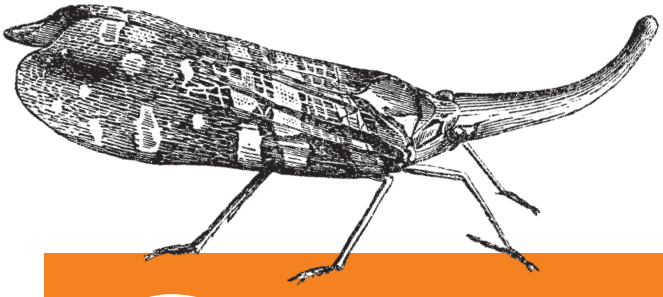
*Эффективная работа с Oracle SQL*

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** . . . . . 11

**1. Введение в SQL** . . . . . 19

Что такое SQL? . . . . . 19
Краткая история SQL . . . . . 21
Простая база данных . . . . . 22
Операторы DML. . . . . 24

**2. Инструкция WHERE** . . . . . 33

Жизнь без WHERE. . . . . 33
На помощь приходит WHERE. . . . . 34
Вычисление инструкции WHERE . . . . . 36
Условия и выражения . . . . . 38
Куда идем дальше? . . . . . 44

**3. Объединения** . . . . . 46

Внутренние объединения . . . . . 47
Внешние объединения. . . . . 50
Самообъединения . . . . . 59
Объединения и подзапросы . . . . . 64
Операторы DML и представление объединения . . . . . 64
Синтаксис объединения стандарта ANSI в Oracle9i . . . . . 71

**4. Групповые операции** . . . . . 78

Обобщающие функции . . . . . 78
Инструкция GROUP BY . . . . . 82
Инструкция HAVING . . . . . 88

**5. Подзапросы** . . . . . 91

Что такое подзапрос? . . . . . 91
Несвязанные подзапросы . . . . . 92
Связанные подзапросы . . . . . 99

Встроенные представления . . . . . 101
Изучаем пример подзапроса: N лучших работников . . . . . 115

**6. Обработка дат и времени** . . . . . 121

Внутренний формат хранения даты . . . . . 122
Вставка дат в БД и извлечение дат из БД . . . . . 122
Работа с датами . . . . . 137
Новые возможности Oracle9i по обработке даты и времени . . . . . 152
Литералы INTERVAL . . . . . 160

**7. Операции над множествами** . . . . . 173

Операторы работы с множествами . . . . . 174
Использование операций над множествами
для сравнения двух таблиц . . . . . 178
Использование NULL в составных запросах . . . . . 181
Правила и ограничения, налагаемые
на операции над множествами . . . . . 183

**8. Иерархические запросы** . . . . . 187

Представление иерархической информации . . . . . 187
Простые операции над иерархическими данными . . . . . 190
Расширения Oracle SQL . . . . . 193
Сложные иерархические операции . . . . . 198
Ограничения, налагаемые на иерархические запросы . . . . . 205

**9. DECODE и CASE** . . . . . 207

DECODE, NVL и NVL2 . . . . . 207
История CASE . . . . . 211
Примеры использования DECODE и CASE . . . . . 214

**10. Разделы, объекты и коллекции** . . . . . 226

Разделение таблиц . . . . . 226
Объекты и коллекции . . . . . 237

**11. PL/SQL** . . . . . 249

Что такое PL/SQL? . . . . . 249
Процедуры, функции и пакеты . . . . . 250
Вызов хранимых функций из запросов . . . . . 252
Ограничения на вызов PL/SQL из SQL . . . . . 257
Хранимые функции в операторах DML . . . . . 261
SQL внутри PL/SQL . . . . . 263

**12. Сложные групповые операции** . . . . . 266
ROLLUP . . . . . 266
CUBE . . . . . 276
Функция GROUPING . . . . . 283
GROUPING SETS . . . . . 288
Возможности группировки в Oracle9i . . . . . 289
Функции GROUPING_ID и GROUP_ID . . . . . 299

**13. Аналитический SQL** . . . . . 307
Обзор аналитического SQL . . . . . 307
Ранжирующие функции . . . . . 313
Оконные функции . . . . . 327
Функции для создания отчетов . . . . . 333
Резюме . . . . . 338

**14. Советы умудренных опытом** . . . . . 339
Когда и какие конструкции использовать? . . . . . 339
Избегайте ненужного разбора операторов . . . . . 345
Применяйте полностью определенный SQL для систем поддержки принятия решений . . . . . 350

**Алфавитный указатель** . . . . . 352

# 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.

|

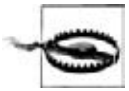
В описании синтаксиса вертикальная черта разделяет элементы списка, помещаемого в фигурные скобки, например: {TRUE | FALSE}.

