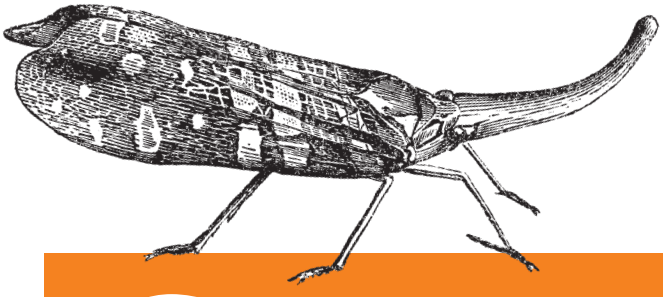
Эффективная работа с Oracle SQL

Охватывает Oracle9i



Секреты Oracle SQL

СИМВОЛ®

O'REILLY®

Санжей Мишра и Алан Бьюли

Mastering Oracle SQL

Sanjay Mishra & Alan Beaulieu

O'REILLY®

Секреты Oracle SQL

Санжей Мишра и Алан Бьюли

Санкт-Петербург — Москва
2003

Санжей Мишра, Алан Бьюли

Секреты Oracle SQL

Перевод П. Шера

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*А. Королев*
Редактор	*А. Петухов*
Корректор	*С. Беляева*
Верстка	*Н. Гриценко*

Мишра С., Бьюли А.
Секреты Oracle SQL. – Пер. с англ. – СПб: Символ-Плюс, 2003. – 368 с., ил.
ISBN 5-93286-047-2

Большинство книг по SQL не выходит за рамки обсуждения синтаксиса и азов применения. Книга «Секреты Oracle SQL» – одно из исключений, где авторы, используя Oracle8*i* и Oracle9*i*, на большом количестве примеров показывают, как творчески применять мощные и гибкие средства SQL для быстрого создания эффективных и удобных для сопровождения запросов в среде Oracle. Если вы Java-программист, администратор базы данных или программист на PL/SQL, эта книга для вас. Вы сможете повысить производительность своего труда и приобрести уверенность в правильности написания ваших SQL-запросов.

В этой книге описываются наиболее важные и полезные свойства Oracle SQL и пути их применения для решения конкретных задач. Вы найдете множество оригинальных приемов, которые можно использовать для улучшения ваших собственных приложений. Вы научитесь: применять ANSI-совместимый синтаксис объединения; работать с новыми типами для времени и даты; максимально использовать такие конструкции SQL, как подзапросы, слияния, группы и объединения; обрабатывать иерархические данные; использовать инструкции DECODE и CASE для реализации условной логики в запросах SQL; строить запросы, работающие с разделами, объектами и коллекциями; применять аналитические функции SQL для создания ранжирующих и оконных запросов; пользоваться сложными группирующими функциями.

ISBN 5-93286-047-2
ISBN 0-596-00129-0 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 16.06.2003. Формат 70x100 1/16 . Печать офсетная.
Объем 23 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с диапозитивов в Академической типографии «Наука» РАН
199034, Санкт-Петербург, 9 линия, 12.

Посвящаю эту книгу моему отцу.
Хотел бы, чтобы он дожил до ее появления.
Санжей Мишра

Моим дочерям, Мишель и Николь.
Алан Бьюли

Оглавление

Предисловие

SQL (Structured Query Language, язык структурированных запросов) – это язык, применяемый для доступа к реляционной базе данных (БД). SQL состоит из набора операторов, позволяющих сохранять данные в БД и извлекать их из БД. Популярность языка неуклонно росла с того самого момента, когда на свет появилась первая реляционная база данных. Предлагались и другие языки запросов, но на данный момент SQL признан стандартным языком почти для всех реализаций реляционных баз данных, включая Oracle.

SQL отличается от других языков программирования тем, что он непроцедурный. В отличие от программ на других языках, в которых необходимо задавать последовательность действий для выполнения, программа на SQL (более точно называемая SQL-оператором) всего лишь описывает желаемый результат. Система управления базами данных сама определяет, каким образом следует обработать данные для получения желаемого результата. Непроцедурная природа SQL облегчает доступ к данным из прикладных программ.

Если вы пользуетесь базой данных Oracle, то SQL – это интерфейс, который вы применяете для доступа к данным, хранящимся в базе. SQL позволяет создавать структуры базы данных, такие как таблицы (для хранения данных), представления и индексы. SQL позволяет заносить данные в БД и извлекать хранимые данные в нужном формате (например, отсортированными). Наконец, SQL позволяет модифицировать, удалять и всячески манипулировать хранимыми данными. SQL – это ключ ко всем вашим действиям с базой данных. Совершенное владение языком SQL является одним из наиболее важных требований к разработчику или администратору баз данных.

Зачем мы написали эту книгу

Наши собственные впечатления от изучения и применения базы данных Oracle и реализации SQL от Oracle побудили нас написать эту книгу. Документация Oracle по SQL состоит из справочного руководства, которое не углубляется в тонкости практического применения различ-

ных особенностей SQL, поддерживаемых Oracle, и не содержит сложных примеров из реальной жизни.

Обратившись за помощью в магазин компьютерной литературы, мы обнаружили, что в действительности существуют лишь два типа книг по SQL. Большинство из них справочного типа; они описывают функции и синтаксис языка, но ничего не говорят о том, как применить эти знания к реальным задачам. Другой тип книг, весьма малочисленный, обсуждает применение SQL в сухом теоретическом стиле, не используя какую-либо реализацию конкретного производителя. А поскольку каждый производитель баз данных реализует собственный вариант SQL, мы считаем книги по «стандартному» SQL не очень-то полезными.

Мы решили написать практическое руководство, сконцентрировавшись непосредственно на Oracle-версии SQL. Oracle лидирует на рынке СУБД, к тому же, это та база данных, с помощью которой мы приобрели свои навыки работы с SQL. В данной книге мы не только представляем наиболее важные и полезные свойства Oracle SQL, но и описываем пути их применения для решения конкретных задач.

Цели этой книги

Единственная и важнейшая цель этой книги – помочь вам максимально использовать всю мощь Oracle SQL. Вы научитесь:

- Понимать особенности и возможности языка SQL в реализации Oracle.
- Использовать такие возможности SQL, как внешние объединения, связанные подзапросы, иерархические запросы, группирующие операции, аналитические запросы и т. д.
- Использовать инструкции `DECODE` и `CASE` для внесения условной логики в SQL-запросы.
- Строить SQL-запросы, работающие с разделами, объектами и коллекциями, такими как вложенные таблицы и массивы переменной длины.
- Использовать новые возможности SQL, представленные в Oracle9*i*, такие как новые особенности работы с датой и временем, ANSI-совместимые объединения и новые группирующие и аналитические функции.
- Использовать лучшие приемы для написания эффективных и удобных для сопровождения SQL-запросов.

Для кого наша книга?

Эта книга предназначена для разработчиков и администраторов баз данных Oracle. Новичок ли вы в мире баз данных или опытный специалист – если вы используете SQL для доступа к базе данных Oracle, эта

книга для вас. Вне зависимости от того, используете ли вы простые запросы для доступа к данным или же встраиваете их в PL/SQL- или Java-программы, SQL является основой для всех задач доступа к данным в вашем приложении. Овладев мощными и гибкими средствами SQL, вы повысите производительность своего труда, будете успевать сделать больше за меньшее время и при этом будете уверены в правильности написанных вами SQL-запросов.

