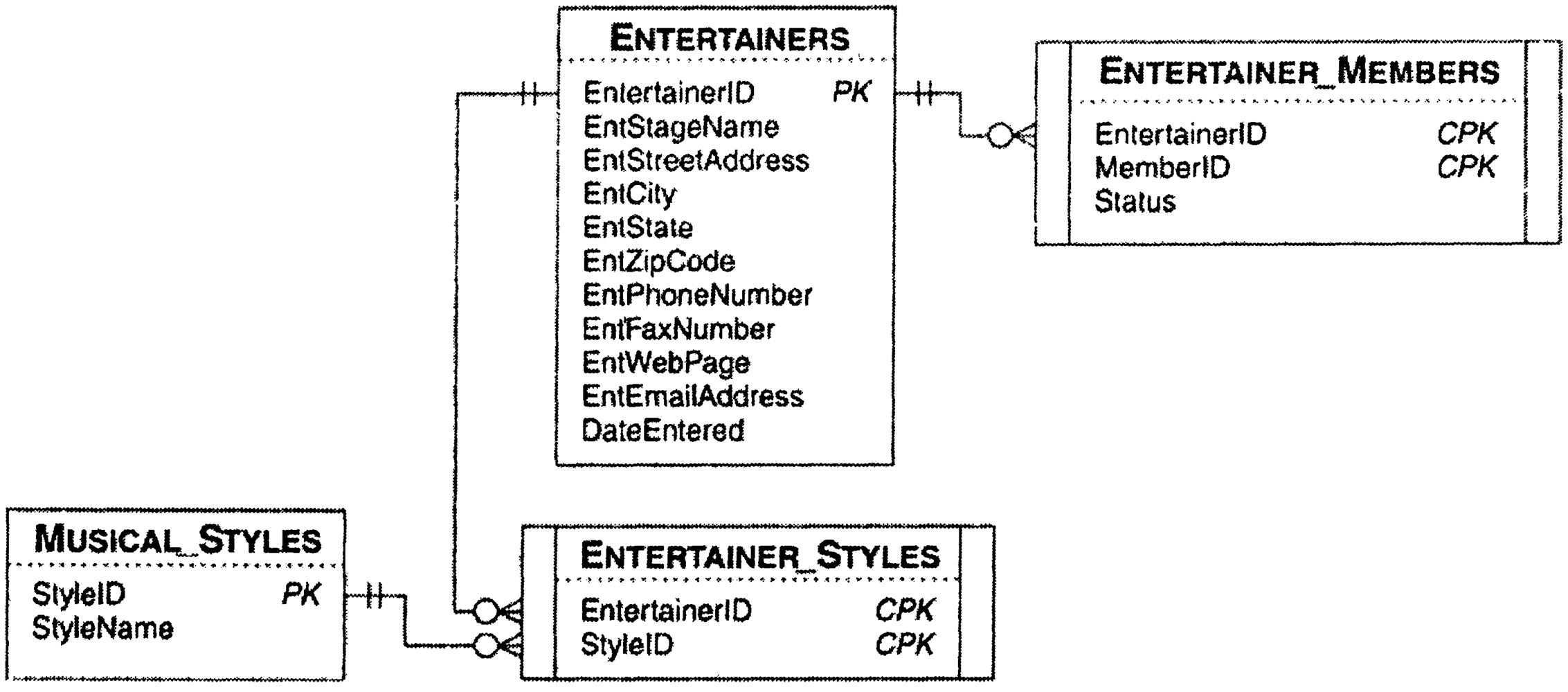


Рис. 14.1. *Таблицы, необходимые для отображения эстрадных артистов, исполняющих джаз в составе группы более трех человек*

Внимание! Снова воспользуемся методом "Запрос/Преобразование/Уточнение/SQL", который первоначально был введен в главе 4. Также используем методы JOIN (см. главы 8 и 9) и подзапросы (см. главу 11).

Не зная об условии HAVING, возможно, вас привлечет вариант решения задачи следующим *неверным* способом:

Преобразование: Select the entertainer stage name and the count of members from the entertainers table joined with the entertainer members table on entertainer ID where the entertainer ID is in the selection of entertainer IDs from the entertainer styles table joined with the musical styles table on style ID where the style name is 'Jazz' and where the count of the members is greater than 3, grouped by entertainer stage name
(Выбрать псевдоним эстрадного артиста и подсчитать количество участников из таблицы "Эстрадные артисты", соединенной с таблицей "Участники эстрадной группы" по идентификатору эстрадного артиста, где идентификатор

эстрадного артиста находится в выборке идентификаторов эстрадных артистов из таблицы “Стили эстрадных артистов”, соединенной с таблицей “Музыкальные стили” по идентификатору стиля, где название стиля — “Джаз” и где количество участников больше трех, сгруппированные по псевдониму эстрадного артиста)

Уточнение:

Select ~~the~~ entertainer stage name ~~and the~~ count(*) ~~of members~~ as Count of Members from ~~the~~ entertainers ~~table~~ join~~ed with the~~ entertainer members ~~table~~ on entertainer ID where ~~the~~ entertainer ID is in the (select~~ion of~~ entertainer ID~~s~~ from ~~the~~ entertainer styles ~~table~~ join~~ed with the~~ musical styles ~~table~~ on style ID where ~~the~~ style name ~~is~~ = ‘Jazz’) and ~~where the~~ count(*) ~~of the members is greater than~~ > 3, group~~ed~~ by entertainer stage name (Выбрать псевдоним эстрадного артиста, подсчитать (*) участников как Count of Members из “Эстрадные артисты”, соединенной с “Участники эстрадной группы” по идентификатору эстрадного артиста, где идентификатор эстрадного артиста в (Выбрать идентификатор эстрадного артиста из “Стили эстрадных артистов”, соединенной с “Музыкальные стили” по идентификатору стиля, где название стиля = ‘Jazz’) и подсчитать(*) > 3, сгруппированные по псевдониму эстрадного артиста)

SQL

```
SELECT Entertainers.EntStageName,
    COUNT(*) AS CountOfMembers
FROM Entertainers
INNER JOIN Entertainer_Members
ON Entertainers.EntertainerID =
    Entertainer_Members.EntertainerID
WHERE Entertainers.EntertainerID
IN
    (SELECT Entertainer_Styles.EntertainerID
    FROM Entertainer_Styles
    INNER JOIN Musical_Styles
    ON Entertainer_Styles.StyleID =
        Musical_Styles.StyleID
    WHERE Musical.Styles.StyleName = ‘Jazz’)
AND COUNT(*) > 3
GROUP BY Entertainers.EntStageName
```

Что же здесь неверно? Разгадка в том, что любой столбец, указанный в условии WHERE (см. главу 6), *должен* быть столбцом в одной из таблиц, определенных в условии FROM. Является ли COUNT(*) столбцом, сгенерированным в условии

FROM? Мы так не думаем! Фактически подсчет COUNT для каждой группы можно выполнить только после группирования строк.

Похоже, что нам требуется новое условие после GROUP BY. На рис. 14.2 представлен полный синтаксис оператора SELECT, включая новое условие HAVING.

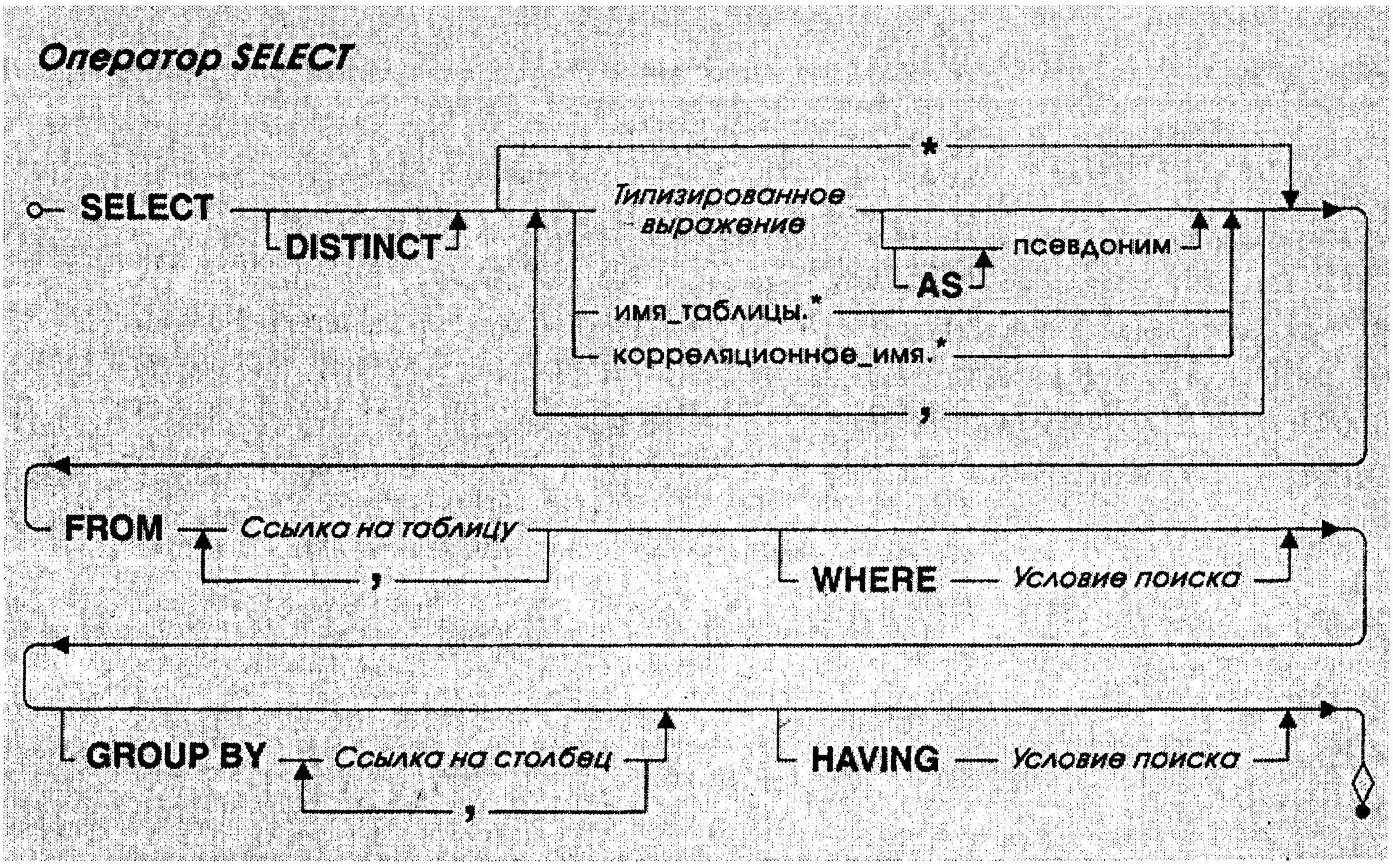


Рис. 14.2. *Оператор SELECT и все его условия*

Поскольку условие HAVING действует на строки *после* того, как они были сгруппированы, стандарт SQL устанавливает некоторые ограничения на столбцы, указываемые в любом предикате условия поиска. Обратите внимание, что, когда условие GROUP BY отсутствует, условие HAVING оперирует всеми строками, возвращенными условиями FROM и WHERE, как если бы они были одной группой.

Это такие же ограничения, как и для столбцов, указываемых в условии SELECT сгруппированного запроса. Любая ссылка на столбец в предикате внутри условия поиска в HAVING должна либо указать имя столбца, перечисленного в условия GROUP BY, либо должна быть вложена в агрегатную функцию. Это имеет смысл, поскольку любое сравнение столбцов должно использовать что-либо, сгенерированное из сгруппированных строк,— либо группирующее значение, либо составное вычисление по строкам в каждой группе.

Запишем решение для приведенной выше задачи правильным способом:

"Show me the entertainer groups who play in a jazz style and who have more than three members".
("Показать эстрадные группы, исполняющие джаз и состоящие более чем из трех человек".)

Преобразование: Select the entertainer stage name and the count of members from the entertainers table joined with the entertainer members table on entertainer ID where the entertainer ID is in the selection of entertainer IDs from the entertainer styles table joined with the musical styles table on style ID where the style name is "Jazz", grouped by entertainer stage name, and having the count of the members greater than 3
(Выбрать псевдоним эстрадного артиста и количество участников из таблицы "Эстрадные артисты", соединенной с таблицей "Участники эстрадной группы" по идентификатору эстрадного артиста, где идентификатор эстрадного артиста находится в выборке идентификаторов эстрадных артистов из таблицы "Стили эстрадных артистов", соединенной с таблицей "Музыкальные стили" по идентификатору стиля, где название стиля — "Джаз", сгруппированные по псевдониму эстрадного артиста и имеющие количество участников больше 3)

Уточнение: Select ~~the~~ entertainer stage name ~~and the~~ count(*) ~~of members~~ as CountOfMembers from ~~the~~ entertainers ~~table~~ join~~ed with the~~ entertainer members ~~table~~ on entertainer ID where ~~the~~ entertainer ID ~~is~~ in ~~the~~ (select~~ion of~~ entertainer ID~~s~~ from ~~the~~ entertainer styles ~~table~~ join~~ed with the~~ musical styles ~~table~~ on style ID where ~~the~~ style name ~~is~~ = "Jazz"), group~~ed~~ by entertainer stage name, and ~~having the~~ count(*) ~~of the members greater than~~ > 3
(Выбрать псевдоним эстрадного артиста, count(*) как CountOfMembers из "Эстрадные артисты", соединенной с "Участники эстрадной группы" по идентификатору эстрадного артиста, где идентификатор эстрадного артиста (выбрать идентификаторы эстрадного артиста из "Стили эстрадных артистов", соединенной с "Музыкальный стиль" по идентификатору стиля, где название стиля = "Джаз"), сгруппированные по псевдониму эстрадного артиста и count(*) > 3)

SQL
```
SELECT Entertainers.EntStageName,
    COUNT(*) AS CountOfMembers
FROM Entertainers
INNER JOIN Entertainer_Members
ON  Entertainers.EntertainerID =
    Entertainer_Members.EntertainerID
WHERE Entertainers.EntertainerID
```

```
IN
    (SELECT Entertainer_Styles.EntertainerID
     FROM Entertainer_Styles
     INNER JOIN Musical_Styles
     ON  Entertainer_Styles.StyleID =
         Musical_Styles.StyleID
     WHERE Musical_Styles.StyleName = 'Jazz')
GROUP BY Entertainers.EntStageName
HAVING COUNT(*) > 3
```

Хотя count (подсчет) был включен в окончательный вывод запроса, совсем не обязательно было это делать, чтобы запросить COUNT(*) в условии HAVING. До тех пор, пока какое-либо вычисленное значение или ссылка на столбец, используемые в условии HAVING, могут быть получены из сгруппированных строк, все хорошо. Приведенный выше запрос сохранен в учебной базе данных Entertainment Agency (База данных агентства эстрадных мероприятий) как Jazz_Entertainers_More_Than_3.