...

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.

# 14

# Советы умудренных опытом

Для написания эффективного SQL-кода необходим опыт. Любой запрос можно написать несколькими способами, при этом один из них может оказаться в сотни раз медленнее другого. В этой главе будет дано несколько советов и предложено несколько идей, которые помогут сделать ваши операторы SQL более эффективными.

## Когда и какие конструкции использовать?

В зависимости от обстоятельств использование одной конструкции SQL является более предпочтительным, чем другой. Например, иногда лучше использовать оператор `EXISTS` вместо `IN`. Но для `NOT EXISTS` и `NOT IN` это уже не так. В следующем разделе будут описаны эти конструкции.

### EXISTS вместо DISTINCT

Использование ключевого слова `DISTINCT` в инструкции `SELECT` приводит к удалению из результирующего множества повторяющихся строк. Для удаления дубликатов Oracle выполняет сортировку, которая требует времени и определенного дискового пространства. Так что если присутствие в результирующем множестве повторяющихся строк допустимо, старайтесь не использовать `DISTINCT`. Если же дубликаты недопустимы или приложение не может их обработать, используйте вместо `DISTINCT` оператор `EXISTS`.

Предположим, что нужно вывести фамилии клиентов, сделавших заказы. Запрос базируется на двух таблицах: `CUSTOMER` и `CUST_ORDER`. Запрос, использующий `DISTINCT`, будет выглядеть следующим образом:

```
SELECT DISTINCT C.CUST_NBR, C.NAME
```

```
FROM CUSTOMER C, CUST_ORDER O
WHERE C.CUST_NBR = O.CUST_NBR;
```

Посмотрим на план выполнения этого запроса. Обратите внимание на операцию SORT, которая является результатом применения DISTINCT.

```
Query Plan
----------------------------------------
SELECT STATEMENT   Cost = 3056
  SORT UNIQUE
    MERGE JOIN
      INDEX FULL SCAN IND_ORD_CUST_NBR
      SORT JOIN
        TABLE ACCESS FULL CUSTOMER
```

Перепишем запрос так, чтобы в нем использовался оператор EXISTS:

```
SELECT C.CUST_NBR, C.NAME
FROM CUSTOMER C
WHERE EXISTS (SELECT 1 FROM CUST_ORDER O WHERE C.CUST_NBR = O.CUST_NBR);
```

Обратимся к плану выполнения новой версии запроса. Заметьте, как изменилась стоимость запроса (cost) по сравнению с предыдущей версией, использовавшей DISTINCT:

```
Query Plan
---------------------------------------
SELECT STATEMENT   Cost = 320
  FILTER
    TABLE ACCESS FULL CUSTOMER
    INDEX RANGE SCAN IND_ORD_CUST_NBR
```

Затраты на выполнение запроса с EXISTS составляют менее одной девятой затрат на выполнение версии с DISTINCT. Все дело в том, что удалось избежать сортировки.

## EXISTS и IN

Во многих книгах по SQL обсуждается факт лучшей производительности конструкции NOT EXISTS по сравнению с NOT IN. Наш опыт показывает, что для Oracle8*i* EXPLAIN PLAN, формируемый для NOT EXISTS, полностью эквивалентен формируемому для NOT IN, и производительность двух операторов совпадает.