Платформа и версия

В этой книге мы использовали Oracle8*i* (версии 8.1.6 и 8.1.7) и Oracle9*i* (версия 9.0.1). Мы описали многие новые важные возможности Oracle9*i* SQL, включая синтаксис объединений, соответствующий стандарту ANSI, новые типы для времени и даты, различные аналитические функции. Большинство же понятий, правил синтаксиса и примеров относится и к более ранним версиям Oracle. Новые возможности Oracle9*i* мы отмечаем специально.

Структура этой книги

Книга состоит из 14 глав:

- Глава 1 «Введение в SQL» знакомит читателя с языком SQL и дает краткое представление о его истории. Эта глава предназначена в основном для тех читателей, которые имеют небольшой опыт или вообще не имеют никакого опыта работы с SQL. В ней вы найдете простые примеры основных операторов SQL (`SELECT`, `INSERT`, `UPDATE` и `DELETE`) и стандартных возможностей SQL.
- В главе 2 «Инструкция WHERE» описываются способы фильтрации данных в операторах SQL. Вы научитесь ограничивать результаты запроса теми строками, которые хотите видеть, и распространять результаты выполнения оператора на те строки, которые хотите изменить.
- В главе 3 «Объединения» описываются конструкции, используемые для доступа к данным из нескольких связанных таблиц. В этой главе обсуждаются такие важные понятия, как внутренние и внешние объединения. Также описывается новый, соответствующий стандарту ANSI синтаксис объединений, представленный в Oracle9*i*.
- В главе 4 «Групповые операции» показывается, как формировать суммарную информацию, такую как итоговую и подытоговую суммы ваших данных. Вы узнаете, как определять группы строк и как применять различные обобщающие функции для обработки данных из этих групп.
- В главе 5 «Подзапросы» рассказывается, как использовать связанные и несвязанные подзапросы и встроенные представления для ре-

шения сложных проблем, которые без использования этих средств потребовали бы процедурного кода и нескольких запросов к БД.

- Глава 6 «Обработка дат и времени» рассказывает о том, как обращаться с датой и временем в базе данных Oracle. Вы познакомитесь с различными приемами, используемыми при запрашивании данных о времени. Также вы узнаете о многих новых типах данных для представления даты и времени, введенных в Oracle9*i*.
- В главе 7 «Операции над множествами» показывается, как использовать инструкции `UNION`, `INTERSECT` и `MINUS` для комбинирования результатов двух или более независимых запросов.
- В главе 8 «Иерархические запросы» обсуждается, как хранить и извлекать иерархическую информацию (такую как структура организации) из реляционной таблицы. Oracle обладает рядом возможностей, облегчающих работу с иерархическими данными.
- Глава 9 «DECODE и CASE» рассказывает о двух очень мощных, но в то же время простых возможностях Oracle SQL, позволяющих имитировать условную логику (без них SQL был бы просто декларативным языком). CASE, стандартная конструкция ANSI, была впервые представлена в Oracle8*i* и улучшена в Oracle9*i*.
- В главе 10 «Разделы, объекты и коллекции» обсуждаются вопросы, связанные с доступом к разделам и коллекциям посредством SQL. Вы научитесь строить операторы SQL, которые работают с отдельными разделами и подразделами. Также вы узнаете, как запрашивать объектные данные, вложенные таблицы и массивы переменной длины.
- Глава 11 «PL/SQL» посвящена интеграции SQL и PL/SQL. Вы узнаете, как вызывать хранимые процедуры и функции PL/SQL из SQL-запросов и как писать эффективные операторы SQL в PL/SQL-программах.
- В главе 12 «Сложные групповые операции» рассматриваются сложные групповые операции, использующиеся в основном в системах принятия решений. Мы покажем, как использовать возможности Oracle, такие как `ROLLUP`, `CUBE` и `GROUPING SETS`, для успешного формирования различных уровней суммарной информации, необходимой в системах принятия решений. Кроме того, обсуждаются новые групповые особенности Oracle9*i*, которые делают возможными составные и сцепленные группировки, а также новые функции `GROUP_ID` и `GROUPING_ID`.
- Глава 13 «Аналитический SQL» знакомит вас с аналитическими запросами и новыми аналитическими функциями. Вы узнаете, как использовать ранжирующие, оконные функции и функции составления отчетов, чтобы сформировать информацию для системы принятия решений. Эта глава также охватывает новые аналитические возможности, представленные в Oracle9*i*.

- Глава 14 «Советы умудренных опытом» рассказывает о правилах, которых следует придерживаться, чтобы писать эффективные и удобные в обслуживании запросы. Вы узнаете, какие конструкции SQL наиболее эффективны в конкретной ситуации. Например, мы объясняем, когда лучше использовать `WHERE` вместо `HAVING` для ограничения результатов запроса. Также обсуждается влияние на производительность использования связанных переменных по сравнению с полностью определенным SQL.

Соглашения, используемые в этой книге

В оформлении книги приняты следующие соглашения:

Курсив

Используется для имен файлов и каталогов, названий таблиц и полей, а также для URL. Кроме того, курсивом выделяются специальные термины при первом их появлении в тексте.

`Моноширинный шрифт`

Применяется в примерах, а также для отображения содержимого файлов и выводимых данных.

`Моноширинный курсив`

Применяется в описании синтаксиса для данных, которые должны быть заменены пользователем на реальные значения.

`Моноширинный полужирный шрифт`

Используется в примерах для информации, которую вводит пользователь. Также применяется для выделения участков кода, на которые следует обратить внимание.

`Моноширинный полужирный курсив`

Применяется в коде примеров для выделения конструкций SQL или результатов, которые обсуждаются в тексте.

ПРОПИСНЫЕ БУКВЫ

Прописными буквами в описании синтаксиса выделяются ключевые слова.

строчные буквы

Строчными буквами в описании синтаксиса выделяются определяемые пользователем элементы, например переменные.

[ ]

В описании синтаксиса в квадратные скобки заключены необязательные элементы.

{ }

В описании синтаксиса в фигурные скобки заключается набор элементов, из которых должен быть выбран один.

|

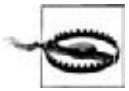
В описании синтаксиса вертикальная черта разделяет элементы списка, помещаемого в фигурные скобки, например: {TRUE | FALSE}.

...

В описании синтаксиса многоточие указывает на повторение элемента.

Особое внимание уделяйте замечаниям, выделенным в тексте следующим образом:

Это совет, предложение или примечание. Например, мы используем примечания, для того чтобы отметить новые возможности Oracle9*i*.



Это предупреждение. Например, подобным образом отмечаются инструкции SQL, неосторожное применение которых может привести к непредвиденным последствиям.

Комментарии и вопросы

Мы протестировали и проверили информацию в этой книге настолько хорошо, насколько это было возможно, но если вы обнаружите, что некоторые свойства изменились или мы допустили ошибки, сообщите, пожалуйста, об этом по адресу:

O'Reilly & Associates
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (FAX)

Также вы можете послать сообщение по электронной почте. При желании попасть в список рассылки или получить каталог, напишите по адресу:

info@oreilly.com

Можно задать технические вопросы или прокомментировать книгу, написав по адресу:

bookquestions@oreilly.com

На веб-сайте книги можно найти список опечаток (найденные ошибки и исправления доступны для публичного просмотра):

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

Для получения дополнительной информации об этой и других книгах посетите сайт O'Reilly:

http://www.oreilly.com

Благодарности

Мы в долгу перед множеством людей, внесших вклад в подготовку и издание этой книги. Мы глубоко обязаны Джонатану Геннику (Jonathan Gennic), редактору этой книги. Его видение книги, внимательное отношение к деталям и исключительные редакторские навыки – вот причины, по которым эта книга сегодня перед вами.

