Предисловие

Впервые я встретился с реляционной СУБД Oracle примерно в 1988 или, может быть, в 1987 году, когда мой шеф бросил мне на стол небольшую коробочку и сказал что-то типа: “Мы получили новый продукт под названием Oracle. Поиграй с ним несколько недель и потом скажи, для чего он нам может пригодиться”.

Это была версия примерно 5.0.22, и в те дни овладеть Oracle было намного легче. Печатное руководство, включающее описание Forms 2.0, SQL*Report и все прочее, помещалось в небольшой портфель, а документация по оператору `create table` занимала около трех страниц.

Если вы заглянете в PDF-файл справочного руководства по SQL для версии 11.2, то обнаружите, что описание `create table` теперь начинается со страницы 16-6 и продолжается до страницы 16-79, занимая 74 страницы. Последний раз, когда я проверял общее количество страниц всех руководств по версии 9*i*, оно составило более 20 000 страниц, и я сомневаюсь, что эта цифра сократилась в версиях 10*g* и 11*g*.

Имея три (совсем тоненькие) руководства по 5.0.22, мне не пришлось потратить много времени, чтобы изучить все, что СУБД Oracle позволяла делать, и как все это делать эффективно. Опций было не так много, а потому было немного возможностей сделать что-то не так. Но как приступить к изучению этого продукта сейчас, когда ядро Oracle скрыто под огромной кучей опций и средств? Хуже того: детали, которые действительно нужно понять, заслоняют горы информации, которую неплохо бы иметь, но не критично для начала.

Ответ прост.

> *Шаг 1.* Читайте руководство по концепциям, чтобы понять, о чем вообще идет речь.
>
> *Шаг 2.* Читайте книгу Тома Кайта, чтобы пройти разумный путь изучения и экспериментирования, который проведет вас от первого запроса `select 'hello world' from dual` до того дня, когда вы уверенно сможете сказать что-то вроде “ вместо этой таблицы мы должны использовать секционированную по диапазонам индекс-таблицу с такими-то столбцами в сегменте переполнения, потому что и т.д.”.

В этой книге Том комбинирует три вещи: согласованный стиль, облегчающий восприятие технических деталей и понимание всех “почему”, стоящих за “как”; структурированную “сюжетную линию”, которая представляет собой последовательное изложение вместо беспорядочного набора случайных подсказок; и массив тщательно продуманных примеров, которые научат тому, как все работает, и как работать и думать самостоятельно.

Возьмем лишь один пример — индексацию. Существует много типов индексов, поэтому нам нужно краткое представление, чтобы различать их. Хорошо бы иметь представление о том, как на самом деле работают индексы со структурой B-дерева (например), чтобы понимать их сильные и слабые стороны. Затем можно переходить к идее индексов на основе функций — индексов по данным, которые “на самом деле не существуют”. Это подводит к пониманию того, что СУБД Oracle умеет делать, но мы можем (и будем) двигаться дальше — к пониманию того, что возможно делать с помощью СУБД Oracle.

Итак, мы видим, как собрать все кусочки вместе, чтобы создать индекс, гарантирующий уникальность среди подмножеств данных, мы видим, как можно — на огромном наборе данных — создать крошечный, дешевый в обслуживании индекс, который в точности идентифицирует данные для доступа и минимизирует риск генерации оптимизатором нерационального плана выполнения.

В принципе, все это есть в руководствах, но эффективно воспользоваться ими можно, лишь имея подсказку в виде простого описания доступных команд и способов применения их для конструирования решений реальных задач. Том Кайт предлагает эту подсказку, и затем стимулирует ваше самостоятельное мышление для дальнейшего движения.

Откровенно говоря, если каждый администратор базы данных и разработчик в мире тщательно проработает эту книгу Тома Кайта, мне, скорее всего, придется заняться консультированием по SQL Server, потому что количество клиентов, нуждающихся в консультациях по Oracle, резко снизится.

Джонатан Льюис

Предисловие из первого издания

"Думайте!" Когда в 1914 г. Томас Дж. Уотсон-старший (Thomas J. Watson, Sr.) пришел в компанию, которой суждено было стать корпорацией IBM, он принес с собой этот простой, состоящий из одного слова девиз. Это был призыв ко всем сотрудникам IBM, независимо от занимаемой должности, участвовать в принятии решений и творчески выполнять свою работу. Вскоре девиз "Думайте!" ("Think!") превратился в пиктограмму, которую можно было встретить в публикациях, календарях и лозунгах в кабинетах руководителей информационных и производственных подразделений, как в самой IBM, так и в других компаниях, и даже в карикатурах на страницах журнала *The New Yorker*. Девиз "Думайте!" был актуален в 1914 г. Он актуален и сегодня.

"Думайте о другом!" Несколько позже корпорация Apple Computer использовала этот лозунг в долговременной рекламной компании, призванной вдохнуть новую жизнь в марку компании и, что еще более важно, изменить отношение людей к технике в повседневной жизни. Вместо того чтобы призывать "думать иначе" ("think differently"), указывая на *способ* мышления, в лозунге компании Apple слово "different" ("другой") было использовано в качестве объекта глагола "think" ("думайте"), указывая на *предмет* размышлений (как в выражении "думайте о большем"). Рекламная компания была обращена к творческим людям. При этом предполагалось, что компьютеры Apple предоставляют уникальные возможности для разработки новаторских решений и творческих достижений.