Но вот сравнение EXISTS и IN – это отдельная история. В некоторых случаях EXISTS работает лучше, чем IN. Давайте рассмотрим такой пример. Используем IN для удаления заказов клиентов региона 5:

```
DELETE FROM CUST_ORDER
WHERE CUST_NBR IN
(SELECT CUST_NBR FROM CUSTOMER
WHERE REGION_ID = 5);
```

План выполнения запроса таков:

```
Query Plan
-----------------------------------
DELETE STATEMENT   Cost = 3
  DELETE  CUST_ORDER
    HASH JOIN
      TABLE ACCESS FULL CUST_ORDER
      TABLE ACCESS FULL CUSTOMER
```

Теперь посмотрим на этот же запрос, но использующий EXISTS:

```
DELETE FROM CUST_ORDER
WHERE EXISTS
(SELECT CUST_NBR FROM CUSTOMER
WHERE CUST_ORDER.CUST_NBR = CUSTOMER.CUST_NBR
AND REGION_ID = 5);
```

Обратимся к плану выполнения EXISTS-версии:

```
Query Plan
-------------------------------------------
DELETE STATEMENT   Cost = 1
  DELETE  CUST_ORDER
    FILTER
      TABLE ACCESS FULL CUST_ORDER
      TABLE ACCESS BY INDEX ROWID CUSTOMER
        INDEX UNIQUE SCAN CUSTOMER_PK
```

Отметьте разницу стоимостей запросов. Версия IN имеет стоимость 3, а EXISTS-версия – всего 1. Если используется инструкция EXISTS, то план выполнения управляется внешней таблицей, в то время как при использовании инструкции IN планом выполнения управляет таблица подзапроса. Запрос с EXISTS будет почти всегда быстрее, чем IN-запрос, за исключением тех случаев, когда таблица подзапроса имеет гораздо меньше строк, чем внешняя таблица.

## WHERE и HAVING

В главе 4 рассказывалось об инструкциях GROUP BY и HAVING. Иногда при написании запроса GROUP BY необходимо задать условие, которое может быть записано и в инструкции WHERE, и в HAVING. Если у вас есть выбор, указывайте условие в инструкции WHERE, так вы улучшите производительность запроса. Удаление строк до выполнения вычислений – это менее дорогостоящая операция, чем удаление результатов.

Давайте на примере рассмотрим преимущества WHERE над HAVING. Напишем запрос с инструкцией HAVING, который выводит количество заказов в 2000 году:

```
SELECT YEAR, COUNT(*)
FROM ORDERS
```

```
GROUP BY YEAR
HAVING YEAR = 2000;

      YEAR   COUNT(*)
---------- ----------
      2000        720
```

План выполнения данного запроса таков:

```
Query Plan
-----------------------------------------
SELECT STATEMENT   Cost = 6
  FILTER
    SORT GROUP BY
      INDEX FAST FULL SCAN ORDERS_PK
```

Теперь посмотрим на тот же самый запрос, в котором ограничение на год задано в инструкции WHERE:

```
SELECT YEAR, COUNT(*)
FROM ORDERS
WHERE YEAR = 2000
GROUP BY YEAR;

      YEAR   COUNT(*)
---------- ----------
      2000        720
```

Обратимся к его плану выполнения:

```
Query Plan
-------------------------------------
SELECT STATEMENT   Cost = 2
  SORT GROUP BY NOSORT
    INDEX FAST FULL SCAN ORDERS_PK
```

Запрос с инструкцией HAVING сначала выполняет групповую операцию, а затем фильтрует группы по указанному условию. WHERE-версия запроса фильтрует строки *до* выполнения групповой операции. Результатом фильтрации является уменьшение количества строк обобщения и, следовательно, увеличение производительности запроса.

Но знайте, что инструкция WHERE может обеспечить не все типы фильтрации. Иногда может потребоваться сначала просуммировать данные, а потом выполнять фильтрацию итоговых данных на основе суммарных значений. В таких ситуациях приходится использовать для фильтрации инструкцию HAVING, так как только HAVING «видит» обобщенные данные. Кроме того, возможны и такие случаи, когда для обеспечения соответствующей фильтрации данных необходимо использовать в запросе и инструкцию WHERE, и инструкцию HAVING. Подробную информацию об этом можно найти в главе 4.