Фильтры: Почувствуйте разницу

Теперь вам известны два способа фильтрации окончательного набора результатов: WHERE и HAVING. Также известно, что существуют определенные ограничения на предикаты, которые можно использовать в условии поиска условия HAVING. Однако в некоторых случаях можно поместить предикат в любое из условий. Рассмотрим доводы в пользу помещения фильтра в условие WHERE вместо условия HAVING.

Размещать ли фильтр в условии WHERE или в HAVING?

Для фильтрации строк, возвращенных условием FROM запроса, можно построить пять основных типов предикатов: сравнение (=, <>, >=, <=), диапазон (BETWEEN), принадлежность множеству (IN), совпадение с образцом (LIKE) и Null (IS NULL). В главе 11 ваш кругозор был расширен и вы узнали, как использовать подзапрос в качестве одного из аргументов в предикатах сравнения и принадлежности множеству. Были также представлены два дополнительных класса предикатов — количественные (ANY, SOME, ALL) и предикаты существования (EXISTS),— которые требуют подзапроса как одного из аргументов.

Помните, что условие поиска в условии WHERE фильтрует строки *до того*, как СУБД объединит их в группы. В общем случае, когда нужно объединить в группу только подмножество строк, лучше вначале исключить ненужные строки в условии WHERE. Предположим, требуется решить следующую задачу:

> *"Show me the states on the west coast of the U.S. where the total of the orders is greater than $1 million".*
> *("Показать штаты на западном побережье США, для которых общая сумма заказов превышает 1 млн долларов".)*

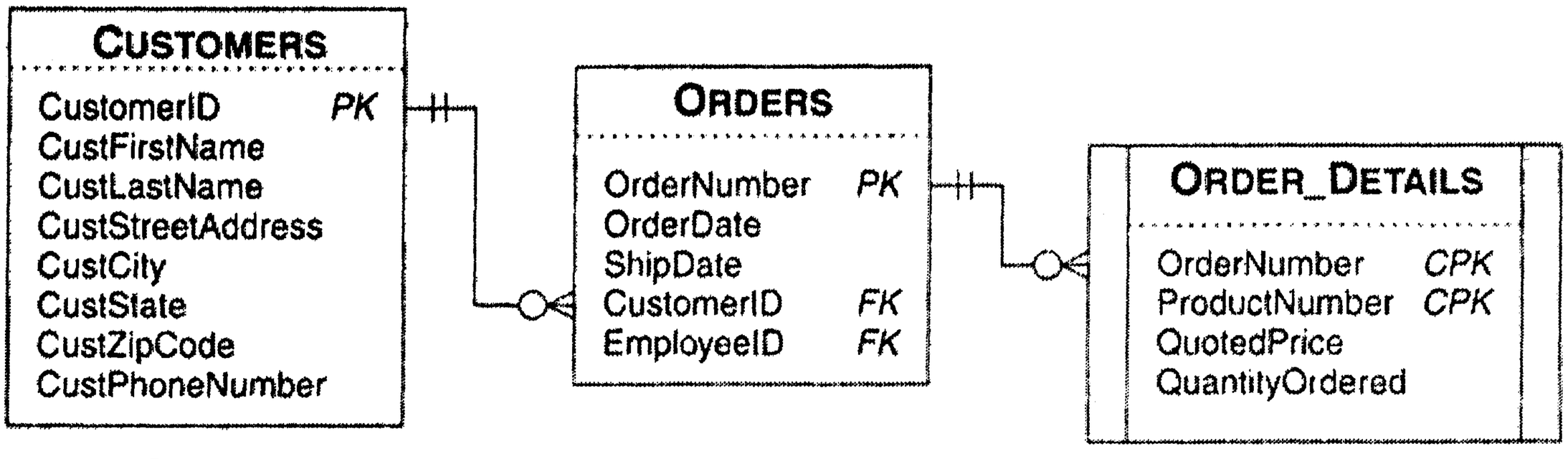


Рис. 14.3. *Таблицы, необходимые для суммирования всех заказов по штату*

На рис. 14.3 представлены таблицы, необходимые для решения этой задачи.

Этот запрос вполне законно можно записать следующим образом, поместив предикат по штату клиента в условие HAVING:

SQL

```
SELECT Customers.CustState,
    SUM(Order_Details.QuantityOrdered *
    Order_Details.QuotedPrice) AS SumOfOrders
FROM (Customers
    INNER JOIN Orders
    ON Customers.CustomerID = Orders.CustomerID)
    INNER JOIN Order_Details
    ON Orders.OrderNumber = Order_Details.OrderNumber
    CROUP BY Customers.CustState
HAVING SUM (Order_Details.QuantityOrdered *
    Order_Details.QuotedPrice) > 1000000
AND CustState IN ('WA', 'OR', 'CA')
```

Поскольку группирование производится по столбцу для штата, в условии HAVING по этому столбцу *можно* построить предикат, но при этом, возможно, системе баз данных придется проделать больше работы, чем необходимо. Как оказывается, итоговая сумма всех заказов для клиентов из штата Техас (который вовсе не на западном побережье) также превышает 1 млн долларов. Если установить фильтр по штату клиента в условии HAVING, как в приведенном выше примере, то база данных вычислит итоговую сумму также и для всех строк со штатом Техас, оценивая первый предикат в условии HAVING и сохраняя результат, а затем в конце отбрасывая его, когда выясняется, что группа для штата Техас не нужна.

Если нужно вычислить результат, основанный на группировании по штату клиента, но требуются клиенты только в Вашингтоне, Орегоне и Калифорнии, то лучше отфильтровать по строкам эти три штата с использованием условия WHERE, прежде чем запрашивать выполнение GROUP BY по штатам. Если сделать не так, условие FROM возвратит строки для всех клиентов во всех штатах, и потребуется выполнить дополнительную работу для группирования тех строк, которые совсем не нужны. Вот лучший способ решения этой задачи:

Преобразование: Select customer state and the sum of quantity ordered times quoted price as SumOfOrders from the customers table joined with the orders table on customer ID, and then joined with the order details table on order number where customer state is in the list: “WA”,"OR","CA", grouped by customer state, and having the sum of the orders greater than $ 1 million
(Выбрать штат клиента и сумму заказанного количества, умноженную на объявленную цену, как SumOfOrders из таблицы “Клиенты”, соединенной с таблицей “Заказы” по идентификатору клиента, а затем соединенной с таблицей “Детали заказа” по номеру заказа, где штат клиента находится в списке: “WA”,"OR","CA", сгруппированные по штату клиента и имеющие сумму заказов больше 1 млн долларов)

Уточнение: Select customer state ~~and the~~ sum ~~of~~ (quantity ordered ~~times~~ * quoted price) as SumOfOrders from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~and then~~ join~~ed with the~~ order details ~~table~~ on order number where customer state ~~is~~ in ~~the list~~ (“WA”,"OR","CA"), group~~ed~~ by customer state, ~~and~~ having ~~the~~ sum of the orders ~~greater than~~ > ~~$1 million~~ 1000000
(Выбрать штат клиента, сумму (заказанное количество * объявленная цена) как SumOfOrders из “Клиенты”, соединенной с “Заказы” по идентификатору клиента, соединенной с “Детали заказа” по номеру заказа, где штат клиента в: “WA”,"OR","CA", сгруппированные по штату клиента, имеющие сумму заказов > 1 000 000)

SQL
```
SELECT Customers.CustState,
   SUM(Order_Details.QuantityOrdered *
   Order_Details.QuotedPrice) AS SumOfOrders
FROM (Customers
   INNER JOIN Orders
   ON Customers.CustomerID = Orders. CustomerID)
INNER JOIN Order_Details
ON Orders.OrderNumber = Order_Details.OrderNumber
WHERE Customers.CustState IN ('WA', 'OR', 'CA')
GROUP BY Customers.CustState
HAVING SUM(Order_Details.QuantityOrdered *
   Order_Details.QuotedPrice) > 1000000
```

Этот запрос сохранен в учебной базе данных как West_Coast_Big_Order_States.

Обход ловушки в HAVING COUNT

Иногда требуется узнать, какие категории элементов имеют меньше, чем определенное количество элементов, например, какие эстрадные группы состоят из двух или менее участников, какие рецепты содержат два или менее молочных компонентов, какие предметы ведут три или менее профессоров, работающих на полную ставку. Хитрость здесь состоит в том, что *также* требуется узнать, какие категории имеют *ноль* элементов.

Рассмотрим запрос, показывающий ловушку, в которую можно попасть:

"Show me the subject categories that have three or fewer full professors teaching that subject".

На рис. 14.4 представлены таблицы, необходимые для решения этой задачи.

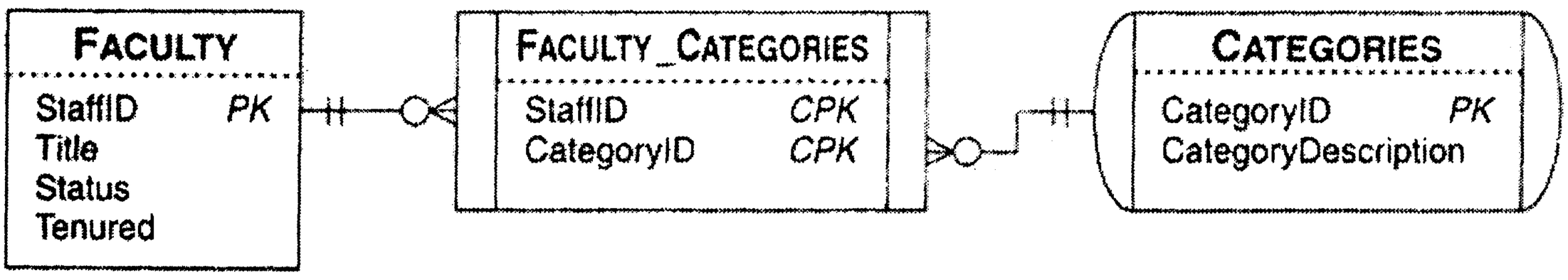


Рис. 14.4. *Категории предметов и профессорско-преподавательский состав, читающий по этим категориям*

Преобразование: Select category description and the count of staff ID as ProfCount from the categories table joined with the faculty categories table on category ID, and then joined with the faculty table on staff ID where title is 'Professor', grouped by category description, and having the count of staff ID less then 3
(Выбрать описание категории и подсчитать идентификаторы персонала как ProfCount из таблицы "Категории", соединенной с таблицей "Категории преподавателей" по идентификатору категории, а затем соединенной с таблицей "Преподаватели" по идентификатору персонала, где должность — 'профессор', сгруппированные по описанию категории и имеющие количество идентификаторов персонала меньше 3)

Уточнение: Select category description ~~and the~~ count ~~of~~ (staff ID) as ProfCount from ~~the~~ categories ~~table~~ join~~ed with the~~ faculty categories ~~table~~ on category ID, ~~and then~~ join~~ed with the~~ faculty ~~table~~ on staff ID where title ~~is~~ = 'Professor', group~~ed~~ by category description, ~~and~~ having ~~the~~ count ~~of~~ (staff ID) ~~less then~~ < 3

(Выбрать описание категории, подсчитать
(идентификатор персонала) как ProfCount из “Категории”,
соединенной с “Категории преподавателей”
по идентификатору категории, а затем соединенной
с “Преподаватели” по идентификатору персонала,
где должность = ‘профессор’, сгруппированные
по описанию категории, имеющие количество
(идентификатор персонала) < 3)

SQL

```
SELECT Categories.CategoryDescription,
    COUNT(Faculty_Categories.StaffID) AS ProfCount
FROM (Categories
INNER JOIN Faculty_Categories
ON Caregories.CategoryID =
    Faculty_Categories.CategoryID)
INNER JOIN Faculty
ON Faculty.StaffID = Faculty_Categories.StaffID
WHERE Faculty.Title = 'Professor'
GROUP BY Categories.CategoryDescription
HAVING COUNT(Faculty_Categories.StaffID) < 3
```

Выглядит неплохо, не так ли? Справа приведен набор результатов, возвращенный этим запросом.

Subjects_Fewer_3_Professors_WRONG

CategoryDescription	ProfCount
Accounting	1
Business	2
Computer Information Systems	1
Economics	1
Geography	1
History	1
Journalism	1
Math	1
Political Science	1

Заметили ли вы, что в наборе результатов *не указаны* категории предметов с нулевым числом профессоров? Причина этого кроется в том, что функция COUNT подсчитывает только строки, оставшиеся в таблице Faculty_Categories после фильтрации по всем профессорам, имеющим докторскую степень. Все потенциально нулевые строки были отброшены в условии WHERE!

Чтобы подтвердить наши подозрения о том, что в некоторых категориях нет профессоров, имеющих докторскую степень, построим запрос, который проверит наши теоретические предположения. Вспомните, что агрегатная функция COUNT возвращает ноль, если от нее запрашивается подсчет пустого множества, и можно получить пустое множество, если потребовать от запроса определить, сколько

существует строк для конкретной категории предмета. От запроса требуется рассматривать категории предмета по одной. Будем выполнять подсчет строк категорий, а не строк, содержащих сведения о преподавателях. Рассмотрим следующий оператор Select:

```
SQL                 SELECT COUNT(Faculty.StaffID)
                    AS BiologyProfessor
                    FROM (Faculty
                    INNER JOIN Faculty_Categories
                    ON Faculty.StaffID = Faculty_Categories.StaffID)
                    INNER JOIN Categories
                    ON Caregories.CategoryID =
                       Faculty_Categories.CategoryID
                    WHERE Categories.CategoryDescription = 'Biology'
                    AND Faculty.Title = 'Professor'
```

BiologyProfessors
0

Этот запрос сохранен в учебной базе данных как Count_Of_Biology_Professors. Как можно видеть, на самом деле в примере базы данных расписания нет профессора, имеющего докторскую степень и преподающего биологию. От запроса требовалось рассмотреть только одну категории предмета. Поскольку отсутствуют строки, имеющие как профессора, так и биологию, то вполне законно получено пустое множество. Поэтому функция COUNT возвращает ноль.