Когда в далеком 1981 г. я поступил на работу в корпорацию Oracle (тогда она носила название Relational Software Incorporated), системы баз данных, построенные на основе реляционной модели, представляли собой новую, только зарождающуюся технологию. Разработчики, программисты и все увеличивающаяся группа администраторов баз данных учились искусству проектирования баз данных с применением методологии нормализации. Тогда еще малоизвестный язык SQL поражал своими возможностями манипулирования данными способом, для которого ранее требовалось трудоемкое процедурное программирование. В то время существовало множество тем для размышлений. Это утверждение справедливо и сегодня. Новые технологии вынуждали людей не только осваивать новые идеи и подходы, но и требовали от них нового образа мышления. Те, кто принял этот подход и придерживается его в настоящее время, добиваются наибольшего успеха в создании новаторских, эффективных решений производственных проблем, максимально используя технологию баз данных.

Возьмем, к примеру, язык баз данных SQL, коммерческая версия которого была представлена корпорацией Oracle. Язык SQL позволяет разработчикам приложений с помощью непроцедурного (или "декларативного") языка манипулировать наборами записей, а не создавать итерационные циклы в программах на обычных языках, которые обрабатывают по одной записи за раз. Когда я впервые познакомился с SQL, оказалось, что для понимания использования операций обработки наборов, таких как соединения и подзапросы, он требует "мышления под углом 45 градусов". Для большинства людей новой была не только идея обработки наборов, но и идея непроцедурного языка, когда указывается требуемый результат, а не способ его получения. Эта новая технология действительно требовала "думать иначе", одновременно предоставляя возможность "думать о другом".

Обработка наборов значительно эффективнее обработки по отдельности, поэтому приложения, которые полностью используют эти предоставляемые SQL возможности, обладают большей производительностью, нежели те, которые этого не делают. Тем не менее, просто удивительно, насколько часто приложения имеют производительность, далекую от оптимальной. Фактически, в большинстве случаев наибольшее влияние на общую производительность оказывает именно архитектура приложения, а не параметры настройки базы данных Oracle или другие параметры конфигурации. Поэтому разработчики приложений должны не только досконально изучить свойства баз данных и интерфейсы программирования, но и освоить новые подходы к использованию этих свойств и интерфейсов в своих приложениях.

В сообществе разработчиков и пользователей Oracle циркулирует множество "общепризнанных истин" по настройке системы для достижения максимальной производительности или для наиболее эффективного использования различных свойств Oracle. Иногда, благодаря разработчикам и администраторам баз данных, которые слепо принимают на веру эти концепции или развивают их без какого-либо внятного обоснования, эти "истины" становятся "фольклором" или даже "мифами".

Одним из примеров может служить концепция "если одно хорошо, то чем его больше — намного больше — тем лучше". Это утверждение популярно, однако оно редко оказывается справедливым. Например, рассмотрим интерфейс массива Oracle, который позволяет разработчику вставлять и перезаписывать несколько строк в ходе одного системного вызова. Понятно, что уменьшение количества сообщений, передаваемых по сети между приложением и базой данных — дело хорошее. Но если крепко подумать, то становится очевидным, что в определенный момент полезный эффект подобного ускорения будет сведен к нулю. Хотя загрузка одновременно 100 записей значительно предпочтительнее передачи записей по одной, в общем случае загрузка 1000 записей вместо 100 вообще не дает никакого выигрыша, особенно если учитывать предъявляемые при этом требования к памяти.

Еще один пример некритичного подхода — сосредоточение внимания на несовершенных аспектах архитектуры системы или конфигурации, а не на тех, которые, скорее всего, позволят повысить производительность (либо, скажем, надежность, доступность или безопасность). Возьмем, например, "общепринятую истину" о необходимости настройки системы для максимизации процента удачных обращений к буферу. Для некоторых приложений утверждение, что максимизация вероятности наличия нужных данных в памяти обеспечит наибольшую производительность, справедливо. Однако для большинства приложений лучше сосредоточить внимание на узких местах с точки зрения производительности (так называемых "состояниях ожидания"), а не на конкретных характеристиках системного уровня. Исключите те аспекты архитектуры приложения, которые вызывают задержки, и вы получите наивысшую производительность.

Я пришел к выводу, что разбиение проблемы на части и решение каждой из них по отдельности — прекрасный подход к проектированию приложения. Зачастую он позво-

ляет отыскать изящное и эффективное применение SQL для выполнения требований, предъявляемых к приложению. Часто оказывается, что в одном SQL-операторе можно выполнить действия, которые вначале казались невозможными без применения сложной программы на процедурном языке. Когда удается в полной мере использовать возможности SQL по одновременной, возможно параллельной, обработке наборов записей, это позволяет повысить не только производительность разработки, но и производительность самого приложения!