## UNION и UNION ALL

Операции UNION и UNION ALL обсуждались в главе 7. UNION ALL объединяет результаты двух операторов SELECT. UNION объединяет результаты двух операторов SELECT и возвращает только неповторяющиеся строки объединения, удаляя дубликаты. Очевидно, что для удаления дубликатов UNION выполняет дополнительную операцию (по сравнению с UNION ALL). Этой дополнительной операцией является сортировка, которая снижает производительность. Поэтому если приложение может обрабатывать дубликаты или вы уверены, что повторений в результирующем множестве не будет, используйте UNION ALL вместо UNION.

Чтобы осознать вышесказанное, давайте рассмотрим пример. Следующий запрос использует UNION для вывода списка заказов, которые отпускаются по цене, превышающей $50, либо которые сделаны клиентом, проживающим в регионе 5.

```
SELECT ORDER_NBR, CUST_NBR FROM CUST_ORDER WHERE SALE_PRICE > 50
UNION
SELECT ORDER_NBR, CUST_NBR FROM CUST_ORDER
WHERE CUST_NBR IN
(SELECT CUST_NBR FROM CUSTOMER WHERE REGION_ID = 5);

 ORDER_NBR   CUST_NBR
---------- ----------
      1000          1
      1001          1
      1002          5
      1003          4
      1004          4
      1005          8
      1006          1
      1007          5
      1008          5
      1009          1
      1011          1
      1012          1
      1015          5
      1017          4
      1019          4
      1021          8
      1023          1
      1025          5
      1027          5
      1029          1

20 rows selected.
```

План выполнения запроса UNION таков:

```
Query Plan
-------------------------------------------------------------------------------
SELECT STATEMENT   Cost = 8
```

```
  SORT UNIQUE
    UNION-ALL
      TABLE ACCESS FULL CUST_ORDER
      HASH JOIN
        TABLE ACCESS FULL CUSTOMER
        TABLE ACCESS FULL CUST_ORDER
```

Теперь применим для получения той же самой информации UNION ALL, а не UNION:

```
SELECT ORDER_NBR, CUST_NBR FROM CUST_ORDER WHERE SALE_PRICE > 50
UNION ALL
SELECT ORDER_NBR, CUST_NBR FROM CUST_ORDER
WHERE CUST_NBR IN
(SELECT CUST_NBR FROM CUSTOMER WHERE REGION_ID = 5);

 ORDER_NBR   CUST_NBR
---------- ----------
      1001          1
      1003          4
      1005          8
      1009          1
      1012          1
      1017          4
      1021          8
      1029          1
      1001          1
      1000          1
      1002          5
      1003          4
      1004          4
      1006          1
      1007          5
      1008          5
      1009          1
      1012          1
      1011          1
      1015          5
      1017          4
      1019          4
      1023          1
      1025          5
      1027          5
      1029          1

26 rows selected.
```

Как видите, в выходных данных присутствуют повторения. Однако UNION ALL работает эффективнее, чем UNION, что видно из его плана выполнения:

```
Query Plan
-------------------------------------------------------------------------------
SELECT STATEMENT   Cost = 4
```

```
UNION-ALL
  TABLE ACCESS FULL CUST_ORDER
  HASH JOIN
    TABLE ACCESS FULL CUSTOMER
    TABLE ACCESS FULL CUST_ORDER
```

В данном плане стоимость запроса равна 4, а в предыдущем – 8. Дополнительная операция (`SORT UNIQUE`) запроса `UNION` делает его более медленным, чем `UNION ALL`.

# Избегайте ненужного разбора операторов

Прежде чем ваш SQL-код будет выполнен Oracle, он должен быть проанализирован. Важность разбора в терминах настройки SQL связана с тем, что неважно, сколько раз данный оператор SQL будет выполнен, проанализирован он должен быть только единожды. В процессе разбора выполняются следующие шаги (необязательно в приведенной последовательности):

- Проверяется синтаксис оператора SQL.
- Словарь данных просматривается для проверки определений таблиц и столбцов.
- Словарь данных просматривается для проверки прав доступа к соответствующим объектам.
- Устанавливаются блокировки разбора на требуемые объекты.
- Определяется оптимальный план выполнения.
- Оператор загружается в разделяемую область SQL (также называемую библиотечным кэшем) разделяемого пула системной глобальной области (SGA, system global area). План выполнения и информация о разборе оператора сохраняются и затем используются, если этот же оператор будет выполнен вновь.