Теперь можно вложить этот запрос как подзапрос в условие WHERE, которое извлекает совпадения по идентификатору категории из внешнего запроса. Это заставляет запрос рассматривать категории по одной, по мере извлечения одной строки с описанием категории из таблицы "Категории" во внешнем запросе. SQL будет следующий:

```
SQL                 SELECT Categories.CategoryDescription,
                       (SELECT COUNT(Faculty.StaffID)
                       FROM (Faculty
                       INNER JOIN Faculty_Categories
                       ON Faculty.StaffID = Faculty_Categories.StaffID)
                       INNER JOIN Categories AS C2
                       ON C2.CategoryID = Faculty_Categories.CategoryID
                       WHERE C2.CategoryID = Categories.CategoryID
                       AND Faculty.Tit1e = 'Professor')
                       AS ProfCount
                    FROM Categories
                    WHERE
                       (SELECT COUNT(Faculty.StaffID)
                       FROM (Faculty
```

```
INNER JOIN Faculty_Categories
ON Faculty.StaffID = Faculty_Categories.StaffID)
INNER.JOIN Categories AS C3
ON C3.CategoryID = Faculty_Categories.CategoryID
WHERE C3.CategoryID = Categories.CategoryID
AND Faculty.Title = 'Professor') < 3
```

Этот запрос сохранен в учебной БД как Subjects_Fewer_3_Professors_RIGHT. В условие SELECT также включена копия подзапроса, чтобы можно было видеть реальные количества по категориям. В этот раз все работает правильно, потому что подзапрос в условии WHERE с полным правом возвращает ноль для категории, в которой нет профессоров с докторской степенью. Правильный результат приведен справа.

Subjects_Fewer_3_Professors_RIGHT

CategoryDescription	ProfCount
Accounting	1
Biology	0
Business	2
Chemistry	0
Computer Information Systems	1
Computer Science	0
Economics	1
Geography	1
History	1
Journalism	1
Math	1
Physics	0
Political Science	1
Psychology	0
French	0
German	0

Как можно видеть, многие категории предметов в действительности *не имеют* профессоров с докторской степенью, назначенных для преподавания этого предмета. Хотя это окончательное решение вообще не использует HAVING, оно было включено, чтобы вы осознали, что HAVING — не всегда явное решение для задач подобного типа. HAVING можно использовать для решения запросов вида "...имеют меньше, чем...". Например, если нужно увидеть всех клиентов, затративших менее 500 долларов в прошлый месяц, но не интересуют клиенты, которые вообще ничего не покупали, то решение с использованием HAVING работает просто прекрасно (и, скорее всего, будет выполняться быстрее). Однако если необходимо увидеть клиентов, которые ничего не покупали, нужно использовать метод без HAVING, который был только что представлен.

Использование HAVING

На данный момент вы должны хорошо понимать, как запрашивать подсуммы для групп, используя агрегатные функции и условие GROUP BY, и как отфильтровать сгруппированные данные, используя HAVING. Самый лучший способ проиллюстрировать диапазон применения HAVING — это показать задачи, которые можно решать, используя это новое условие.

"Показать каждого поставщика и среднее для него количество дней доставки товара, которое больше среднего количества дней доставки для всех поставщиков".

"Вывести на экран для каждого товара наименование товара и общее количество продаж, которое больше, чем среднее количество продаж для всех товаров этой категории".

"Привести список всех клиентов и даты их заказов, полное имя клиента и общую стоимость заказанных позиций, которая больше 1000 долларов".

"Сколько имеется заказов только на один товар?"

"Какие агенты сделали заявки более чем на 3000 долларов в декабре 1999 г.?"

"Показать эстрадных артистов, у которых имеется более двух перекрывающихся заявок".

"Показать имя каждого агента, сумму цен контрактов для забронированных ангажементов и общую сумму комиссионных для агентов, у которых эта сумма превышает 1000 долларов".

"Имеют ли какие-либо капитаны команд предварительное количество очков выше, чем у любого другого участника команды?"

"Отобразить на экране дисплея для каждого игрока в боулинг имя игрока и среднее количество очков за игру для игроков, среднее количество очков которых больше 155".

"Привести список игроков в боулинг, у которых наибольшее среднее предварительное количество очков по крайней мере на 20 кеглей больше, чем их текущее среднее значение".

"Для завершенных курсов лекций привести список по категориям и студентам, включающий наименование категории, имя студента и средний балл студента по всем прослушанным курсам лекций в этой категории для тех студентов, которые имеют средний балл 90 или выше".

"Вывести на экран дисплея для каждой категории наименование категории и число курсов лекций, предложенных для тех категорий, которые имеют три и более курсов лекций".

"Привести список штатных сотрудников и количество курсов лекций, которое каждый из них запланировал для прочтения, для тех штатных сотрудников, которые преподают по крайней мере один, но менее трех курсов лекций".

"Привести список рецептов, содержащих как говядину, так и чеснок".

"Подсчитать общее количество соли по видам рецептов и вывести на экран те виды рецептов, которые требуют более трех чайных ложек".

"Для какого типа рецептов имеется два или более рецептов?"

Примеры операторов

Теперь ознакомимся с множеством примеров, для которых вначале требуется группирования информации, а затем фильтрация по составному значению для группы. Они взяты из каждой учебной базы данных.

Сюда также включены примеры наборов результатов, которые должны возвращать эти операции. Они располагаются сразу же после записи строк SQL. Имя, которое появляется непосредственно над набором результатов, присвоено каждому запросу в учебной базе данных. Каждый запрос сохранен в соответствующем примере базы данных во вложенной папке "Chapter 14" на сайте издательства "Лори". Чтобы загрузить примеры на свой компьютер и проверить их, следуйте указаниям, приведенным в начале этой книги.

> **Внимание!** Все имена столбцов и таблиц, используемые в этих примерах, взяты из учебных структур баз данных, представленных в приложении B. Для упрощения процесса этапы преобразования и уточнения объединены. Предполагается, что вами тщательно изучены и поняты концепции, рассмотренные в предыдущих главах, особенно в главах по JOIN и подзапросам.

База данных заказов на закупку

"List for each customer and order date the customer full name and the total cost of items ordered that is greater than $1,000". ("Привести список всех клиентов и даты их заказов, полное имя клиента и общую стоимость заказанных позиций, которая превышает 1000 долларов".)

Преобразование/ Уточнение:	Select customer first name ~~and~~ || ' ' || customer last name as CustomerFullName, ordered date, ~~and the~~ sum ~~of~~ (quoted price ~~times~~ * quantity ordered) as TotalCost from ~~the~~ customers ~~table~~ join~~ed with the~~ orders ~~table~~ on customer ID, ~~and then~~ join~~ed with the~~ order details ~~table~~

on order number, group~~ed~~ by customer first name, customer last name, ~~and~~ order date, having ~~the~~ sum ~~of~~ (quoted price ~~times~~ * quantity ordered) ~~greater than~~ > 1000
(Выбрать имя клиента || ' ' ||, фамилию клиента как CustomerFullName, дату заказа, sum(объявленная цена * заказанное количество) как TotalCost из "Клиенты", соединенной с "Заказы" по идентификатору клиента, соединенной с "Детали заказа" по номеру заказа, сгруппированные по имени клиента, фамилии клиента, дате заказа, имеющие sum(объявленная цена * заказанное количество) > 1000)

SQL

```
SELECT Customers.CustFirstName || ' ' ||
    Customers.CustLastName AS CustFul1Name,
    Orders.OrderDate,
    SUM(Order_Details.QuotedPrice *
    Order.Details.QuantityOrdered) AS TotalCost
FROM (Customers
INNER JOIN Orders
ON Customers.CustomerID = Orders.CustomerID)
INNER JOIN Order_.Details
ON Orders.OrderNumber = Order_Details.OrderNumber
GROUP BY Customers.CustFirstName,
    Customers.CustLastName, Orders.OrderDate
HAVING SUM(Order_Details.QuotedPrice *
    Order._Details.QuantityOrdered) > 1000
```

Order_Totals_By_Customer_And_Date_GT1000 (649 строк)

CustFullName	OrderDate	TotalCost
Alaina Hallmark	1999-07-02	$4,699.98
Alaina Hallmark	1999-07-14	$4,433.95
Alaina Hallmark	1999-07-21	$3,951.90
Alaina Hallmark	1999-07-22	$10,388.68
Alaina Hallmark	1999-07-30	$3,088.00
Alaina Hallmark	1999-08-11	$6,775.06
Alaina Hallmark	1999-08-21	$15,781.10
Alaina Hallmark	1999-08-29	$15,969.50
<< остальные строки >>		

База данных агентства эстрадных мероприятий

"Which agents booked more than $3000 worth of business in December 1999?"
("Какие агенты сделали заявок более чем на 3000 долларов в декабре 1999 г.?")

Преобразование/ Уточнение:

Select ~~the~~ agent first name, agent last name, ~~and the~~ sum ~~of~~ (contract price) as TotalBooked from ~~the~~ agents ~~table~~ join~~ed with the~~ engagements ~~table~~ on agent ID where ~~the~~ engagement start date ~~is~~ between ~~December 1, 1999,~~ '1999-12-01' and ~~December 31, 1999,~~ '1999-12-31' having ~~the~~ sum ~~of~~ (contract price) ~~greater than~~ > $3000
(Выбрать имя агента, фамилию агента, sum (цена контракта) как TotalBooked из "Агенты", соединенной с "Ангажементы" по идентификатору агента, где дата начала ангажемента между '1999-12-01' и '1999-12-31', имеющие sum (цена контракта) > $3000)

SQL

```
SELECT Agents.AgtFirstName, Agents.AgtLastName,
    SUM(Engagements.ContractPrice) AS Total Booked
FROM Agents
INNER JOIN Engagements
ON Agents.AgentID = Engagements.AgentID
WHERE Engagements.StartDate
BETWEEN '1999-12-01' And '1999-12-31'
GROUP BY Agents.AgtFirstName, Agents.AgtLastName
HAVING SUM(Engagements.ContractPrice) > 3000
```

Agents_Book_Over_3000_12_99 (4 строки)

AgtFirstName	AgtLastName	TotalBooked
Gregory	Piercy	$4,785.00
Margaret	Peacock	$4,350.00
Mary	Fuller	$7,120.00
Will	Thompson	$6,555.00

База данных лиги игры в боулинг

"List the bowlers whose highest raw score is at least 2 pins higher than their current average".
("Привести список игроков в боулинг, для которых наибольшее среднее предварительное количество очков по крайней мере на 20 кеглей больше, чем их текущее среднее значение".)

Преобразование/ Уточнение:

Select bowler first name, bowler last name, current average, ~~and the~~ max~~imum~~ (raw score) as HighGame from ~~the~~ bowlers ~~table~~ join~~ed with the~~ bowler scores ~~table~~ on bowler ID, group~~ed~~ by bowler first name, bowler last name, ~~and~~ bowler current average, having ~~the~~ max~~imum~~ (raw score) ~~greater than the~~ > current average ~~plus~~ + 20
(Выбрать имя игрока в боулинг, фамилию игрока в боулинг, текущее среднее, max (предварительное количество очков) как HighGame из “Игроки в боулинг”, соединенной с “Очки игрока в боулинг” по идентификатору игрока, сгруппированные по имени игрока, фамилии игрока, текущему среднему игрока, имеющему max (предварительное количество очков) > текущее среднее + 20)

SQL

```
SELECT Bowlers.BowlerFirstName,
    Bowlers.BowlerLastName,
    Bowlers.CurrentAverage,
    MAX(Bowler_Scores.RawScore) AS HighGame
FROM Bowlers
INNER JOIN Bowler_Scores
ON Bowlers.BowlerID = Bowler_Scores.BowlerID
GROUP BY Bowlers.BowlerFirstName,
    Bowlers.BowlerLastName,
    Bowlers.CurrentAverage
HAVING MAX(Bowler_Scores.RawScore) >
    (Bowlers.CurrentAverage+20)
```

Bowlers_Big_High_Score (15 строк)

BowlerFirstName	BowlerLastName	CurrentAverage	HighGame
Alaina	Hallmark	158.00	180
Carol	Viescas	168.00	195
David	Fournier	157.00	178
Gary	Hallmark	157.00	179
Greg	Piercy	164.00	193
John	Kennedy	166.00	191
John	Viescas	168.00	193
Kathryn	Patterson	162.00	191
<< остальные строки >>			

Внимание! Таблица Bowlers (Игроки в боулинг) содержит в себе столбец CurrentAverage (Текущее среднее), и мы воспользовались этим преимуществом. Чтобы работать со структурой этой таблицы, приложение, которое ее использует, должно повторно вычислять и корректировать указанный столбец каждый раз, когда для игрока в боулинг вводятся новые игры. Но она была спроектирована таким образом с целью избежать повторных вычислений среднего значения, возможно, по сотне игр каждый раз, когда нам потребуется это значение. Также можно было вычислить абсолютное текущее среднее, используя подзапрос по таблице Bowler_Scores, но это выполнялось бы медленнее. Подробнее о "нарушениях правил" проектирования БД можно узнать из книги *Database Design for Mere Mortals* (Reading, MA: Addison-Wesley, 1997).

База данных расписания занятий

"For completed classes, list by category and student the category name, the student name, and the student's average grade of all classes taken in that category for those students who have an average of 90 or better".
("Для завершенных курсов лекций привести список по категории и студенту, включающий наименование категории, имя студента и средний балл студента по всем прослушанным курсам лекций в этой категории для тех студентов, которые имеют средний балл 90 или выше".