Иногда, по мере изменения обстоятельств, даже наилучшие методики, которые ранее были базовыми, становятся в той или иной степени неприменимыми. Например, рассмотрим старый афоризм "для достижения максимальной производительности индексы и данные нужно помещать в разные табличные пространства". Я часто встречал администраторов баз данных, которые неукоснительно придерживались этой концепции, не учитывая повышение скоростей работы и емкостей дисков или особенности конкретной нагрузки. Оценивая применимость этого конкретного "правила", необходимо учитывать *то*, что СУБД Oracle кэширует в памяти часто используемые и недавно использованные блоки (которые нередко принадлежат индексу), и *то*, что для выполнения каждого конкретного запроса она использует блоки индекса и данных последовательно, а не одновременно. Это ведет к тому, что операции ввода-вывода как индекса, так и данных в действительности должны распределяться между всеми одновременно работающими пользователями и между всеми доступными дисководами. Таким образом, блоки индекса и данных можно разделять по административным или личным соображениям, но никак не по соображениям производительности. (Том Кайт приводит ценные материалы по этой теме на веб-сайте Ask Tom по адресу `http://asktom.oracle.com`, где можно найти статьи по теме табличного пространства индексных данных.) Вывод из всего сказанного следующий: решения надлежит принимать не просто на основе реальных обстоятельств, а с учетом всех обстоятельств, имеющих место в данный момент.

Независимо от быстродействия используемых компьютеров и сложности базы данных, а также от возможностей средств программирования, ничто не может заменить человеческий разум в сочетании с "культурой мышления". Таким образом, хотя знание всех нюансов технологий, используемых в приложении, важно, еще важнее знание подходов к их правильному использованию.

Том Кайт — один из наиболее умных людей, которых я знаю, и один из наиболее знающих специалистов по базам данных Oracle, SQL, настройке производительности и проектированию приложений. Я совершенно уверен, что Том — ревностный приверженец девизов "Думайте!" и "Думайте о другом!". Со всей уверенностью можно утверждать, что Том полностью согласен с высказыванием "Дайте человеку рыбу, и вы обеспечите его пищей на один день. Научите человека рыбачить, и вы обеспечите его пищей на всю жизнь". Том получает удовольствие, делясь своими знаниями о СУБД Oracle, что идет на пользу всему обществу, но при этом он не просто дает ответы на отдельные вопросы, а учит думать и обосновывать свою точку зрения.

На своем сайте (`http://asktom.oracle.com`), в своих публичных выступлениях и в настоящей книге Том, помимо всего прочего, ненавязчиво призывает "думать иначе" в процессе проектирования приложений баз данных с помощью средств, предоставляемых базами данных Oracle. Он отказывается от общепринятых истин и точек зрения, призывая исходить из реальных фактов, подтвержденных примерами. При решении проблемы Том использует весьма прагматичный и простой подход, и следование его советам и методологии позволяет повысить производительность и разрабатывать более качественные и быстродействующие приложения.

Написанная Томом книга не только знакомит читателей со свойствами Oracle и со способами их использования, но и проповедует перечисленные ниже простые идеи.

- Не верьте мифам — во всем убеждайтесь сами.
- Не следуйте "общепринятой истине" — зачастую общеизвестные истины оказываются попросту ошибочными!
- Не верьте слухам или чужим мнениям — все проверяйте самостоятельно и принимайте решения на основе доказанных фактов.
- Разбивайте проблему на более простые, объединяя ответы, полученные на каждом из этапов, в изящное и эффективное решение.
- Не реализуйте в своих программах те действия, которые база данных может выполнить лучше и быстрее.
- Всегда помните о различиях между идеалом и реальностью.
- Задавайте вопросы и скептически относитесь к необоснованным "политикам компании" в отношении технических стандартов.
- Учитывайте общую картину, наиболее полно представляющую предъявляемые требования.
- Не жалейте время на то, чтобы ДУМАТЬ.

Том призывает относиться к базам данных Oracle не просто как к "черному ящику". Вместо того чтобы просто помещать данные в эти базы и извлекать их оттуда, Том поможет разобраться в работе СУБД Oracle и научит использовать ее возможности. Научившись творчески и сознательно применять технологию Oracle, вы сможете быстро и изящно решать большинство проблем проектирования приложений.

Я уверен, что в процессе чтения этой книги вы узнаете много нового о технологии баз данных Oracle и ознакомитесь с важными концепциями проектирования приложений. Уверен, что при этом вы начнете "думать иначе" о стоящих перед вами задачах.

Однажды Уотсон из IBM сказал: "От зарождения мира мысль была матерью всех достижений. Ситуация 'я не подумал' стоила миру миллионы долларов". И я, и Том согласны с этим высказыванием. Надеюсь, что вооруженные сведениями и технологиями, представленными в этой книге, вы сможете сэкономить для мира (или, по меньшей мере, для своей организации) миллионы долларов, получив при этом удовлетворение от хорошо выполненной работы.

Кен Якобс,
известный как "Доктор DBA"