Если оператор SQL затрагивает удаленные объекты (например, используется соединение с внешней базой данных), эти шаги повторяются для удаленных объектов. Как видите, в ходе анализа выполняется огромная работа. Однако разбор оператора выполняется, только если Oracle не находит идентичного оператора SQL в совместно используемой области (библиотечном кэше) SGA.

Перед разбором оператора SQL Oracle просматривает библиотечный кэш в поисках идентичного оператора. Если обнаружено точное совпадение, нет необходимости в повторном разборе оператора. Но если идентичный оператор SQL не найден, Oracle проделывает все шаги, перечисленные выше.

Ключевым в последнем абзаце является слово «идентичный». Чтобы разделять одну область SQL, два оператора должны быть идентичными в полном смысле слова. Два оператора, которые похожи друг на

друга или возвращают одинаковые результаты, не обязательно идентичны. Чтобы считаться идентичными, два оператора должны удовлетворять следующим условиям:

- Иметь одинаковые символы с учетом регистра.
- Иметь одинаковые символы-разделители и символы перехода на новую строку.
- Ссылаться на одинаковые объекты, используя одинаковые названия, которые, в свою очередь, должны иметь одинаковых владельцев.

Если есть вероятность того, что ваше приложение будет несколько раз выполнять одинаковые (или похожие) операторы SQL, старайтесь всеми силами избегать ненужных разборов. Это повысит общую производительность приложения. Для уменьшения количества разборов можно применять следующие приемы:

- Использование связанных переменных.
- Использование псевдонимов таблиц.

## Использование связанных переменных

Когда приложение используют несколько пользователей, они фактически снова и снова выполняют один и тот же набор операторов SQL, только с разными значениями данных. Например, один сотрудник отдела обслуживания может выполнить такой запрос:

```
SELECT * FROM CUSTOMER WHERE CUST_NBR = 121;
```

а другой сотрудник того же отдела выполнит следующий запрос:

```
SELECT * FROM CUSTOMER WHERE CUST_NBR = 328;
```

Эти два оператора похожи, но не идентичны – значения идентификаторов клиентов не совпадают, поэтому Oracle должен выполнять разбор дважды.

Так как единственным отличием двух операторов является значение номера клиента, можно переписать приложение, используя связанные переменные. Тогда рассматриваемый оператор SQL мог бы выглядеть следующим образом:

```
SELECT * FROM CUSTOMER WHERE CUST_NBR = :X;
```

Такой оператор Oracle разбирался бы только один раз. Реальные номера клиентов подставлялись бы после разбора при выполнении оператора. Несколько параллельно работающих программ могли бы совместно использовать один экземпляр данного оператора SQL, подставляя различные номера клиентов.

В многопользовательских приложениях ситуации, подобные вышеописанной, встречаются очень часто, и использование связанных пе-

ременных может значительно увеличить общую производительность (за счет отказа от ненужных разборов).

## Использование псевдонимов таблиц

Использование псевдонимов таблиц может повысить эффективность выполнения операторов SQL. Прежде чем вдаваться в тонкости влияния псевдонимов таблиц на производительность, давайте вспомним, что представляют собой эти псевдонимы и как они используются.

При выборке данных из двух или более таблиц необходимо указывать, какой из таблиц принадлежит каждый столбец. Иначе, если две таблицы содержат столбцы с одинаковыми названиями, будет выдано сообщение об ошибке:

```
SELECT CUST_NBR, NAME, ORDER_NBR
FROM CUSTOMER, CUST_ORDER;
SELECT CUST_NBR, NAME, ORDER_NBR
       *
ERROR at line 1:
ORA-00918: column ambiguously defined
```