Искренняя благодарность нашим техническим рецензентам: Дайане Лоренц (Diana Lorentz), Джеффу Коксу (Jeff Cox), Стефану Андерту (Stephan Andert), Ричу Уайту (Rich White), Петеру Линсли (Peter Linsley) и Крису Ли (Chris Lee), которые посвятили массу своего бесценного времени чтению и комментированию черновика книги. Их вклад сделал ее более точной и ценной, облегчил ее чтение.

Эта книга не смогла бы появиться без огромной работы и поддержки высококвалифицированных сотрудников O'Reilly & Associates, включая дизайнеров обложки Элли Волькхаусен (Ellie Volckhausen) и Эмму Колби (Emma Colby), дизайнера книги Дэвида Футато (David Futato), Нила Уолса (Neil Walls), который конвертировал файлы, редактора и выпускающего редактора Коллина Гормана (Coleen Gorman), иллюстраторов Роба Романо (Rob Romano) и Джессамин Рид (Jessamyn Read), а также Шерил Авруч (Sheryl Avruch) и Анну Ширмер (Ann Schirmer), контролировавших качество, и Тома Динса (Tom Dinse), составителя индекса. Также спасибо Тиму O'Рейлли (Tim O'Reilly) за просмотр книги и ценный отзыв.

От Санжея

Сердечная благодарность моему соавтору Алану за его великолепные технические навыки и совместную работу над книгой. Особая благодарность Джонатану Геннику не только за редактирование книги, но и за предоставленный мне удаленный доступ к его базе данных Oracle9*i*.

Мои приключения, связанные с Oracle, начались в проекте Tribology Workbench в Тата Стил, Джамшедпур (Индия). Искренняя благодарность моим коллегам по проекту Tribology Workbench за все эксперименты и исследования, сделанные в ходе изучения Oracle. Особая благодарность Сарошу Мунчержи (Sarosh Muncherji), заместителю руководителя группы, за то, что он пригласил меня в проект и погрузил в мир Oracle, поручив мне администрирование базы данных. С тех пор технология баз данных Oracle стала для меня образом жизни.

Искренняя благодарность моим коллегам в i2 Technologies за их поддержку.

Последняя в списке, но не последняя по значимости благодарность моей жене Сьюдипти за ее понимание и поддержку.

От Алана

Я хотел бы поблагодарить моего соавтора Санжея и редактора Джонатана Генника за то, что они разделили мои взгляды на эту книгу, и за их технический и редакторский героизм. Я никогда бы не дошел до финиша без вашей помощи и поддержки.

Больше всего я хотел бы поблагодарить мою жену Нэнси за ее поддержку и терпение и моих дочерей, Мишель и Николь, за их любовь и понимание.

1

Введение в SQL

Во вступительной главе мы поговорим об истоках языка SQL, обсудим его наиболее полезные возможности и определим простую базу данных, которая ляжет в основу большинства примеров книги.

Что такое SQL?

SQL (Structured Query Language, язык структурированных запросов) – это специальный язык, используемый для определения данных, доступа к данным и их обработки. SQL относится к *непроцедурным* (*nonprocedural*) языкам – он лишь описывает нужные компоненты (например, таблицы) и желаемые результаты, не указывая, как именно эти результаты должны быть получены. Каждая реализация SQL является надстройкой над *процессором базы данных* (*database engine*), который интерпретирует операторы SQL и определяет порядок обращения к структурам БД для корректного и эффективного формирования желаемого результата.

Язык SQL состоит из двух специальных наборов команд. DDL (Data Definition Language, язык определения данных) – это подмножество SQL, используемое для определения и модификации различных структур данных, а DML (Data Manipulation Language, язык манипулирования данными) – это подмножество SQL, применяемое для получения и обработки данных, хранящихся в структурах, определенных ранее с помощью DDL. DDL состоит из большого количества команд, необходимых для создания таблиц, индексов, представлений и ограничений, а в DML входит всего четыре оператора:

INSERT

Добавляет данные в базу данных.

UPDATE

Изменяет данные в базе данных.

DELETE

Удаляет данные из базы данных.

SELECT

Извлекает данные из базы данных.

Некоторым кажется, что применение DDL является прерогативой администраторов базы данных, а операторы DML должны писать разработчики, но эти два языка не так-то просто разделить. Сложно организовать эффективный доступ к данным и их обработку, не понимая, какие структуры доступны и как они связаны. Также сложно проектировать соответствующие структуры, не зная, как они будут обрабатываться. Сказав это, сосредоточимся на DML. DDL в книге будет встречаться лишь тогда, когда это необходимо для иллюстрации применения DML. Причины особого внимания к DML таковы:

- DDL хорошо описан во многих книгах по проектированию и администрированию баз данных, а также в справочниках по SQL.
- Проблемы производительности обычно бывают вызваны неэффективными операторами DML.
- Хотя операторов всего четыре, DML – настолько большая тема, что ее хватило бы на целую серию книг.[1]

Но почему вообще кого-то должен интересовать SQL? Зачем в наш век Интернета и многоуровневых архитектур заботиться о доступе к данным? На самом деле эффективное хранение и извлечение информации сейчас важно как никогда ранее:

- Все больше компаний предлагают свои услуги через Интернет. В часы пик они вынуждены обслуживать тысячи параллельных запросов, и задержки означают прямую потерю прибыли. Для таких систем каждый оператор SQL должен быть тщательно продуман, чтобы обеспечивать требуемую производительность при увеличении объема данных.
- Сегодня есть возможность хранить гораздо больше данных, чем пять лет назад. Один дисковый массив вмещает десятки терабайт данных, и уже не за горами хранение сотен терабайт. Программное обеспечение, применяемое для загрузки и анализа данных в этих средах, должно использовать весь потенциал SQL, чтобы обрабатывать неизменно увеличивающийся объем данных за постоянные (или сокращающиеся) промежутки времени.

---

[1] Каждый, кто работает с SQL в среде Oracle, должен вооружиться тремя книгами: справочником по языку SQL, таким как «Oracle SQL: The Essential Reference» (O'Reilly), руководством по оптимизации производительности, например «Oracle SQL Tuning Pocket Reference» (O'Reilly), и этой книгой, которая показывает, как наилучшим образом использовать и комбинировать разнообразные возможности Oracle SQL.

Надо надеяться, теперь вы имеете представление о том, зачем нужен и почему так важен SQL. В следующем разделе будет рассказано об истоках SQL и о поддержке стандартов SQL продуктами Oracle.

Краткая история SQL

В начале семидесятых годов двадцатого века исследователь из IBM, доктор Е. Ф. Кодд (Codd E. F.) попытался применить строгий математический подход к еще неизведанному тогда миру хранения и извлечения данных. Работа Кодда привела к определению *реляционной модели данных* (*relational data model*) и появлению языка DSL/Alpha для манипулирования данными в реляционной БД. Компании IBM понравилось увиденное, и они открыли проект под названием System/R для построения прототипа на основе работы Кодда. Помимо всего прочего команда System/R разработала упрощенную версию DSL, названную SQUARE, затем переименованную в SEQUEL и, наконец, – в SQL.

Результатом работы, проделанной в рамках System/R, стал выпуск различных продуктов IBM на базе реляционной модели. Под реляционным флагом сплотились и многие другие компании, в том числе и Oracle. К середине 80-х годов язык SQL приобрел на рынке достаточный вес для того, чтобы обратить на себя внимание Американского национального института стандартов (ANSI, American National Standards Institute). В 1986 году ANSI выпустил первый стандарт SQL, затем последовали обновления в 1989, 1992 и 1999 годах.