Преобразование/ Уточнение:

Select category description, student first name, student last name, ~~and the average~~ avg(~~of~~ grade) as AvgOfGrade from ~~the~~ categories ~~table~~ join~~ed with the~~ subjects ~~table~~ on category ID, ~~then~~ join~~ed with the~~ classes ~~table~~ on subject ID, ~~then~~ join~~ed with the~~ student schedules ~~table~~ on class ID, ~~then~~ join~~ed with the~~ student class status ~~table~~ on class status, ~~and finally~~ join~~ed with the~~ students ~~table~~ on student ID where class status description ~~is~~ = 'Completed', group~~ed~~ by category description, student first name, student last name, ~~and~~ having ~~the average~~ avg(~~of~~ grade) ~~greater than~~ > 90

(Выбрать описание категории, имя студента, фамилию студента, avg (средний балл) как AvgOfGrade из "Категории", соединенной с "Предметы" по идентификатору категории, соединенной с "Расписание занятий студента" по идентификатору класса, соединенной со "Студенты" по идентификатору студента, где описание состояния курса лекций = "Завершено", сгруппированные по описанию категории, имени студента, фамилии студента, имеющего avg (средний балл) > 90)

SQL

```
SELECT Categories.CategoryDescription,
    Students.StudFirstName, Students.StudLastName,
    AVG(Student_Schedules.Grade) AS AvgOfGrade
FROM ((((Categories
INNER JOIN Subjects
ON Categories.CategoryID = Subjects.CategoryID)
INNER JOIN Classes
ON Subjects.SubjectID = Classes.SubjectID)
INNER JOIN Student_Schedules
ON Classes.ClassID = Student_Schedules.ClassID)
INNER JOIN Student_Class_Status
ON Student_Class_Status.ClassStatus =
    Student_Schedules.ClassStatus)
INNER JOIN Students
ON Students.StudentID = Student_Schedules.StudentID
WHERE Student_Class_Status.ClassStatusDescription =
    'Completed'
GROUP BY Categories.CategoryDescription,
    Students.StudFirstName, Students.StudLastName
HAVING AVG(Student_Schedules.Grade) > 90
```

A_Students (13 строк)

CategoryDescription	StudFirstName	StudLastName	AvgOfGrade
Accounting	Elizabeth	Hallmark	90.24
Accounting	Michael	Viescas	90.01
Accounting	Sarah	Leverling	90.67
Art	Kendra	Bonnicksen	90.63
Art	Sarah	Leverling	91.72
English	Elizabeth	Hallmark	92.90
English	John	Kennedy	93.70
English	Mel	Ehrlich	93.86
English	Sarah	Thompson	97.39
Music	Karen	Smith	92.08
Music	Mary	Fuller	98.26
Music	Mel	Ehrlich	93.26
Music	Nancy	Davolio	93.28

"List each staff member and the count of classes each is scheduled to teach for those staff members who teach at least one but fewer than three classes". ("Привести список штатных сотрудников и количество курсов лекций, которое каждый из них запланировал для преподавания, для тех сотрудников, которые преподают по крайней мере один, но менее трех курсов лекций".)

Внимание! Проблемы с нулем в HAVING COUNT удалось избежать вследствие конкретного указания, что нужны штатные преподаватели, читающие по крайней мере один курс лекций.

Преобразование/ Уточнение:

Select staff first name, staff last name, ~~and the~~ count ~~of classes~~ (*) as ClassCount from ~~the~~ staff ~~table~~ join~~ed~~ ~~with the~~ faculty classes ~~table~~ on staff ID, group~~ed~~ by staff first name ~~and~~ staff last name, ~~and~~ having ~~the~~ count ~~of classes~~ (*) ~~less than~~ < 3
(Выбрать имя преподавателя, фамилию преподавателя, count(*) как ClassCount из "Штатные преподаватели", соединенной с "Курсы лекций факультета" по идентификатору преподавателя, сгруппированные по имени преподавателя, фамилии преподавателя, имеющие count(*) < 3)

SQL

```
SELECT Staff.StfFirstName, Staff.StfLastName,
    Count(*) AS ClassCount
FROM Staff
INNER JOIN Faculty_Classes
ON Staff.StaffID = Faculty.Classes.StaffID
GROUP BY Staff.StfFirstName, Staff.StfLastName
HAVING COUNT(*) < 3
```

Staff_Class_Count_1_To_3 (8 строк)

StfFirstName	StfLastName	ClassCount
Allan	Davis	1
David	Callahan	2
James	Leverling	2
Joyce	Bonnicksen	2
Katherine	Ehrlich	2
Laura	Callahan	2
Ryan	Ehrlich	2
Suzanne	Viescas	2

База данных рецептов

"List the recipes that contain both beef and garlic".
("Привести список рецептов, содержащих как говядину, так и чеснок".)

Преобразование/ Уточнение:

Select recipe title from ~~the~~ recipes ~~table~~ where ~~the~~ recipe ID ~~is~~ in ~~the~~ (select~~ion of~~ recipe ID from ~~the~~ ingredients ~~table~~ join~~ed with the~~ recipe ingredients ~~table~~ on ingredient ID where ~~the~~ ingredient name ~~is~~ = 'Beef' or ~~the~~ ingredient name ~~is~~ = 'Garlic', group~~ed~~ by recipe ID ~~and~~ having ~~the~~ count (*) ~~of rows equal to~~ = 2
(Выбрать заголовок рецепта из "Рецепты", где идентификатор рецепта в (выбрать идентификатор рецепта из "Компоненты", соединенной с "Компоненты рецепта" по идентификатору компонента, где название компонента = 'говядина' или название компонента = 'чеснок', сгруппированные по идентификатору рецепта, имеющие count (*) = 2)

SQL

```
SELECT Recipes.RecipeTitle
FROM Recipes
WHERE Recipes.RecipeID
   IN (SELECT Recipe_Ingredients.RecipeID
       FROM Ingredients
       INNER JOIN Recipe_Ingredients
       ON  Ingredients.IngredientID =
           Recipe_Ingredients.IngredientID
       WHERE Ingredients.IngredientName 'Beef'
       OR  Ingredients.IngredientName = 'Garlic'
       GROUP BY Recipe_Ingredients.RecipeID
       HAVING COUNT(Recipe_Ingredients.RecipeID) = 2)
```

Внимание! Здесь показано творческое использование GROUP BY и HAVING в подзапросе для поиска рецептов, которые содержат два компонента. Если рецепт не содержит ни одного из компонентов, COUNT будет иметь значение ноль. Только когда рецепт содержит оба компонента, COUNT будет равно 2. Тем не менее будьте осторожны. Если в конкретном рецепте предусматривается содержание как фарша, так и чеснока, но не говядины, то этот метод не работает! Будет получен ответ 2 для двух записей для чеснока, поэтому рецепты буду выбраны, даже если они не содержат говядину. Если вас удивляет использование оператора OR, когда нужно наличие как говядины, так и чеснока, обратитесь повторно к теме "Использование OR" в разделе "Использование нескольких условий" главы 6. Альтернативный способ решения этой задачи был показан в главе 8.

Recipes_Beef_And_Garlic (1 строка)

RecipeTitle
Roast Beef

Итоги

В данной главе мы обсудили "сужение" групп, составляемых с помощью условия HAVING, для фильтрации групп, основанных на составных вычислениях. Был представлен и пояснен на простых примерах синтаксис оператора SELECT для этого последнего условия.

Затем мы привели пример использования для фильтрации строк условия WHERE, а не HAVING. Когда имеется выбор, лучше поместить фильтр в условие WHERE. Мы показали, как избежать обычной ловушки при подсчете групп, которые могут содержать нулевой результат, а также альтернативный способ решения таких задач.

Условие HAVING является полезным и помогает решить многие задачи. Были приведены примеры построения запросов, которые требуют использования этого условия.

Задачи для самостоятельного решения

Ниже приведены формулировки запросов и имена решений этих запросов в учебных базах данных. Попрактикуйтесь немного и разработайте SQL для каждого запроса, а затем сверьте свой ответ с запросом, который сохранен нами в этих БД. Не беспокойтесь, если ваш синтаксис не совсем точно совпадает с синтаксисом сохраненных запросов — важно, чтобы набор результатов был тем же.

База данных заказов на закупку

1. *"Show me each vender and the average by vendor of the number of days to deliver products that is greater than the average delivery days for all vendors"*.
   ("Показать каждого поставщика и среднее для него количество дней доставки товара, которое больше среднего количества дней доставки для всех поставщиков".)
   (*Совет*: Необходимо воспользоваться подзапросом для извлечения среднего времени доставки для всех поставщиков.)
   Решение можно найти в Vendor_Avg_Delivery_GT_Overall_Avg (5 строк).

2. *"Display for each product the product name and the total sales that is greater than the average of sales for all products in that category"*.
   ("Вывести на экран для каждого товара наименование товара и общее количество продаж, которое больше, чем среднее количество продаж для всех товаров этой категории".)

(*Совет*: Чтобы вычислить значение для сравнения, вначале требуется выполнить SUM для продаж в пределах категории, а затем AVG этих сумм по категориям.)
Решение можно найти Sales_By_Product_GT_Category Avg (13 строк).

3. *"How many orders are for only one product?"*
("Сколько имеется заказов только на один товар?")
(*Совет*: Необходимо использовать внутренний запрос, который выводит список номеров заказов для заказов, имеющих только одну строку, а затем COUNT этих строк.)
Решение можно найти в Single_Item_Order_Count (1 строка).

База данных агентства эстрадных мероприятий

1. *"Show me the entertainers who have more then two overlapped booking"*.
("Показать эстрадных артистов, у которых имеется более двух перекрывающихся заявок".)
(*Совет*: Используйте подзапрос для поиска эстрадных артистов с перекрывающимися заявками, HAVING (имеющих) их COUNT (количество) больше 2.)
Решение можно найти в Entertainers_MoreThan_2_Overlap (1 строка).

2. *"Show each agent's name, the sum of the contract price for the engagements booked, and the agent's total commission for agents whose total commission is more than $1000"*.
("Показать имя каждого агента, сумму цен контрактов для забронированных ангажементов и общую сумму комиссионных для агентов, у которых эта сумма превышает 1000 долларов".)
(*Совет*: Воспользуйтесь решением подобной задачи из предыдущего раздела и добавьте к нему условие HAVING.)
Решение можно найти в Agent_Sales_Big_Commissions (4 строки).

База данных лиги игры в боулинг

1. *"Do any team captains have a raw score that is higher than any other member on the team?"*
("Имеют ли какие-либо капитаны команд предварительное количество очков выше, чем у любого другого участника команды?")
(*Совет*: Наибольшее количество предварительных очков для капитанов можно найти, выполнив операцию JOIN команд и игроков по идентификатору капитана, а затем с очками игроков в боулинг. Воспользуйтесь условием HAVING для сравнения значения MAX для всех других элементов из подзапроса.)
Решение можно найти в Captains_Who_Are_Hotshots (0 строк).
(Оказалось, что ни один капитан не является лучшим в команде!)

2. *"Display for each bowler the bowler name and the average of their raw game scores for bowlers whose average is greater than 155."*
("Отобразить на экране для каждого игрока в боулинг имя игрока и среднее количество очков за игру для игроков, у которых это среднее количество больше 155".)
(*Совет*: Нужно простое условие HAVING, сравнивающее AVG с числовым литералом.)
Решение можно найти в Good_Bowlers (16 строк).

База данных расписания занятий

1. *"Display by category the category name and the count of classes offered for those categories that have three or more classes"*.
("Вывести на экран дисплея для каждой категории наименование категории и число курсов лекций, предложенных для тех категорий, которые имеют три и более курсов лекций".)
(*Совет*: Воспользуйтесь JOIN категории по предметам, а затем по курсам лекций, COUNT строки и добавьте условие HAVING для получения окончательного результата.)
Решение можно найти в Category_Class_Count_3_Or_More (11 строк).

2. *"List each staff member and the count of classes each is scheduled to teach for those staff members who teach fewer than three classes"*.
("Привести список штатных сотрудников и количество курсов лекций, которое каждый из них запланировал прочитать, для тех сотрудников, которые преподают менее трех курсов лекций".)
(*Совет*: Это ловушка с нулем в HAVING COUNT! Вместо них используйте подзапросы!)
Решение можно найти в Staff_Teaching_LessThan_3 (12 строк).

База данных рецептов

1. *"Sum the amount of salt by recipe class and display those recipe classes that require more than 3 teaspoons"*.
("Подсчитать общее количество соли по видам рецептов и вывести на экран те виды рецептов, для которых требуется более трех чайных ложек соли".)
(*Совет*: Для этого необходима сложная операция JOIN пяти таблиц, чтобы отфильтровать соль и чайные ложки, SUM результата, а затем исключить те виды рецептов, в которых используется более 3 чайных ложек соли.)
Решение можно найти в Recipe_Classes_Lots_Of_Salt (1 строка).

2. *For what class of recipe do I have two or more recipes?"*
("Для какого типа рецептов имеется два или больше рецептов?")
(*Совет*: JOIN (соедините) виды рецептов с рецептами, подсчитайте результат и, воспользовавшись условием HAVING, сохраните виды, имеющие два или более рецепта.)
Решение можно найти в Recipe Classes Two Or More (4 строки).

Заключение

“Именно в этом состоит познание.
Вы внезапно понимаете что-то,
о чем вам было известно всю вашу жизнь,
но уже с новой стороны”.

— Дорис Лессинг

К настоящему моменту вы изучили все инструменты, необходимые для успешного обращения с запросом к базе данных. Мы создавали как простые, так и сложные операторы SELECT и работали с данными различного типа. Нами рассмотрены: порядок фильтрации данных в условиях поиска; работа с несколькими таблицами с использованием операций JOIN; формирование статистической информации посредством объединения данных в группы.

Как при любых попытках освоения чего-то нового всегда многое остается для последующего изучения. Ваша очередная задача состоит в том, чтобы методы, изученные в этой книге, применить к своей системе базы данных. Не забывайте справляться в документации по конкретной базе данных о наличии или отсутствии каких-либо различий между стандартным синтаксисом SQL и синтаксисом SQL, используемым базой данных. Если ваша база данных позволяет создавать запросы с помощью графического интерфейса, то вы увидите, что интерфейс стал более осмысленным и пользоваться им намного легче.

Наше внимание в основном было сосредоточено на той части SQL, которая связана с манипулированием данными. В SQL все еще имеется много областей, в которые можно углубиться, если в этом есть необходимость. Например, можно изучить, как создавать структуры данных, включающие в представление несколько таблиц, или вставлять операторы SQL в прикладную программу. Если есть стремление к дальнейшему изучению SQL, советуем начать его с любой из книг, перечисленных в приложении C.

Надеемся, что вы получили такое же удовольствие от чтения этой книги, какое получили мы в процессе ее написания. Зная, что книги на подобные темы чаще всего получаются скучными, мы старались сделать ее немного более забавной и добавить, по возможности, немного юмора. Совсем не обязательно, чтобы процесс обучения

был скучным и утомительным. Напротив, стоит жить в предвкушении того, что каждый новый день позволит вам узнать что-то новое.

Написание книги всегда дает понять, насколько больше требуется знать о рассматриваемом предмете. И во время работы над книгой неизбежно рассмотрение вещей с новой точки зрения и в другом свете. Обнаружилось, сколько правды в утверждении Дорис Лессинг.