В данном случае ошибка вызвана тем, что и таблица CUSTOMER, и таблица CUST_ORDER содержат столбец CUST_NBR. Oracle не знает, на какой из них вы ссылаетесь. Чтобы исправить положение, можно переписать оператор следующим образом:

```
SELECT CUSTOMER.CUST_NBR, CUSTOMER.NAME, CUST_ORDER.ORDER_NBR
FROM CUSTOMER, CUST_ORDER
WHERE CUSTOMER.CUST_NBR = CUST_ORDER.CUST_NBR;

  CUST_NBR NAME                           ORDER_NBR
---------- ------------------------------ ----------
         1 Cooper Industries                    1001
         1 Cooper Industries                    1000
         5 Gentech Industries                   1002
         4 Flowtech Inc.                        1003
         4 Flowtech Inc.                        1004
         8 Zantech Inc.                         1005
         1 Cooper Industries                    1006
         5 Gentech Industries                   1007
         5 Gentech Industries                   1008
         1 Cooper Industries                    1009
         1 Cooper Industries                    1012
         1 Cooper Industries                    1011
         5 Gentech Industries                   1015
         4 Flowtech Inc.                        1017
         4 Flowtech Inc.                        1019
         8 Zantech Inc.                         1021
         1 Cooper Industries                    1023
         5 Gentech Industries                   1025
```

```
         5 Gentech Industries                1027
         1 Cooper Industries                 1029

20 rows selected.
```

Обратите внимание на использование названия таблицы для уточнения названия каждого столбца. Это устраняет двусмысленность, возникшую при ссылке на столбец CUST_NBR.

Вместо того чтобы указывать для столбцов полные названия таблиц, можно использовать псевдонимы, например:

```
SELECT C.CUST_NBR, C.NAME, O.ORDER_NBR
FROM CUSTOMER C, CUST_ORDER O
WHERE C.CUST_NBR = O.CUST_NBR;

  CUST_NBR NAME                           ORDER_NBR
---------- ------------------------------ ----------
         1 Cooper Industries                    1001
         1 Cooper Industries                    1000
         5 Gentech Industries                   1002
         4 Flowtech Inc.                        1003
         4 Flowtech Inc.                        1004
         8 Zantech Inc.                         1005
         1 Cooper Industries                    1006
         5 Gentech Industries                   1007
         5 Gentech Industries                   1008
         1 Cooper Industries                    1009
         1 Cooper Industries                    1012
         1 Cooper Industries                    1011
         5 Gentech Industries                   1015
         4 Flowtech Inc.                        1017
         4 Flowtech Inc.                        1019
         8 Zantech Inc.                         1021
         1 Cooper Industries                    1023
         5 Gentech Industries                   1025
         5 Gentech Industries                   1027
         1 Cooper Industries                    1029

20 rows selected.
```

Буквы «C» и «O» – это псевдонимы таблиц. Вы можете указать псевдонимы таблиц за соответствующими названиями таблиц в инструкции FROM, и затем они могут использоваться в запросе вместо названий таблиц. Псевдонимы таблиц представляют собой удобную краткую запись, что позволяет запросам быть более читабельными и лаконичными.

Длина псевдонимов таблиц не ограничена одним символом, она может достигать 30 символов.

Еще один факт, о котором необходимо помнить при использовании псевдонимов таблиц, заключается в том, что если в инструкции FROM

определены псевдонимы таблиц, далее следует использовать только эти псевдонимы, а не реальные названия таблиц. Если вы указываете для таблицы псевдоним, а затем используете в запросе фактическое название таблицы, то будет выдано сообщение об ошибке:

```
SELECT C.CUST_NBR, C.NAME, O.ORDER_NBR
FROM CUSTOMER C, CUST_ORDER O
WHERE CUSTOMER.CUST_NBR = CUST_ORDER.CUST_NBR;
WHERE CUSTOMER.CUST_NBR = CUST_ORDER.CUST_NBR
                                    *
ERROR at line 3:
ORA-00904: invalid column name
```