Спустя тридцать лет после того, как команда System/R начала работать с прототипом реляционной базы данных, SQL все еще занимает устойчивое положение на рынке. Было много попыток свержения реляционных баз данных. Несмотря на это хорошо спроектированные реляционные базы данных вместе с правильно написанными операторами SQL успешно применяются для решения задач обработки больших объемов сложных данных, в то время как другие методы терпят неудачу.

Реализация SQL от Oracle

Зная, что Oracle рано встала на сторону реляционной модели и SQL, можно подумать, что компанией было приложено немало усилий для достижения соответствия стандартам ANSI. Однако продолжительное время парням из Oracle казалось достаточным, что их реализация SQL была функционально эквивалентна ANSI-стандартам, и полное соответствие их не очень волновало. Начиная же с версии Oracle8*i* компания стала работать над соответствием ANSI, и в инструментарии Oracle появились оператор CASE и новый синтаксис левого/правого/полного внешнего объединения.

Ирония в том, что, похоже, бизнес-сообщество движется в противоположном направлении. Несколько лет назад все очень заботились о пе-

реносимости и требовали от разработчиков ANSI-совместимого SQL, который бы обеспечивал возможность реализации систем на разных процессорах баз данных. Сегодня же компании выбирают какой-либо процессор базы данных, который будет использоваться в пределах предприятия, и разрешают своим разработчикам применять полный спектр возможностей, не думая об ANSI-совместимости. Одной из причин такого поворота событий служит появление многоуровневых архитектур, где весь доступ к базе данных можно заложить на одном уровне, а не разбрасывать по всему приложению. Другой возможной причиной может быть то, что за последние пять лет на рынке СУБД появились явные лидеры, так что менеджеры ощущают меньший риск при выборе процессора базы данных.

Теоретическая и практическая терминологии

Штудируя различные труды по реляционной модели данных, вы можете встретить терминологию, которая не будет использоваться в этой книге (например, такие понятия, как *отношения* (*relations*) или *кортежи* (*tuples*)). Мы будем применять практические термины, например таблицы и строки. Говоря же о различных частях оператора SQL, будем называть их по имени, а не по выполняемой функции (например, «инструкция SELECT», а не *проекция* (*projection*)). При всем уважении к доктору Кодду надо сказать, что слово *кортеж* в деловых кругах не произносят, а так как эта книга написана для тех, кто применяет продукты Oracle для решения задач бизнеса, *кортежей* вы не встретите.

Простая база данных

Данная книга является практическим руководством, поэтому в ней много примеров. Чтобы не придумывать в каждом разделе каждой главы новые наборы таблиц и столбцов, мы решили построить одну простую схему, которая будет применяться в большей части примеров. Выбранной для моделирования предметной областью является работа дистрибьютора (например, оптового продавца автомобильных запчастей или поставщика медицинского оборудования). В этом бизнесе от покупателей поступают заказы на детали, предоставляемые внешними поставщиками. На рис. 1.1 показана модель «объект-отношение» для такого вида деятельности.

Приведем краткое описание модели объект-отношение для тех, кто не знаком с ней. Каждый прямоугольник в модели – это *объект* (*entity*), соответствующий таблице[1] в базе данных. Линии между объектами

[1] В зависимости от назначения модели данных объекты могут соответствовать или не соответствовать таблицам базы данных. Например, *логическая модель* описывает бизнес-объекты и их отношения, в то время как *физическая модель* иллюстрирует таблицы и их первичные/внешние ключи. Модель на рис. 1.1 – это физическая модель.

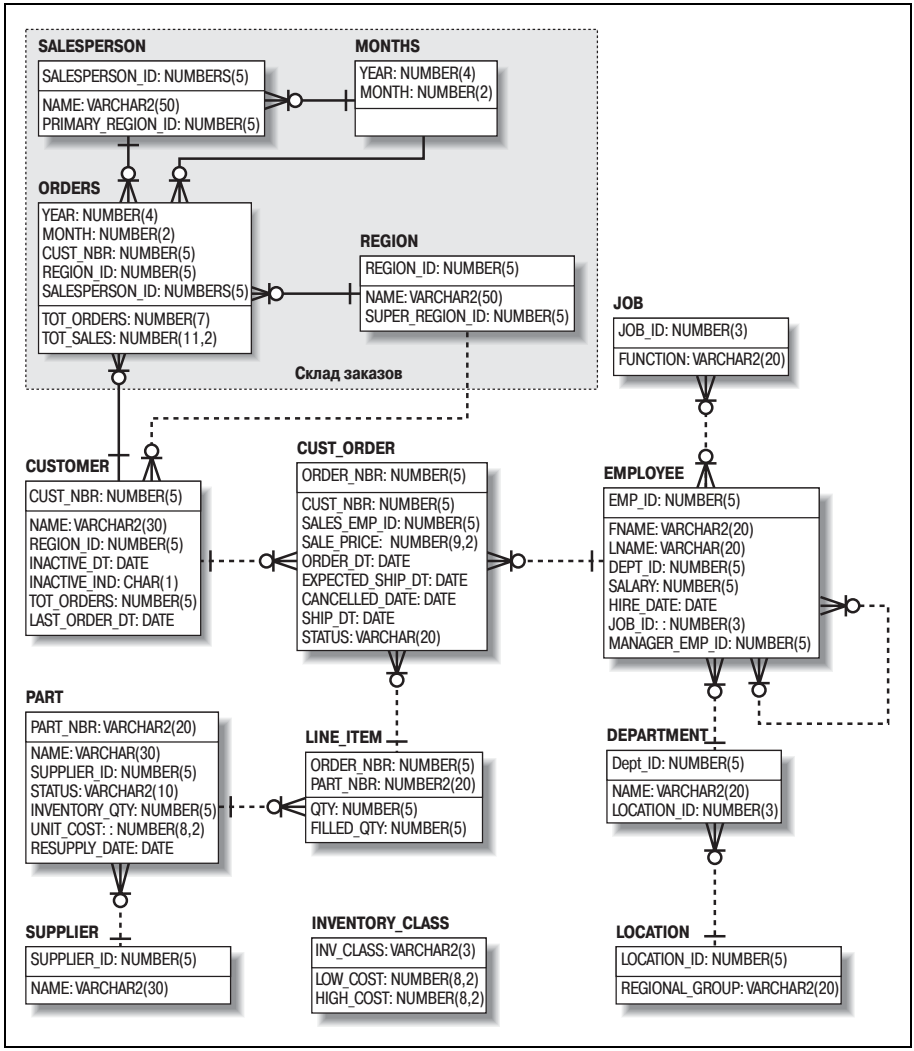


Рис. 1.1. *Модель дистрибьютора деталей*

представляют *отношения* (*relationships*) между таблицами, соответствующие внешним ключам. Например, таблица `CUST_ORDER` (заказы покупателей) содержит внешний ключ для таблицы `EMPLOYEE` (сотрудники), указывающий на продавца, ответственного за определенный заказ. Практически это означает, что таблица `CUST_ORDER` содержит столбец с идентификаторами (ID) сотрудников и что для каждого заказа этот идентификатор указывает на сотрудника, обработавшего данный заказ. Если вам что-то непонятно, используйте схему просто как иллюстрацию таблиц и столбцов, встречающихся в нашей базе данных.