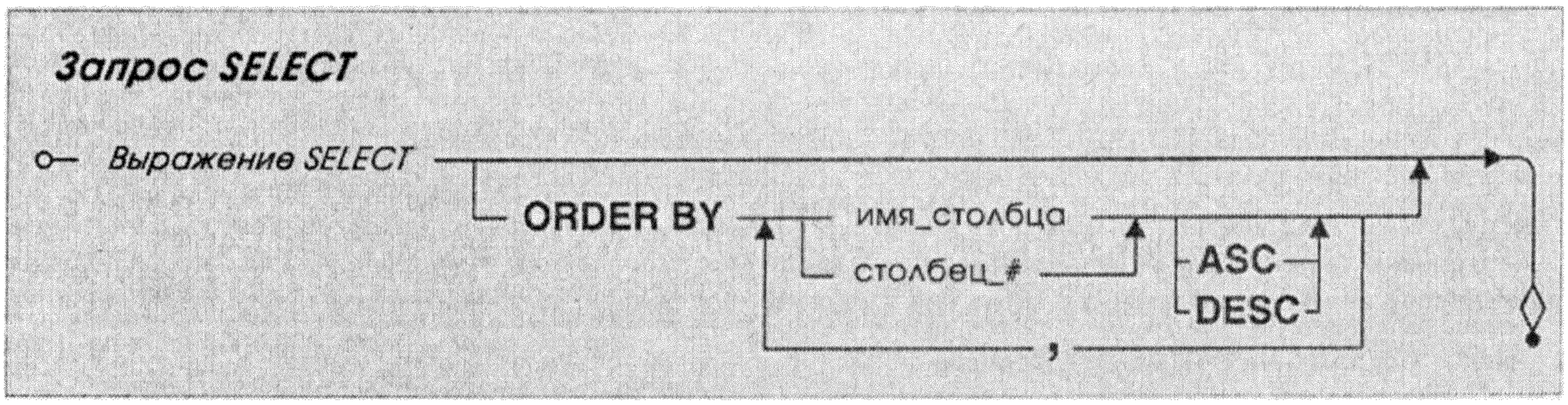
Надеемся, что вы также согласны с этим.

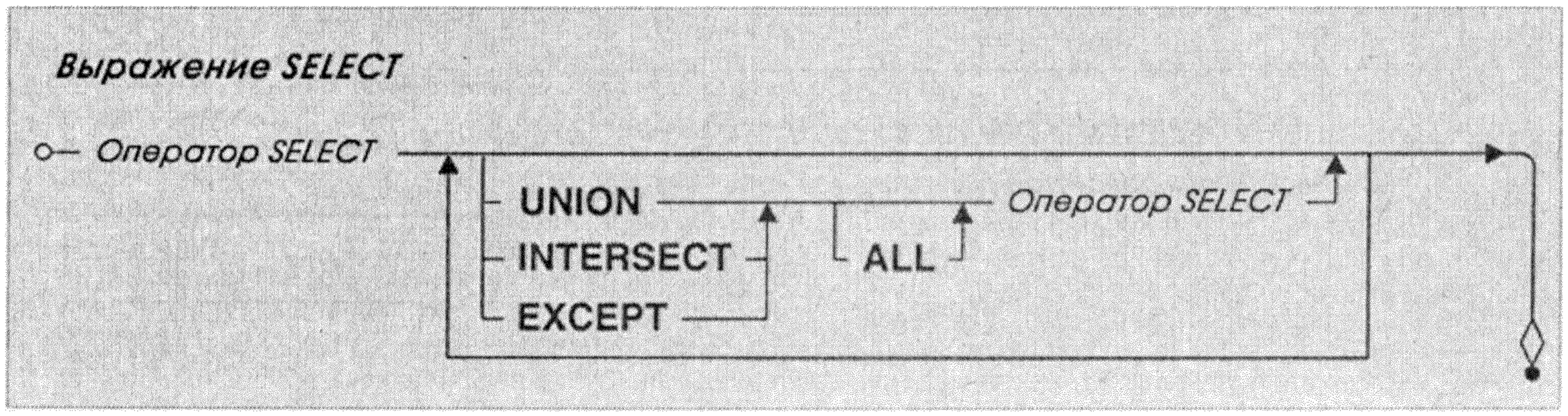
Приложения

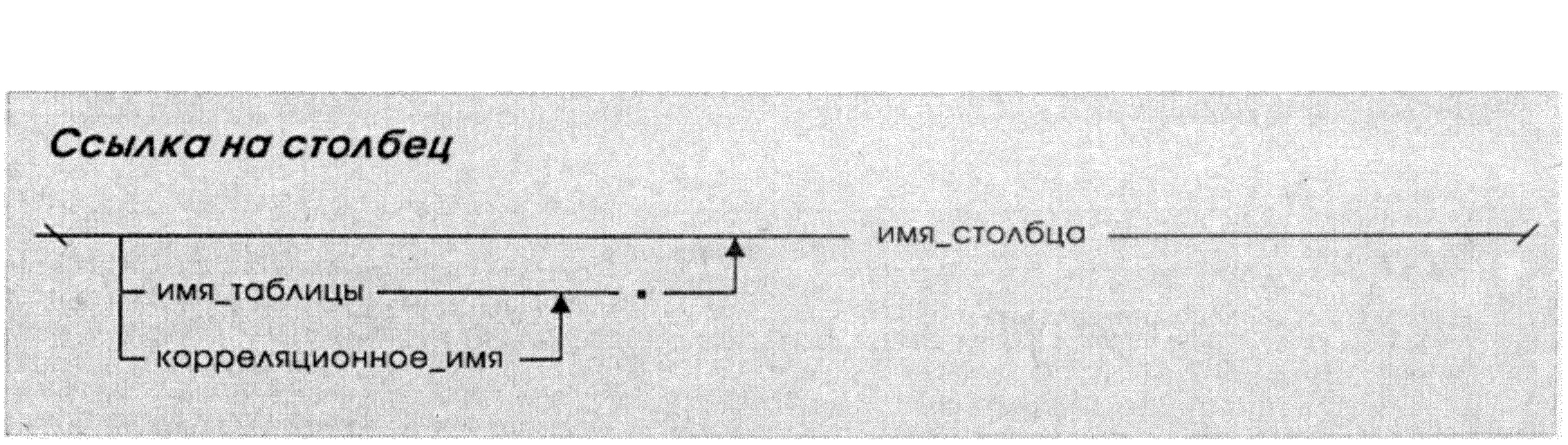
А

Диаграммы Стандарта SQL

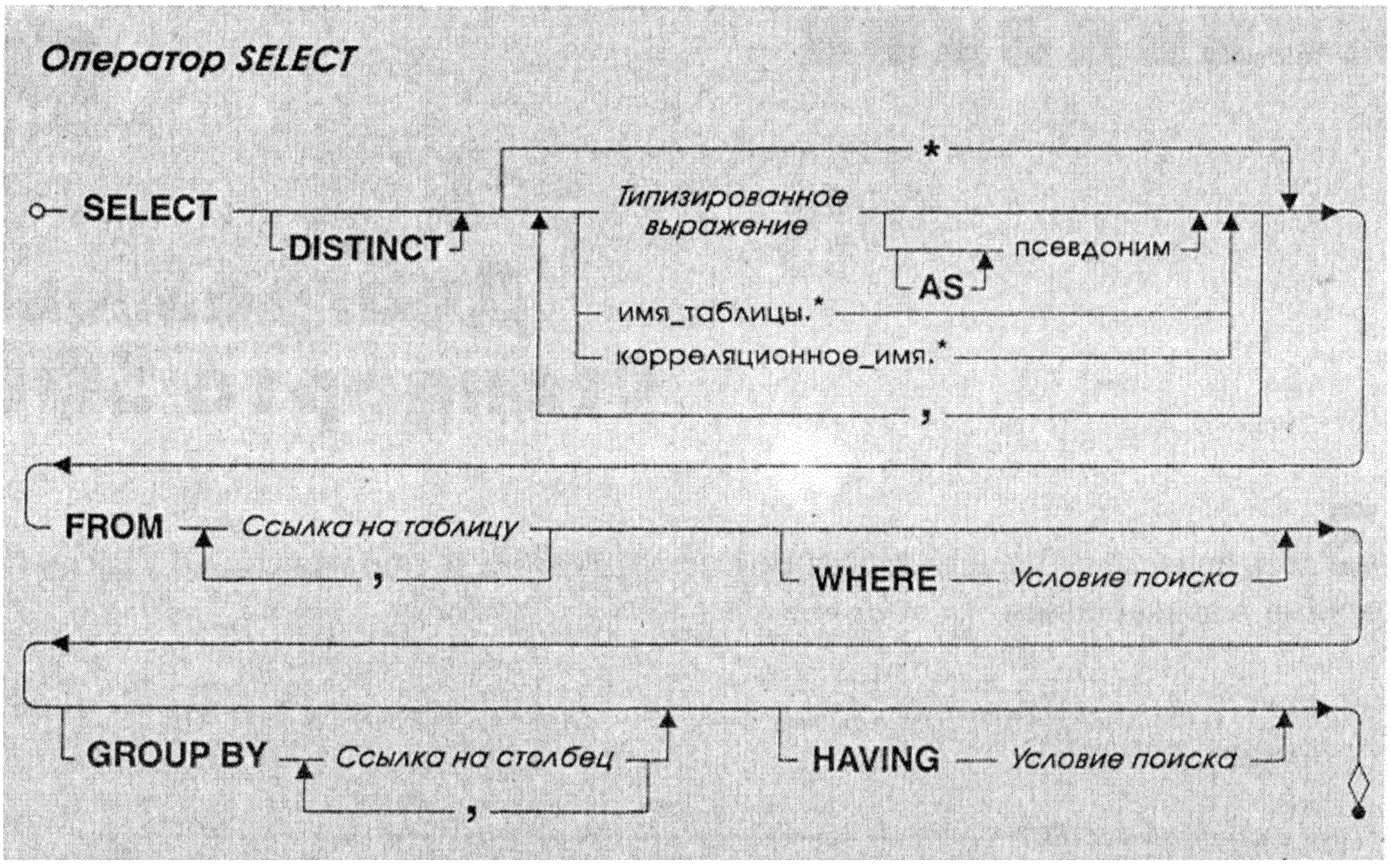
Здесь приведены полные диаграммы по грамматике и синтаксису SQL, рассмотренные в данной книге.







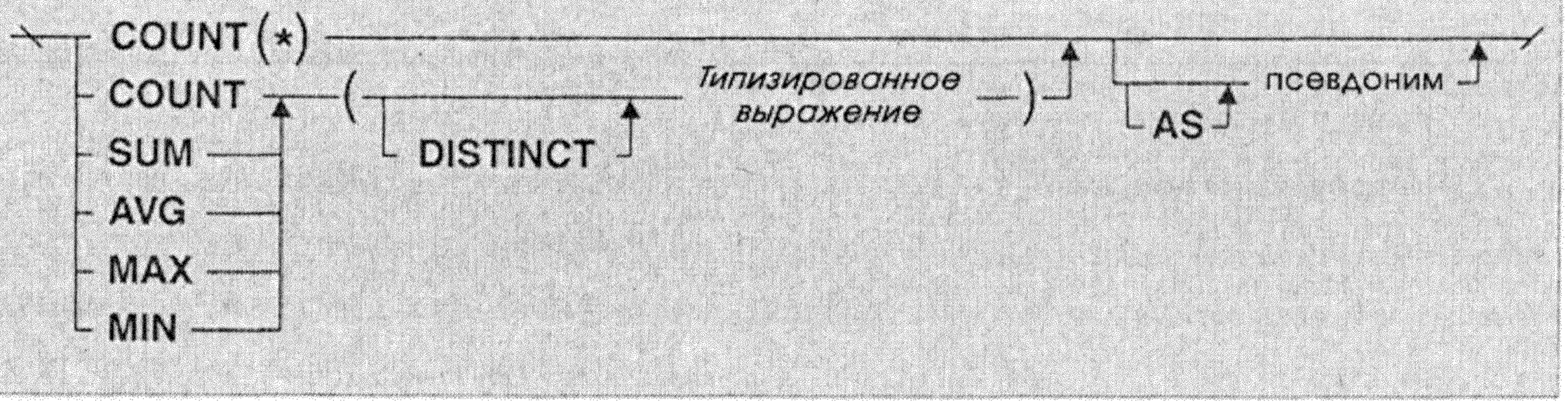
Оператор SELECT



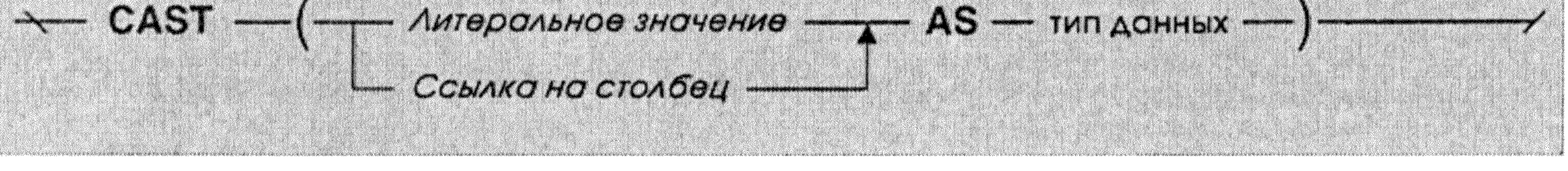
Типизированное выражение

Типы выражений	Допустимые операторы
Символьный	||
Числовой	+, -, *, /
Дата/Время	+, -