Столбец CUST_NBR присутствует в таблицах CUSTOMER и CUST_ORDER. Без необходимого уточнения столбец будет считаться неоднозначно определенным. Следовательно, необходимо указывать столбец CUST_NBR с псевдонимом таблицы (или с полным названием таблицы, если вы не применяете псевдонимы). Однако остальные два столбца, используемые запросом, определяются однозначно. Поэтому оператор, в котором уточнение приведено только для столбца CUST_NBR, вполне работоспособен:

```
SELECT C.CUST_NBR, NAME, ORDER_NBR
FROM CUSTOMER C, CUST_ORDER O
WHERE C.CUST_NBR = O.CUST_NBR;

  CUST_NBR NAME                           ORDER_NBR
---------- ------------------------------ ----------
         1 Cooper Industries                    1001
         1 Cooper Industries                    1000
         5 Gentech Industries                   1002
         4 Flowtech Inc.                        1003
         4 Flowtech Inc.                        1004
         8 Zantech Inc.                         1005
         1 Cooper Industries                    1006
         5 Gentech Industries                   1007
         5 Gentech Industries                   1008
         1 Cooper Industries                    1009
         1 Cooper Industries                    1012
         1 Cooper Industries                    1011
         5 Gentech Industries                   1015
         4 Flowtech Inc.                        1017
         4 Flowtech Inc.                        1019
         8 Zantech Inc.                         1021
         1 Cooper Industries                    1023
         5 Gentech Industries                   1025
         5 Gentech Industries                   1027
         1 Cooper Industries                    1029

20 rows selected.
```

Пришло время поговорить о псевдонимах таблиц с точки зрения производительности. Так как запрос не уточняет столбцы NAME и ORDER_NBR, Oracle, разбирая оператор, должен просматривать обе таблицы CUSTOMER и CUST_ORDER для того, чтобы определить, какой из таблиц принадлежит каждый из столбцов. Для одного запроса можно пренебречь временем, потраченным на такой поиск, но если предстоит разобрать множество подобных запросов, накопится значительное время. Хорошей программистской привычкой является указание псевдонимов таблиц для *всех* столбцов запроса, даже если они однозначно определены, чтобы избавить Oracle от дополнительного поиска при разборе оператора.

## Применяйте полностью определенный SQL для систем поддержки принятия решений

Мы только что говорили о преимуществах использования связанных переменных. Применение связанных переменных часто благотворно влияет на производительность. Но необходимо упомянуть и об одном недостатке. Связанные переменные скрывают от оптимизатора реальные значения. Такое утаивание может иметь и негативные последствия, особенно для производительности систем поддержки принятия решений. Рассмотрим, например, такой оператор:

```
SELECT * FROM CUSTOMER WHERE REGION_ID = :X
```

Оптимизатор может разобрать данный оператор, но не может принять в расчет реальный выбираемый регион. Если 90% клиентов проживает в регионе с номером 5, то при выборе таких клиентов наиболее эффективным подходом был бы полный просмотр таблицы. Для выбора клиентов других регионов больше подошел бы просмотр индекса. Когда вы жестко кодируете значения в операторах SQL, оптимизатор, оценивающий стоимость, может посмотреть на гистограммы (разновидность статистики) и сформировать план выполнения, который принимает в расчет конкретные значения. При использовании связанных переменных оптимизатор формирует план выполнения, не имея полной картины оператора SQL. Такой план может быть самым эффективным, а может и нет.

В системах поддержки принятия решений (DSS) очень редко случается так, что несколько пользователей снова и снова используют один и тот же запрос. Обычно небольшое количество пользователей выполняет различные сложные запросы к большой базе данных. Так как операторы SQL будут повторяться очень редко, время, сохраненное за счет использования связанных переменных, будет незначительным. В то же время, так как приложения DSS выполняют сложные запросы к большим базам данных, значительным может быть время, необходимое для выборки результирующих данных. Поэтому важно, чтобы оптимизатор формировал для запроса наиболее эффективный план вы-

полнения. Чтобы помочь оптимизатору создать такой план, необходимо снабдить его как можно большим количеством информации, в том числе реальными значениями столбцов или переменных. Поэтому в приложениях DSS следует использовать полностью определенные операторы SQL с жестко закодированными значениями вместо связанных переменных.

Данная ранее рекомендация по применению связанных переменных остается в силе для приложений OLTP (Online Transaction Processing – оперативная обработка транзакций). В системах OLTP несколько пользователей одновременно применяют одни и те же программы, выполняя одни и те же запросы. Объем данных, возвращаемых запросом, обычно достаточно мал. Поэтому время разбора является более важным фактором для производительности, чем в системах DSS. При разработке приложений OLTP используйте связанные переменные для того, чтобы сберечь время разбора и пространство разделяемой области SQL.