По мере работы с примерами возвращайтесь время от времени к этой диаграмме, и вы увидите, что начинаете разбираться в отношениях.

Операторы DML

В этом разделе вы познакомитесь с четырьмя операторами DML. Представленной информации должно быть достаточно для того, чтобы начать писать операторы DML. Однако кажущаяся простота DML обманчива (поговорим об этом в конце раздела), так что помните, что у DML множество аспектов, и лишь о немногих из них мы здесь поговорим.

Оператор SELECT

Оператор SELECT используется для извлечения данных из базы. Множество данных, извлекаемое оператором SELECT, называется *результирующим множеством* (*result set*). Как и таблица, результирующее множество состоит из строк и столбцов, что позволяет заполнить таблицу данными результирующего множества. Общий вид оператора SELECT таков:

```
SELECT <один или несколько объектов>
FROM <одно или несколько мест>
WHERE <ни одного, одно или несколько условий>
```

Инструкции SELECT и FROM необходимы, а вот инструкция WHERE необязательна (хотя и она почти всегда используется). Начнем с простого примера, извлекающего три столбца из каждой строки таблицы CUSTOMER (заказчики):

```
SELECT cust_nbr, name, region_id
FROM customer;
```

```
  CUST_NBR NAME                          REGION_ID
---------- ----------------------------- ----------
         1 Cooper Industries                      5
         2 Emblazon Corp.                         5
         3 Ditech Corp.                           5
         4 Flowtech Inc.                          5
         5 Gentech Industries                     5
         6 Spartan Industries                     6
         7 Wallace Labs                           6
         8 Zantech Inc.                           6
         9 Cardinal Technologies                  6
        10 Flowrite Corp.                         6
        11 Glaven Technologies                    7
        12 Johnson Labs                           7
        13 Kimball Corp.                          7
        14 Madden Industries                      7
        15 Turntech Inc.                          7
```

```
        16 Paulson Labs                          8
        17 Evans Supply Corp.                    8
        18 Spalding Medical Inc.                 8
        19 Kendall-Taylor Corp.                  8
        20 Malden Labs                           8
        21 Crimson Medical Inc.                  9
        22 Nichols Industries                    9
        23 Owens-Baxter Corp.                    9
        24 Jackson Medical Inc.                  9
        25 Worcester Technologies                9
        26 Alpha Technologies                   10
        27 Phillips Labs                        10
        28 Jaztech Corp.                        10
        29 Madden-Taylor Inc.                   10
        30 Wallace Industries                   10
```

Инструкция WHERE не была использована, и никаких ограничений на данные мы не наложили, поэтому запрос возвращает все строки таблицы заказчиков. Если необходимо ограничить возвращаемый набор данных, можно добавить в оператор инструкцию WHERE с одним условием:

```
SELECT cust_nbr, name, region_id
FROM customer
WHERE region_id = 8;

  CUST_NBR NAME                           REGION_ID
---------- ------------------------------ ----------
        16 Paulson Labs                            8
        17 Evans Supply Corp.                      8
        18 Spalding Medical Inc.                   8
        19 Kendall-Taylor Corp.                    8
        20 Malden Labs                             8
```

Теперь результирующее множество содержит только заказчиков, проживающих в области с идентификатором region_id, равным 8. Но что если нужно ссылаться на область по имени, а не по номеру? Можно выбрать нужное имя из таблицы REGION, а затем, зная region_id, обратиться к таблице CUSTOMER. Чтобы не писать два разных запроса, можно получить тот же результат с помощью одного запроса, использующего *объединение* (*join*):

```
SELECT customer.cust_nbr, customer.name, region.name
FROM customer, region
WHERE region.name = 'New England'
  AND region.region_id = customer.region_id;

  CUST_NBR NAME                           NAME
---------- ------------------------------ -----------
         1 Cooper Industries              New England
         2 Emblazon Corp.                 New England
         3 Ditech Corp.                   New England
         4 Flowtech Inc.                  New England
         5 Gentech Industries             New England
```

Теперь в инструкции FROM не одна таблица, а две, и инструкция WHERE содержит *условие объединения* (*join condition*), которое указывает, что таблицы заказчиков и областей должны быть объединены по столбцу region_id, имеющемуся в каждой таблице. Объединения и их условия будут детально рассмотрены в главе 3.

Так как обе таблицы содержат столбец с названием *name*, необходимо как-то определить, какой именно столбец вас интересует. В предыдущем примере это делается с помощью точечной нотации – добавления через точку имени таблицы перед именем столбца. Если же на написание полных имен таблиц у вас уходит слишком много времени, назначьте для каждого названия таблицы в инструкции FROM *псевдоним* (*alias*) и используйте его вместо имени в инструкциях SELECT и WHERE:

```
SELECT c.cust_nbr, c.name, r.name
FROM customer c, region r
WHERE r.name = 'New England'
  AND r.region_id = c.region_id;
```

В этом примере псевдоним «c» был присвоен таблице заказчиков, а псевдоним «r» – таблице областей. Теперь можно в инструкциях SELECT и WHERE писать «c» вместо «customer» и «r» вместо «region».

Элементы инструкции SELECT

Рассмотренные ранее результирующие множества, порожденные нашими запросами, содержали столбцы одной или нескольких таблиц. Как правило, элементами инструкции SELECT действительно являются ссылки на столбцы; среди них также могут встречаться:

- Константы, такие как числа (1) или строки ('abc')
- Выражения, например shape.diameter * 3.1415927
- Функции, такие как TO_DATE('01-JAN-2002','DD-MON-YYYY')
- Псевдостолбцы, например ROWID, ROWNUM или LEVEL

Если первые три пункта в списке довольно просты, то последний нуждается в пояснении. В Oracle есть несколько столбцов-призраков, называемых *псевдостолбцами* (*pseudocolumns*), которые не присутствуют ни в одной таблице. Их значения появляются во время выполнения запроса, и в некоторых ситуациях они бывают полезны.

Например, псевдостолбец ROWID представляет физическое положение строки, обеспечивая быстрейший механизм доступа к строкам. Он может оказаться полезным, если вы планируете удалять или обновлять полученные в результате запроса строки. Но никогда не храните значения ROWID в базе данных и не ссылайтесь на них за пределами транзакции, в которой они были получены, так как в определенных ситуациях значение ROWID для строки может поменяться, а при удалении строки ее значение ROWID может перейти к другой.

Приведем пример, демонстрирующий использование всех элементов из списка:

```
SELECT rownum,
  cust_nbr,
  1 multiplier,
  'cust # ' || cust_nbr cust_nbr_str,
  'hello' greeting,
  TO_CHAR(last_order_dt, 'DD-MON-YYYY') last_order
FROM customer;

ROWNUM CUST_NBR MULTIPLIER CUST_NBR_STR GREETING LAST_ORDER
------ -------- ---------- ------------ -------- -----------
     1        1          1 cust # 1     hello    15-JUN-2000
     2        2          1 cust # 2     hello    27-JUN-2000
     3        3          1 cust # 3     hello    07-JUL-2000
     4        4          1 cust # 4     hello    15-JUL-2000
     5        5          1 cust # 5     hello    01-JUN-2000
     6        6          1 cust # 6     hello    10-JUN-2000
     7        7          1 cust # 7     hello    17-JUN-2000
     8        8          1 cust # 8     hello    22-JUN-2000
     9        9          1 cust # 9     hello    25-JUN-2000
    10       10          1 cust # 10    hello    01-JUN-2000
    11       11          1 cust # 11    hello    05-JUN-2000
    12       12          1 cust # 12    hello    07-JUN-2000
    13       13          1 cust # 13    hello    07-JUN-2000
    14       14          1 cust # 14    hello    05-JUN-2000
    15       15          1 cust # 15    hello    01-JUN-2000
    16       16          1 cust # 16    hello    31-MAY-2000
    17       17          1 cust # 17    hello    28-MAY-2000
    18       18          1 cust # 18    hello    23-MAY-2000
    19       19          1 cust # 19    hello    16-MAY-2000
    20       20          1 cust # 20    hello    01-JUN-2000
    21       21          1 cust # 21    hello    26-MAY-2000
    22       22          1 cust # 22    hello    18-MAY-2000
    23       23          1 cust # 23    hello    08-MAY-2000
    24       24          1 cust # 24    hello    26-APR-2000
    25       25          1 cust # 25    hello    01-JUN-2000
    26       26          1 cust # 26    hello    21-MAY-2000
    27       27          1 cust # 27    hello    08-MAY-2000
    28       28          1 cust # 28    hello    23-APR-2000
    29       29          1 cust # 29    hello    06-APR-2000
    30       30          1 cust # 30    hello    01-JUN-2000
```