Агрегатные функции

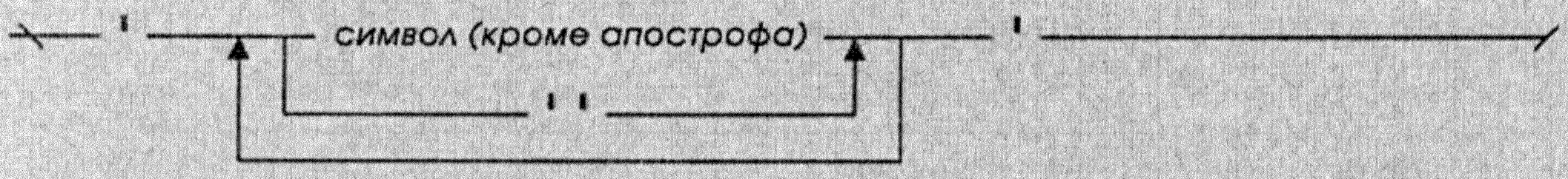


Функция CAST

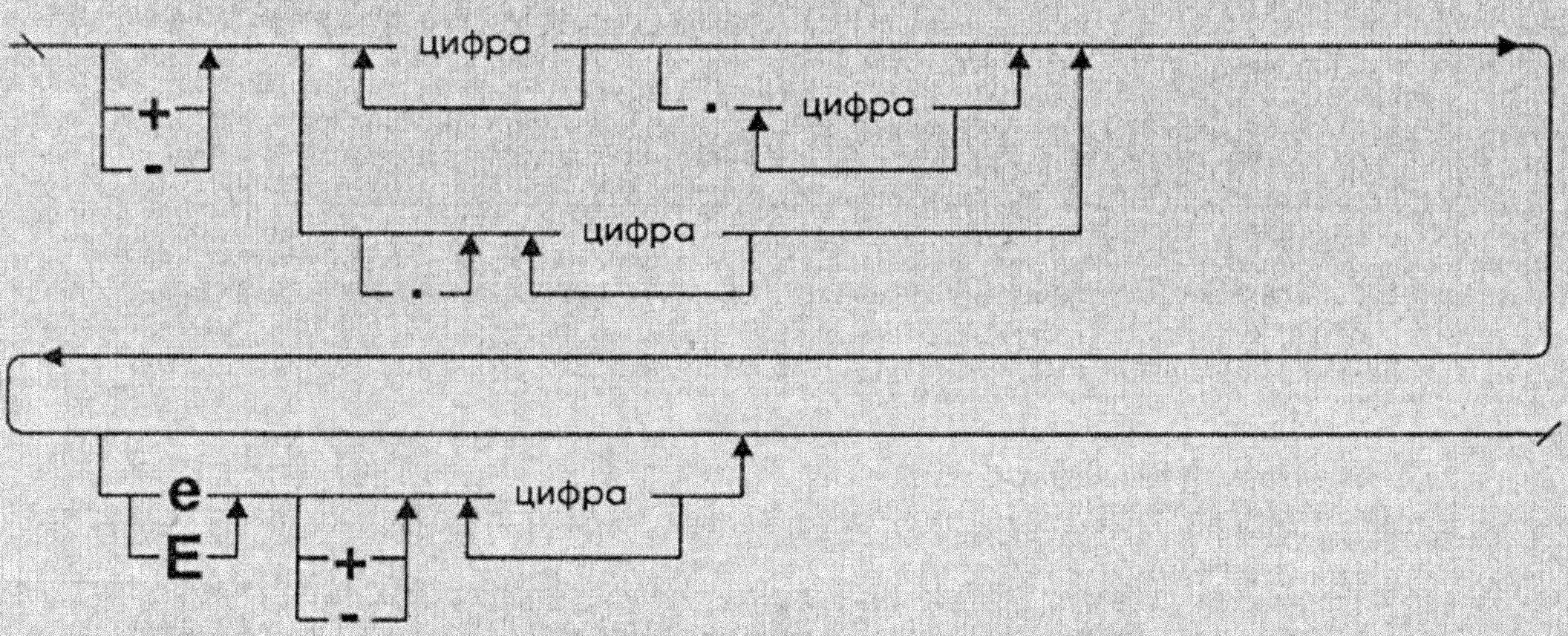


Значение литерала

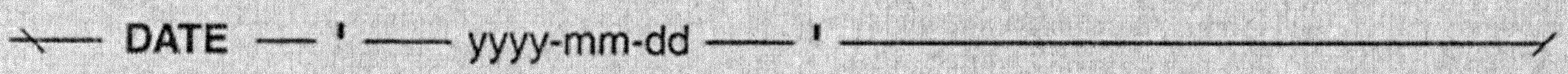
Строка символов



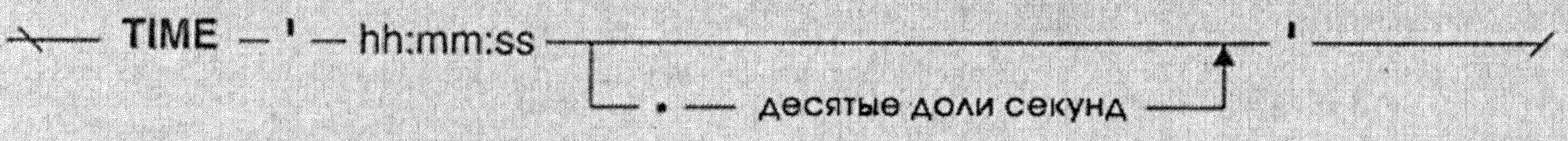
Цифровой

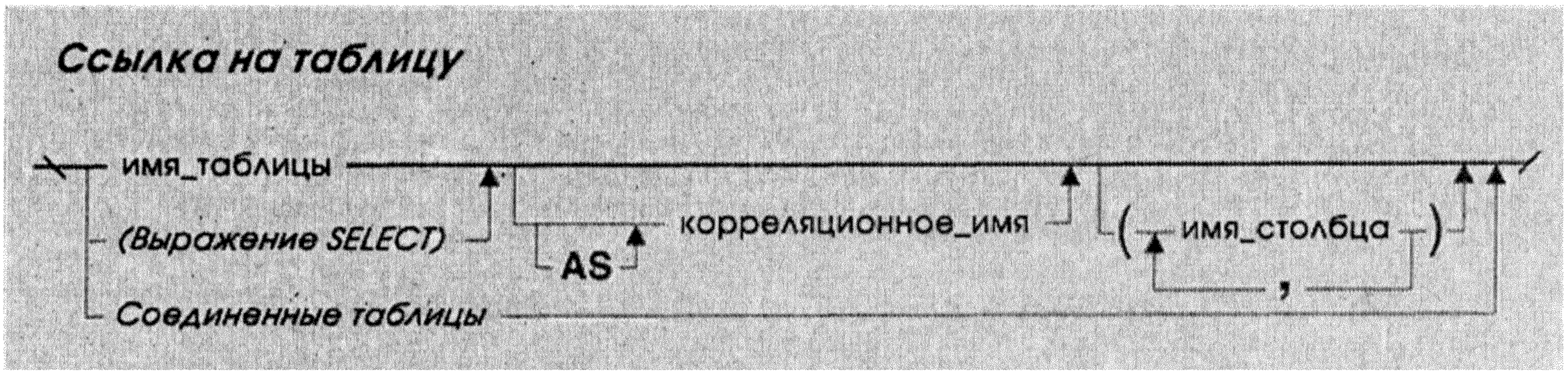


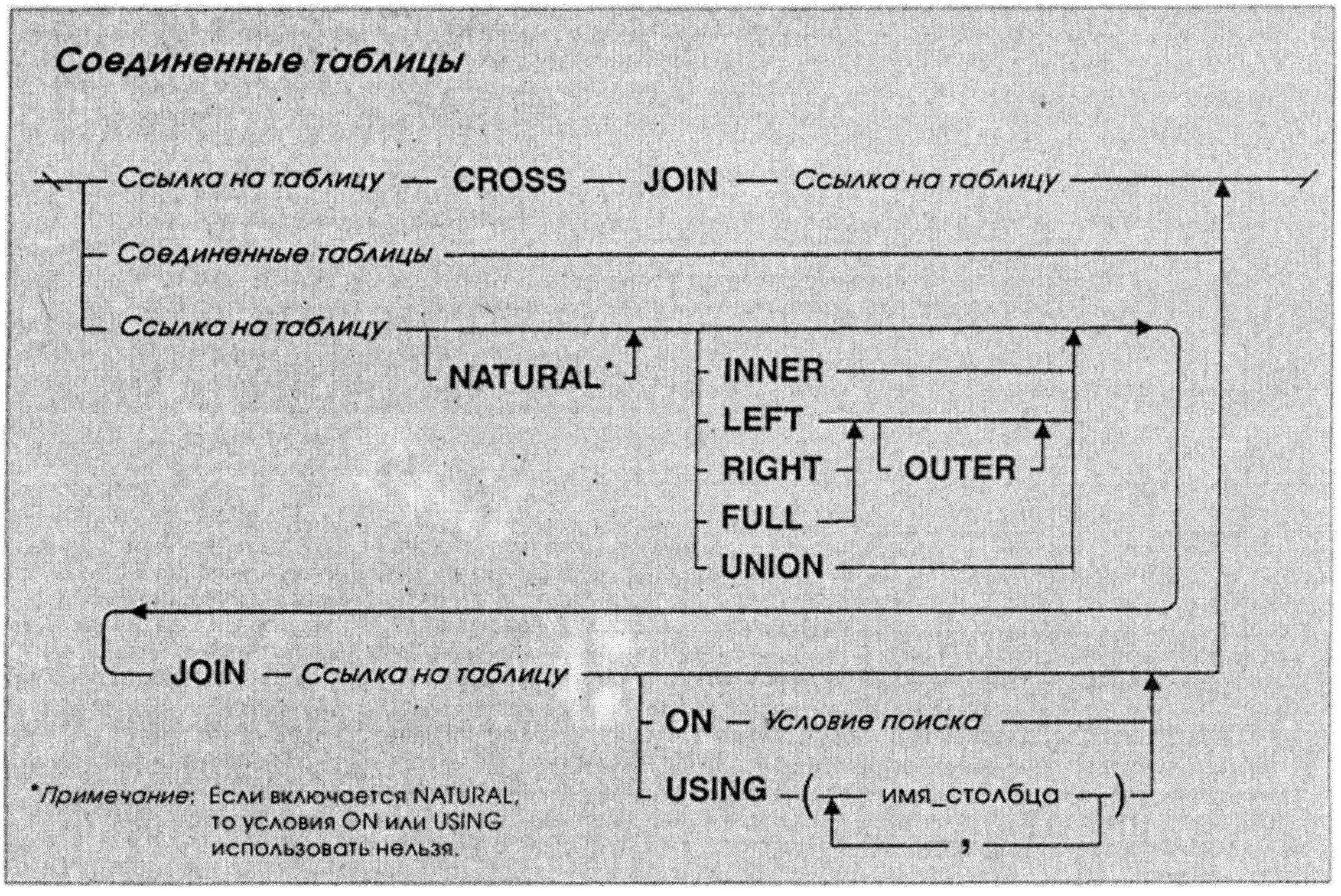
Дата

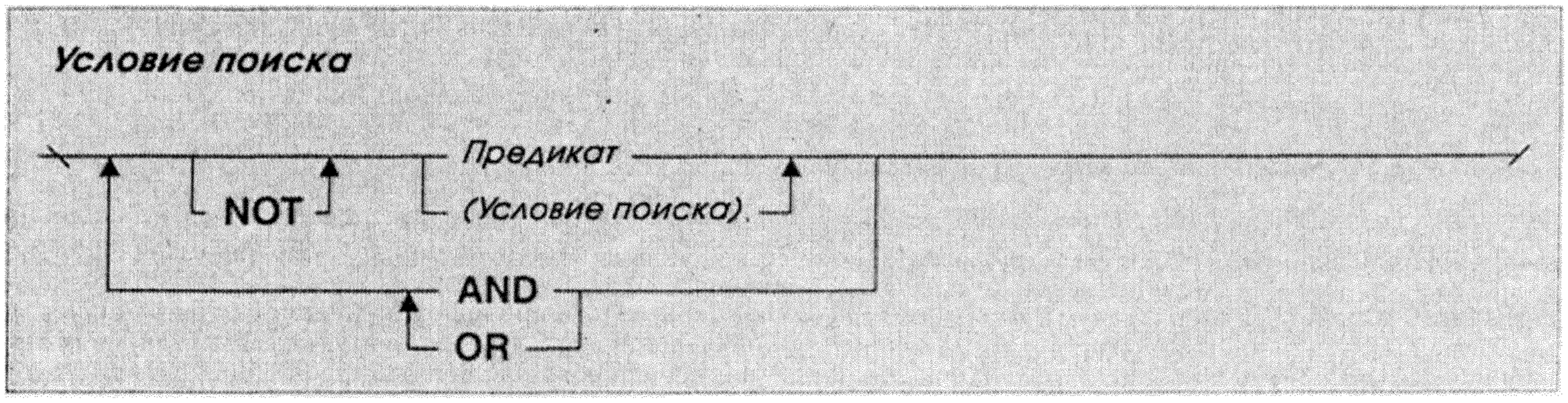


Время

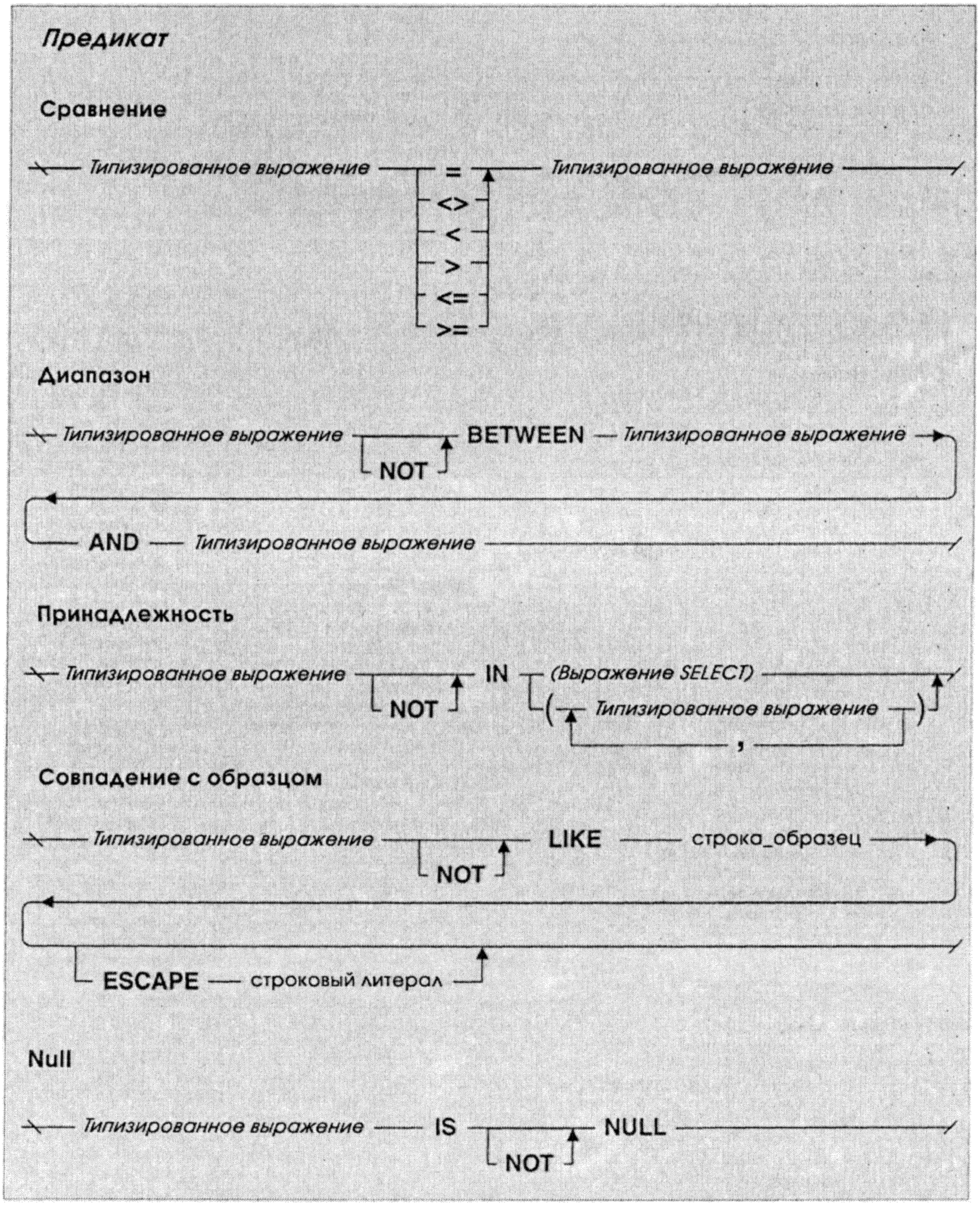


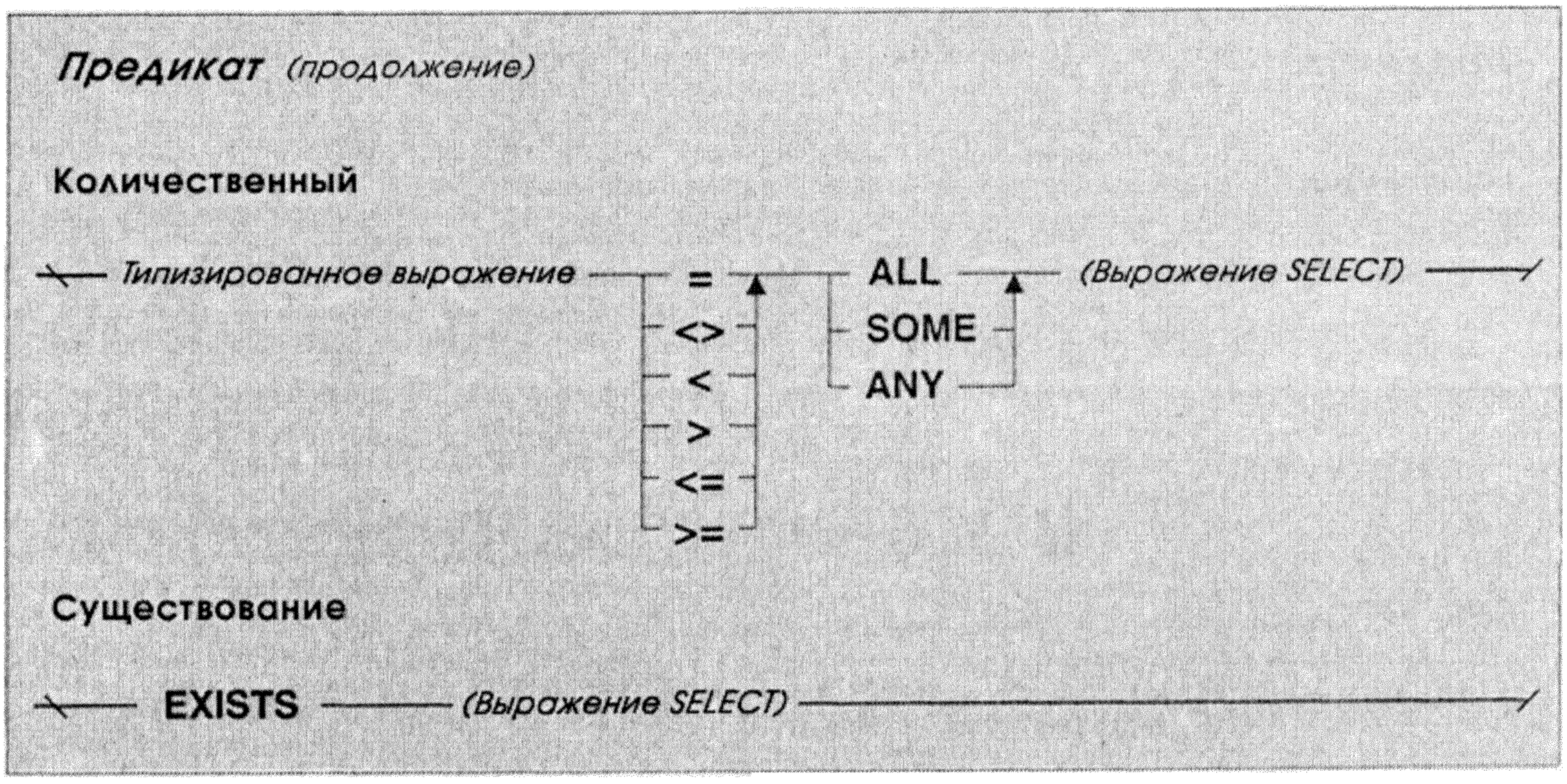

Ссылка на таблицу
имя_таблицы
(Выражение SELECT)
AS
корреляционное_имя
(
имя_столбца
,
)
Соединенные таблицы


Соединенные таблицы
Ссылка на таблицу
CROSS
JOIN
Ссылка на таблицу
Соединенные таблицы
Ссылка на таблицу
NATURAL*
INNER
LEFT
RIGHT
FULL
OUTER
UNION
JOIN
Ссылка на таблицу
ON
Условие поиска