Интересно, что в инструкции SELECT совсем необязательно ссылаться на столбцы таблиц инструкции FROM. Например, результирующее множество следующего запроса целиком построено из констант:

```
SELECT 1 num, 'abc' str
FROM customer;
```

```
       NUM STR
---------- ---
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
         1 abc
```

Таблица CUSTOMER содержит 30 строк, поэтому результирующее множество состоит из 30 одинаковых строк данных.

Упорядочение результатов

В общем случае нет гарантии, что результирующее множество будет сформировано в каком-либо определенном порядке. Если нужно отсортировать результаты по одному или нескольким столбцам, следует добавить инструкцию ORDER BY сразу же после WHERE. Следующий пример сортирует заказчиков из Новой Англии по фамилиям:

```
SELECT c.cust_nbr, c.name, r.name
FROM customer c, region r
WHERE r.name = 'New England'
  AND r.region_id = c.region_id
ORDER BY c.name;
```

```
CUST_NBR NAME                           NAME
-------- ------------------------------ -----------
       1 Cooper Industries              New England
```

```
         3 Ditech Corp.                  New England
         2 Emblazon Corp.                New England
         4 Flowtech Inc.                 New England
         5 Gentech Industries            New England
```

Сортируемые столбцы можно определять по их положению в инструкции SELECT. Отсортируем предыдущий запрос по столбцу CUST_NBR (номер заказчика), который в инструкции SELECT указан первым:

```
SELECT c.cust_nbr, c.name, r.name
FROM customer c, region r
WHERE r.name = 'New England'
  AND r.region_id = c.region_id
ORDER BY 1;

  CUST_NBR NAME                           NAME
---------- ------------------------------ -----------
         1 Cooper Industries              New England
         2 Emblazon Corp.                 New England
         3 Ditech Corp.                   New England
         4 Flowtech Inc.                  New England
         5 Gentech Industries             New England
```

Указание ключей сортировки по позиции сэкономит вам немного времени, но может привести к ошибкам, если в дальнейшем порядок следования столбцов в инструкции SELECT будет изменен.

Удаление дубликатов

В некоторых случаях результирующее множество может содержать одинаковые данные. Например, при формировании списка проданных за последний месяц деталей те детали, которые присутствовали в нескольких заказах, встретятся в результирующем множестве несколько раз. Чтобы исключить повторы, вставьте в инструкцию SELECT ключевое слово DISTINCT:

```
SELECT DISTINCT li.part_nbr
FROM cust_order co, line_item li
WHERE co.order_dt >= TO_DATE('01-JUL-2001','DD-MON-YYYY')
  AND co.order_dt < TO_DATE('01-AUG-2001','DD-MON-YYYY')
  AND co.order_nbr = li.order_nbr;
```

Запрос возвращает множество отличающихся друг от друга записей о деталях, заказанных в течение июля 2001 года. Без ключевого слова DISTINCT вывод содержал бы по одной строке для каждой строки каждого заказа, и одна деталь могла бы встречаться несколько раз, если бы она содержалась в нескольких заказах. Принимая решение о включении DISTINCT в инструкцию SELECT, следует иметь в виду, что поиск и удаление дубликатов требует сортировки, которая будет служить дополнительной нагрузкой при выполнении запроса.

Оператор INSERT

Оператор INSERT – это механизм загрузки данных в базу данных. За один раз данные можно вставить только в одну таблицу, зато брать вставляемые данные можно из нескольких дополнительных таблиц. Вставляя данные в таблицу, не нужно указывать значения для каждого столбца, однако следует обратить внимание на то, допускают ли столбцы использование значений NULL[1] или же нет. Посмотрим на определение таблицы EMPLOYEE (служащие):

```
describe employee

Name                                    Null?    Type
--------------------------------------- -------- ------------
EMP_ID                                  NOT NULL NUMBER(5)
FNAME                                            VARCHAR2(20)
LNAME                                   NOT NULL VARCHAR2(20)
DEPT_ID                                 NOT NULL NUMBER(5)
MANAGER_EMP_ID                                   NUMBER(5)
SALARY                                           NUMBER(5)
HIRE_DATE                                        DATE
JOB_ID                                           NUMBER(3)
```

Пометка NOT NULL для столбцов emp_id, lname и dept_id означает, что они обязательны для заполнения. Поэтому в операторе INSERT необходимо указать значения как минимум для трех этих столбцов, например, как показано ниже:

```
INSERT INTO employee (emp_id, lname, dept_id)
VALUES (101, 'Smith', 2);
```

Количество элементов в инструкции VALUES должно совпадать с количеством элементов в списке столбцов, а их типы данных должны соответствовать определениям столбцов. В нашем примере emp_id и dept_id хранят численные значения, а lname – строковое, поэтому оператор INSERT выполнится без ошибок. Oracle всегда автоматически пытается преобразовать один тип данных в другой, поэтому следующий оператор тоже выполнится без ошибок:

```
INSERT INTO employee (emp_id, lname, dept_id)
VALUES ('101', 'Smith', '2');
```

Иногда вставляемые данные нужно предварительно извлечь из одной или нескольких таблиц. Так как оператор SELECT формирует результирующее множество, состоящее из строк и столбцов, то можно непосредственно передать его оператору INSERT:

```
INSERT INTO employee (emp_id, fname, lname, dept_id, hire_date)
SELECT 101, 'Dave', 'Smith', d.dept_id, SYSDATE
```

[1] NULL означает отсутствие значения; подробно обсуждается в главе 2.

```
FROM department d
WHERE d.name = 'Accounting';
```

В данном примере оператор SELECT извлекает идентификатор отдела для бухгалтерии (Accounting). Остальные четыре столбца в операторе SELECT представлены константами.

Оператор DELETE

Оператор DELETE обеспечивает удаление данных из базы. Как и SELECT, оператор DELETE содержит инструкцию WHERE с условиями для идентификации удаляемых строк. Забыв указать инструкцию WHERE в операторе DELETE, вы удалите все строки из указанной таблицы. Следующий оператор удаляет всех сотрудников с фамилией Hooper из таблицы EMPLOYEE:

```
DELETE FROM employee
WHERE lname = 'Hooper';
```

Иногда значения, необходимые для построения условия в инструкции WHERE, располагаются в других таблицах. Например, решение компании вынести вовне функции бухучета потребует удаления всего бухгалтерского персонала из таблицы EMPLOYEE:

```
DELETE FROM employee
WHERE dept_id =
 (SELECT dept_id
  FROM department
  WHERE name = 'Accounting');
```

Подобное использование оператора SELECT носит название *подзапроса* (*subquery*) и будет изучено в главе 5.

Оператор UPDATE

С помощью оператора UPDATE вносятся изменения в существующие данные. Как и DELETE, оператор UPDATE включает в себя инструкцию WHERE для указания тех строк, которые будут изменены. Посмотрим, как можно предоставить 10-процентное повышение зарплаты тем, у кого годовой доход меньше 40 000 долларов:

```
UPDATE employee
SET salary = salary * 1.1
WHERE salary < 40000;
```

Если необходимо изменить несколько столбцов, вы можете выбрать один из двух вариантов: задать набор пар столбец-значение, разделенных запятыми, или указать набор столбцов и подзапрос. Два следующих оператора UPDATE изменяют столбцы inactive_dt и inactive_ind в

таблице CUSTOMER для клиентов, не сделавших ни одного заказа за последний год:

```
UPDATE customer
SET inactive_dt = SYSDATE, inactive_ind = 'Y'
WHERE last_order_dt < SYSDATE - 365;

UPDATE customer
SET (inactive_dt, inactive_ind) =
 (SELECT SYSDATE, 'Y' FROM dual)
WHERE last_order_dt < SYSDATE - 365;
```

Подзапрос во втором примере выглядит немного неестественно, так как он обращается к таблице dual[1] для построения результирующего множества, состоящего из двух констант; он приведен для иллюстрации использования подзапросов в операторе UPDATE. В следующих главах вы найдете намного более интересные примеры использования подзапросов.

И зачем остальные 13 глав?

Прочтя эту главу, вы могли подумать, что SQL (или, по крайней мере, DML) чрезвычайно прост. Да, на поверхности он кажется простым, и вы уже знаете об этом языке достаточно, чтобы писать программы. Но со временем вы поймете, что существует множество путей, ведущих к одной и той же цели, и некоторые из них гораздо эффективнее и элегантнее, чем другие. Настоящий мастер SQL может разом стереть весь написанный за год код и переделать его заново. Один из нас шел к этому девять лет. Надеемся, что эта книга поможет сократить ваш путь.

Читая оставшуюся часть книги, вы заметите, что большинство примеров – это операторы SELECT, а остаток практически равномерно распределен между операторами INSERT, UPDATE и DELETE. Такое неравенство не означает, что SELECT важнее других операторов DML. Операторы SELECT преобладают, потому что есть возможность показать результат выполнения, что помогает лучше понять запрос. К тому же, многие приемы, использованные в операторах SELECT, также применимы и к операторам UPDATE и DELETE.

[1] Dual – это специальная таблица Oracle, состоящая ровно из одной строки и одного столбца. Она бывает полезной, когда необходимо построить запрос, возвращающий ровно одну строку.

2

Инструкция WHERE

Инструкция WHERE – это тот механизм, который необходим для идентификации набора данных при организации запроса, изменении или удалении данных. В этой главе будет рассмотрена роль инструкции WHERE в операторах SQL, а также различные варианты построения этой инструкции.

Жизнь без WHERE

Прежде чем приступить к исследованию инструкции WHERE, давайте попробуем представить, что ее не существует. Пусть, например, речь идет о работе с таблицей деталей (part). Чтобы получить данные из этой таблицы, создадим следующий запрос:

```
SELECT part_nbr, name, supplier_id, status, inventory_qty
FROM part;
```

Если таблица деталей содержит 10 000 элементов, то возвращенное запросом результирующее множество будет состоять из 10 000 строк, в каждой из которых 5 столбцов. Затем предстоит загрузить эти 10 000 строк в оперативную память и произвести необходимые изменения.

После того как данные, находящиеся в оперативной памяти, изменены, необходимо внести эти изменения в таблицу деталей. Указать, какие строки необходимо изменить, невозможно, так что остается только удалить все строки таблицы и заново вставить все 10 000 строк:

```
DELETE FROM part;

INSERT INTO part (part_nbr, name, supplier_id, status, inventory_qty)
VALUES ('XY5-1002', 'Wonder Widget', 1, 'IN-STOCK', 1);

/* Остальные 9 999 операторов INSERT */
```

Теоретически такой подход возможен, но он губителен для производительности, параллелизма (возможности одновременного изменения данных несколькими пользователями) и масштабируемости.

Пусть теперь необходимо изменить только те данные таблицы, которые относятся к деталям одного поставщика – Acme Industries. Имена поставщиков хранятся в таблице `supplier`, поэтому в инструкции `FROM` следует указать обе таблицы: деталей (part) и поставщиков (supplier):

```
SELECT p.part_nbr, p.name, p.supplier_id, p.status, p.inventory_qty,
  s.supplier_id, s.name
FROM part p, supplier s;
```

Если 100 компаний поставляют 10 000 деталей (таблица `part`), то такой запрос вернет 1 000 000 строк. Это число, называемое прямым, или декартовым, произведением (Cartesian product), равно количеству всех возможных комбинаций строк двух таблиц. «Просеивая» миллион строк, будем искать те, в которых значения `p.supplier_id` и `s.supplier_id` равны и значение столбца `s.name` равно `'Acme Industries'`. Если фирма Acme Industries поставляет всего 50 из 10 000 деталей, представленных в базе данных, нам предстоит отбросить 999 950 из 1 000 000 строк, возвращенных запросом.

На помощь приходит WHERE

Эти примеры наглядно иллюстрируют полезность инструкции `WHERE`, которая предоставляет возможность:

1. Отфильтровывать ненужные данные из результирующего множества.
2. Изолировать одну или несколько строк таблицы для изменения.
3. Выполнить условное объединение двух или более наборов данных.

Чтобы посмотреть, как все это работает, давайте добавим инструкцию `WHERE` в написанный ранее оператор `SELECT`, который занимается поиском всех деталей, поставляемых Acme Industries:

```
SELECT p.part_nbr, p.name, p.supplier_id, p.status, p.inventory_qty,
  s.supplier_id, s.name
FROM part p, supplier s
WHERE s.supplier_id = p.supplier_id
  AND s.name = 'Acme Industries';
```

В данном случае инструкция `WHERE` состоит из двух частей, называемых *условиями* (*conditions*), которые вычисляются и оцениваются по отдельности. Значения условий всегда равны или `TRUE`, или `FALSE`. Если в инструкцию `WHERE` входит несколько условий, то для того чтобы некоторая строка была включена в результирующее множество, значения всех условий для нее должны быть равны `TRUE`.[1] В рассматриваемом

примере строка, образованная при объединении данных из таблиц part и supplier, будет включена в окончательное результирующее множество только в том случае, если обе таблицы имеют одинаковое значение столбца supplier_id и значение столбца name таблицы supplier совпадает с 'Acme Industries'.[1] Любые другие сочетания данных из двух таблиц будут оценены как FALSE и отброшены.