USING
(
имя_столбца
,
)
*Примечание: Если включается NATURAL, то условия ON или USING использовать нельзя.


Условие поиска
Предикат
NOT
(Условие поиска)
AND
OR

Предикат

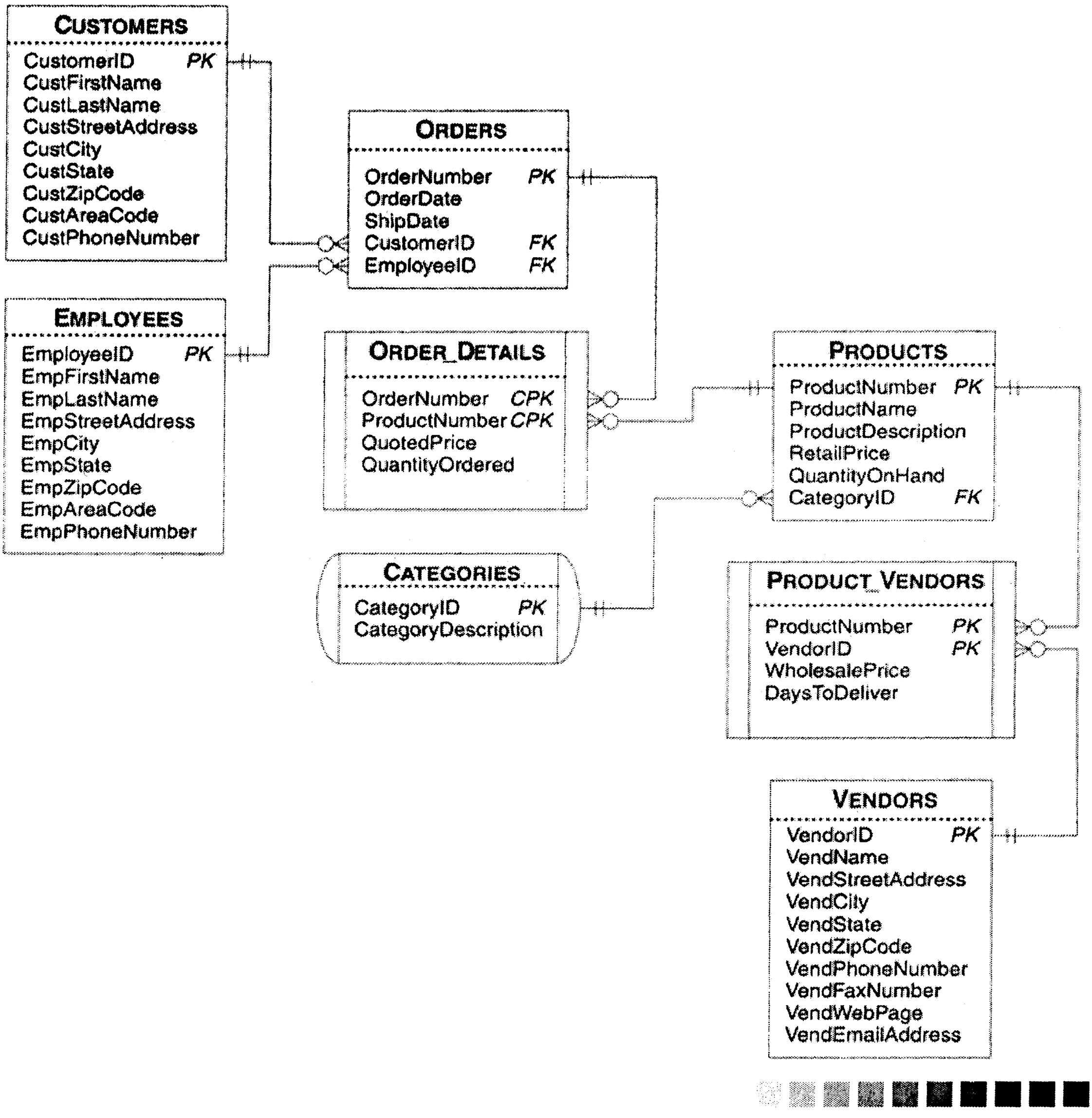

Сравнение
Типизированное выражение
=
<>
<
>
<=
>=
Типизированное выражение
Диапазон
Типизированное выражение
NOT
BETWEEN
Типизированное выражение
AND
Типизированное выражение
Принадлежность
Типизированное выражение
NOT
IN
(Выражение SELECT)
(
Типизированное выражение
,
)
Совпадение с образцом
Типизированное выражение
NOT
LIKE
строка_образец
ESCAPE
строковый литерал
Null
Типизированное выражение
IS
NOT
NULL


Предикат (продолжение)
Количественный
Типизированное выражение
=
<>
<
>
<=
>=
ALL
SOME
ANY
(Выражение SELECT)
Существование
EXISTS
(Выражение SELECT)