С добавлением в рассмотренный ранее пример инструкции WHERE работу по отсеиванию ненужных строк результирующего множества берет на себя Oracle, и запрос вернет всего 50, а не 1 000 000 строк. После того как нужные 50 строк извлечены из базы данных, можно приступать к изменению данных. Помните, что теперь, когда в вашем распоряжении есть инструкция WHERE, больше не нужно удалять и вновь вставлять измененные данные, вместо этого можно просто использовать оператор UPDATE для изменения отдельных строк на основе значения столбца part_nbr, который является уникальным идентификатором строки таблицы:

```
UPDATE part
SET status = 'DISCONTINUED'
WHERE part_nbr = 'AI5-4557';
```

Хотя это явное усовершенствование, можно пойти и дальше. Если планируется изменить статус для всех 50 деталей, поставленных Acme Industries, то в запросе вообще нет необходимости. Просто выполняем оператор UPDATE, который находит и изменяет все 50 записей:

```
UPDATE part
SET status = 'DISCONTINUED'
WHERE supplier_id =
 (SELECT supplier_id
  FROM supplier
  WHERE name = 'Acme Industries');
```

Инструкция WHERE этого оператора состоит из одного условия, которое определяет равенство столбца supplier_id значению, возвращаемому запросом к таблице supplier. Запрос, заключенный в скобки внутри другого SQL-оператора, называется *подзапросом* (*subquery*); о подзапросах будет подробно рассказано в главе 5, пока же просто не пугай-

[1] Это чрезмерно упрощенное пояснение. Далее вы узнаете, что использование операторов OR и NOT может привести к тому, что значение инструкции WHERE будет вычислено как TRUE несмотря на то, что отдельные условия оцениваются как FALSE.

[1] Еще одно упрощение. Оптимизатор Oracle (компонент, задача которого состоит в поиске наиболее эффективного способа выполнения запроса) не создает все возможные комбинации строк всех таблиц или представлений, входящих в инструкцию FROM прежде, чем вычислять условия. Оптимизатор выбирает порядок, в котором следует вычислять условия и объединять данные, чтобы время исполнения было минимальным.

тесь их. В результате условие будет переписано с использованием величины, возвращенной подзапросом, например:

```
UPDATE part
SET status = 'DISCONTINUED'
WHERE supplier_id = 1;
```

Условие оценивается как TRUE для 50 из 10 000 строк таблицы part, и значение поля status для этих 50 строк меняется на 'DISCONTINUED'.

Вычисление инструкции WHERE

Вы видели, как работает инструкция WHERE, теперь давайте поговорим о том, как она вычисляется. Уже упоминалось, что инструкция WHERE состоит из одного или нескольких условий, которые независимо друг от друга оцениваются как TRUE или FALSE. Если инструкция WHERE состоит из нескольких условий, такие условия разделяются логическими операторами AND и OR. В зависимости от значения каждого отдельного условия и размещения между ними логических операторов Oracle присваивает соответствующее конечное значение TRUE или FALSE каждой «испытываемой» строке, определяя тем самым, войдет ли данная строка в итоговое результирующее множество.

Давайте вернемся к запросу об Acme Industries:

```
SELECT p.part_nbr, p.name, p.supplier_id, p.status, p.inventory_qty,
  s.supplier_id, s.name
FROM part p, supplier s
WHERE s.supplier_id = p.supplier_id
  AND s.name = 'Acme Industries';
```

Инструкция WHERE состоит из двух условий, разделенных оператором AND. Поэтому строка будет включена в результирующее множество, только если оба условия будут вычислены как TRUE. Варианты оценки нескольких условий, разделенных AND, приведены в табл. 2.1 (условия заменены своими возможными значениями TRUE и FALSE).

Таблица 2.1. Вычисление нескольких условий, разделенных оператором AND

Промежуточный результат	Конечный результат
WHERE TRUE AND TRUE	TRUE
WHERE FALSE AND FALSE	FALSE
WHERE FALSE AND TRUE	FALSE
WHERE TRUE AND FALSE	FALSE

Используя основные логические правила, устанавливаем, что единственная комбинация результатов оценки отдельных условий, итоговое

значение которой для исследуемой строки равно TRUE, возникает в том случае, когда оба условия вычислены как TRUE. Варианты результатов для условий, разделенных оператором OR, приведены в табл. 2.2.

Таблица 2.2. Вычисление нескольких условий, разделенных оператором OR

Промежуточный результат	Конечный результат
WHERE TRUE OR TRUE	TRUE
WHERE FALSE OR FALSE	FALSE
WHERE FALSE OR TRUE	TRUE
WHERE TRUE OR FALSE	TRUE

Давайте теперь немного оживим наш запрос, включив в него детали, как поставляемые Acme Industries, так и полученные от Tilton Enterprises:

```
SELECT p.part_nbr, p.name, p.supplier_id, p.status, p.inventory_qty,
  s.supplier_id, s.name
FROM part p, supplier s
WHERE s.supplier_id = p.supplier_id
  AND (s.name = 'Acme Industries'
    OR s.name = 'Tilton Enterprises');
```

Теперь в инструкции WHERE три условия, разделенные AND и OR, при этом два условия заключены в скобки. Возможные результаты оценки приведены в табл. 2.3.

Таблица 2.3. Вычисление нескольких условий, разделенных операторами OR и AND

Промежуточный результат	Конечный результат
WHERE TRUE AND (TRUE OR FALSE)	TRUE
WHERE TRUE AND (FALSE OR TRUE)	TRUE
WHERE TRUE AND (FALSE OR FALSE)	FALSE
WHERE FALSE AND (TRUE OR FALSE)	FALSE
WHERE FALSE AND (FALSE OR TRUE)	FALSE
WHERE FALSE AND (FALSE OR FALSE)	FALSE

Так как одна деталь не может быть получена одновременно и от Acme Industries и от Tilton Enterprises, такие промежуточные результаты, как TRUE AND (TRUE AND TRUE) и FALSE AND (TRUE AND TRUE), не рассматриваются.

Чтобы сделать все еще интереснее, введем оператор NOT. Следующий запрос возвращает данные о деталях, полученных не от Acme Industries или Tilton Enterprises:

```
SELECT p.part_nbr, p.name, p.supplier_id, p.status, p.inventory_qty,
  s.supplier_id, s.name
FROM part p, supplier s
WHERE s.supplier_id = p.supplier_id
  AND NOT (s.name = 'Acme Industries'
    OR s.name = 'Tilton Enterprises');
```

В табл. 2.4 можно увидеть, как добавление оператора NOT повлияло на вычисляемый результат.

Таблица 2.4. Вычисление нескольких условий, разделенных операторами OR, AND и NOT

Промежуточный результат	Конечный результат
WHERE TRUE AND NOT (TRUE OR FALSE)	FALSE
WHERE TRUE AND NOT (FALSE OR TRUE)	FALSE
WHERE TRUE AND NOT (FALSE OR FALSE)	TRUE
WHERE FALSE AND NOT (TRUE OR FALSE)	FALSE
WHERE FALSE AND NOT (FALSE OR TRUE)	FALSE
WHERE FALSE AND NOT (FALSE OR FALSE)	FALSE

Надо сказать, что использование оператора NOT в предыдущем примере несколько надуманно; позже вы узнаете, что существуют более естественные пути выражения той же логики.

Условия и выражения

Как вычисляются и группируются условия, вы уже знаете, пришло время поговорить об элементах, составляющих условия. Условие состоит из одного или нескольких *выражений* (*expressions*) и одного или нескольких *операторов* (*operators*). К выражениям относятся:

- Числа
- Столбцы, например supplier_id
- Константы, например 'Acme Industries'
- Функции, например UPPER('abcd')
- Списки простых выражений, например (1, 2, 3)
- Подзапросы

К операторам относятся:

- Арифметические операторы, такие как +, -, * и /
- Операторы сравнения, такие как =, <, >=, !=, LIKE и IN

В последующих разделах будут рассмотрены наиболее распространенные типы условий, использующие различные комбинации перечисленных выше типов выражений и операторов.