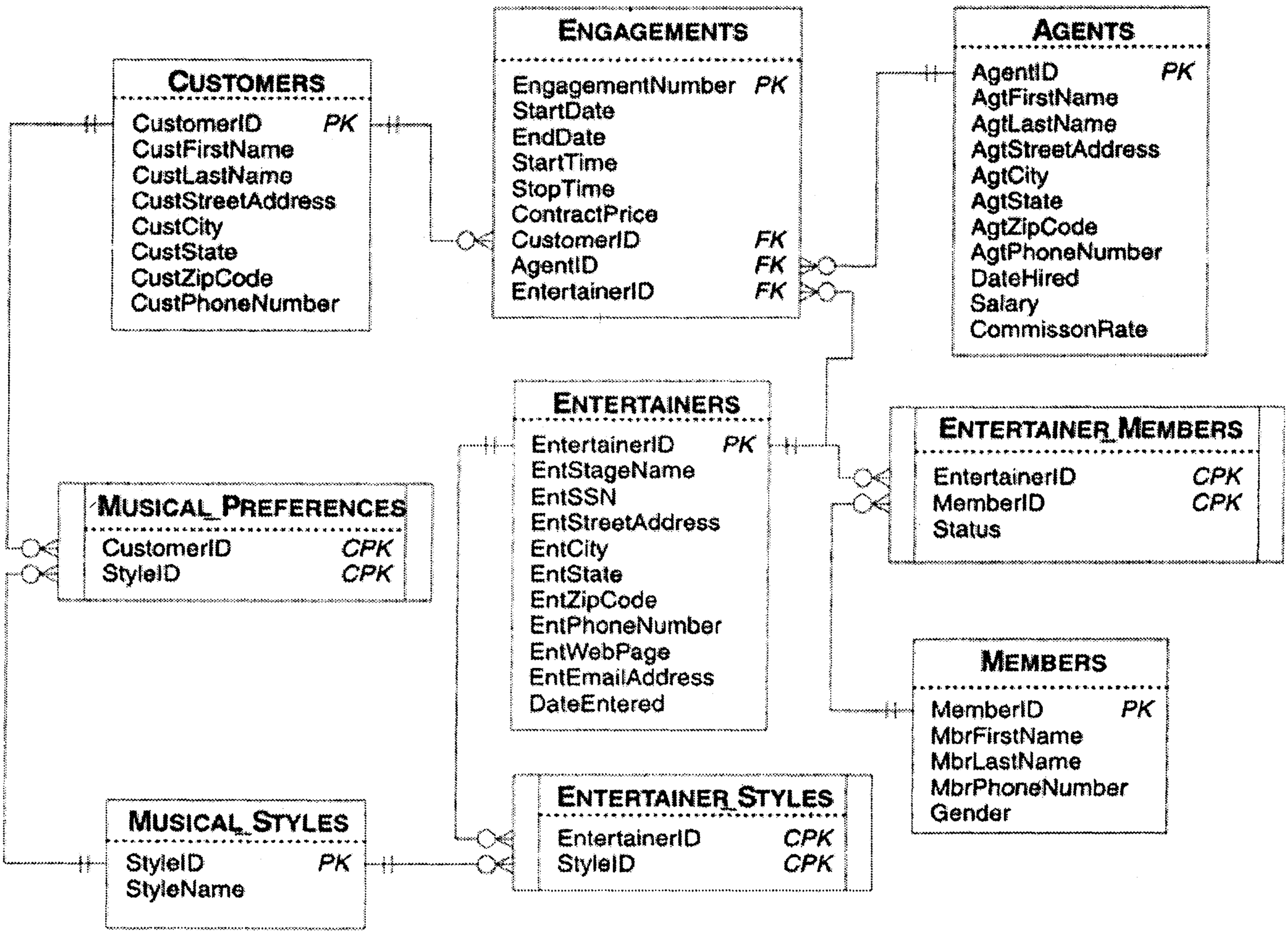
В

Структуры баз данных, использованных в качестве примеров

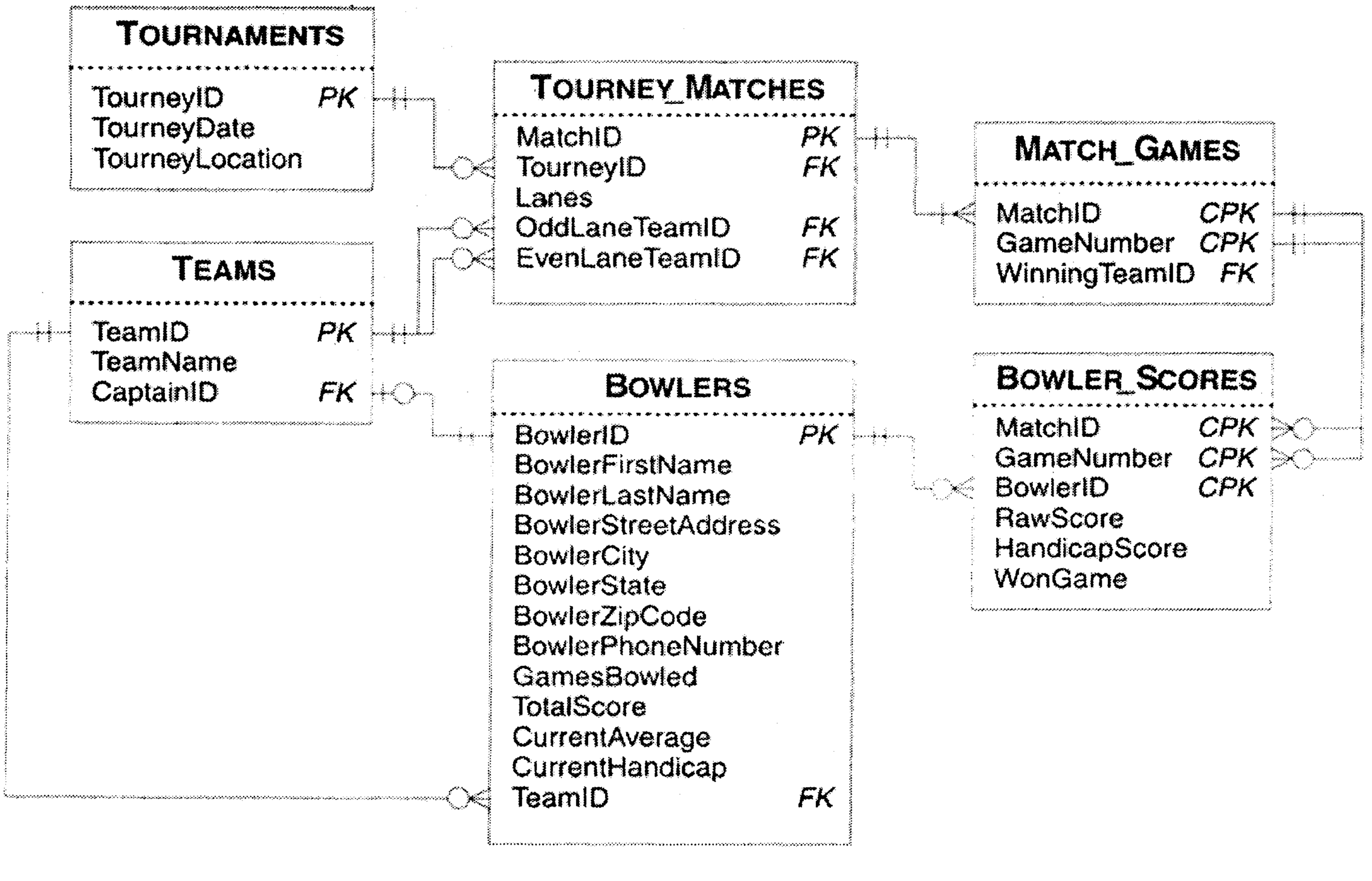
База данных заказов на закупку



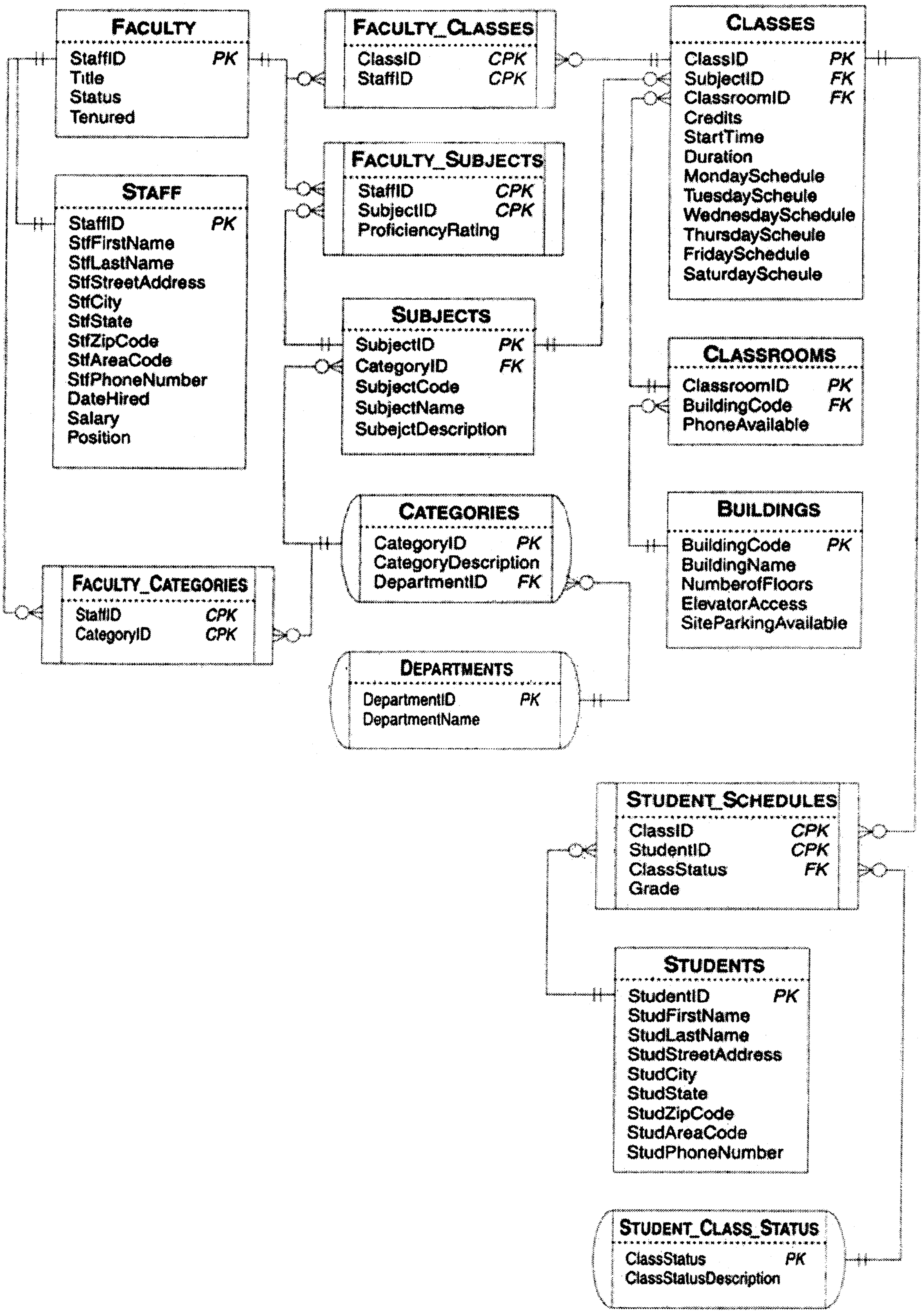
База данных агентства эстрадных мероприятий



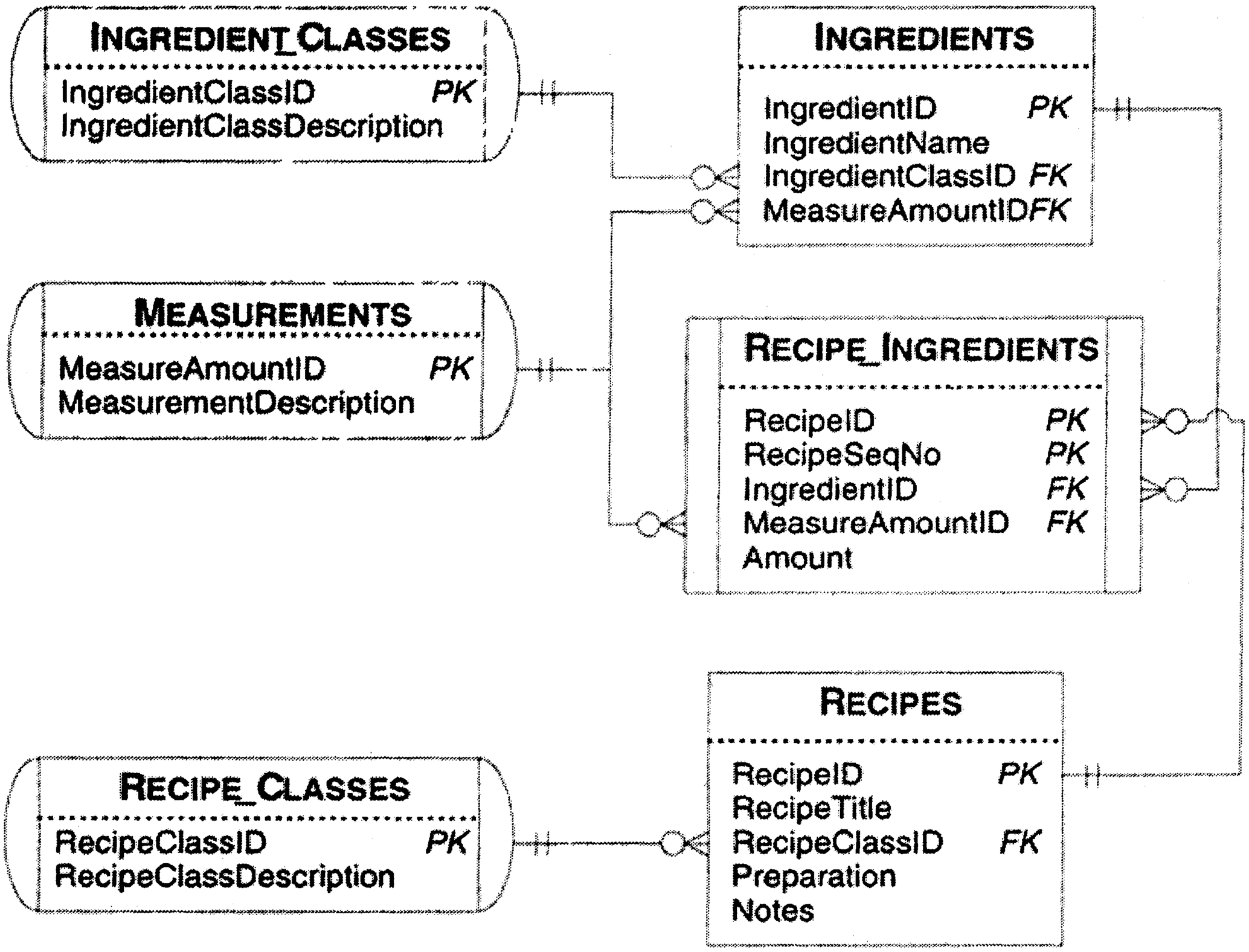
База данных лиги игры в боулинг



База данных расписания занятий



База данных рецептов



C

Литература, рекомендуемая для чтения

Ниже приводится список книг, которые рекомендуется прочитать тем, кто хочет больше узнать о проектировании баз данных или расширить свои знания по SQL. Часть этих книг достаточно сложна для восприятия, и по своему характеру они более специальные. Кроме того, некоторые авторы предполагают, что читатели имеют достаточно глубокие знания в области компьютеров, баз данных и программирования.

Книги по базам данных

Date, C. J. *An Introduction to Database Systems (7th Edition)*. Reading, MA: Addison-Wesley, 1999.

Connolly Thomas, Carolyn Begg, and Anne Strachan. *Database Systems—A Practical Approach to Design, Implementation, and Management*. Essex, England: Addison-Wesley, 1995.

Hernandez, Michael J. *Database Design for Mere Mortals*. Reading, MA: Addison-Wesley, 1997.

Книги по SQL

Bowman, Judith S., Sandra L. Emerson, and Marcy Darnovsky. *The Practical SQL Handbook (3rd Edition)*. Reading, MA: Addison-Wesley, 1996.

Celko, Joe. *Instant SQL Programming*. Chicago, IL: Wrox Press Ltd., 1995.

Celko, Joe. Joe Celko's *SQL for Smarties: Advanced SQL Programming (Second Edition)*. San Francisco, CA: Morgan Kaufmann Publishers, 1999.

Date, C. J., and Hugh Darwen. *A Guide to the SQL Standard (4th Edition)*. Reading, MA: Addison-Wesley, 1997.

Groff, James R., and Paul N. Weinberg. *LAN Times Guide to SQL*. Berkeley, CA: Osborne McGraw-Hill, 1994.

Gruber, Martin. *SQL Instant Reference*. Alameda, CA: Sybex Inc., 1993. (Русский перевод: Мартин Грабер. *SQL*. М., "Лори", 2001)

Melton, Jim, and Alan R. Simon. *Understanding the New SOL: A Complete Guide*. San Francisco, CA: Morgan Kaufmann Publishers, 1993.

SQL Queries for Mere Mortals
A Hands-On Guide to Data Manipulation in SQL
Michael J. Hernandez, John L. Viescas
Copyright © 2000
All rights reserved

SQL-запросы для простых смертных
Практическое руководство по манипулированию данными в SQL
Майкл Дж. Хернандес, Джон Л. Вьескас

Переводчик А. Головко
Научный редактор А. Киселева
Корректор Е. Пресс
Верстка Т. Кирпичевой

Copyright © 2000 by Michael J. Hernandez and John L. Viescas

Person Education Corporate Sales Division
One Lake Street, Upper Saddle River, NJ 07458, (800) 382-3419

Library of Congress Cataloging-in-Publication Data
ISBN 0-201-43336-2

© Издательство "Лори", 2003

Изд. № : OAI (03)
ЛР № : 070612 30.09.97 г.
ISBN 5-85582-178-1

Подписано в печать 10.01.2003 Формат 70 × 100/16
Бумага офсет № 1 Гарнитура Литературная Печать офсетная
Печ. л. 30 Тираж 3200 Заказ № 747
Цена договорная

Изд-во "Лори". Москва 123100 Шмитовский пр., д. 13/6, стр. 1 (пом. ТАРП ЦАО)
Телефон для оптовых покупателей: (095)256-02-83
Размещение рекламы: (95)259-01-62

WWW.LORY–PRESS.RU

Отпечатано в ООО "Типография ИПО профсоюзов Профиздат".
109044, Москва, ул. Крутицкий вал, д. 18