Содержание

Благодарности

Я бы хотел выразить признательность всем тем, кто оказал мне помощь при создании этой книги. Я работал с лучшими в Oracle, и каждый из них тем или иным образом внес свой вклад в этот проект.

Во-первых, я хотел бы поблагодарить технических редакторов книги: Джонатана Левиса, Коннора Макдональда, Могенса Норгаарда, Энджи Колка и Марка Уильямса. Их проницательные, резкие и честные комментарии сделали эту книгу намного лучше.

Я благодарю моих сотрудников за оказанную мне поддержку в процессе написания книги. Эта книга заняла гораздо больше времени и сил, чем я ожидал, и я высоко ценю их за проявленную терпимость.

Я также признателен пользователям Oracle. Большую часть материала, изложенного в книге, я получил благодаря их замечательным запросам. В этой книге были рассмотрены многие из вопросов пользователей.

И наконец, я бы хотел выразить признательность моей семье за постоянную поддержку. Я не уверен, что без энтузиазма моей жены Лори, сына Алана и дочери Меган смог бы когда-нибудь закончить эту работу.

Введение

Эта книга предназначена для всей команды разработчиков Oracle — группы людей, осуществляющих стопроцентный контроль над работой системы. В состав этой команды входят создатели модели данных, разработчики и штат администраторов базы данных. Подобный взгляд противоречит известному мифу о том, что администраторы базы данных полностью контролируют и отвечают за все аспекты работы приложений. Проведем аналогию с автогонками для того, чтобы понять ошибочность этого старого представления. Администратор базы данных (DBA) — это рабочий, меняющий колеса, проверяющий работу двигателя и другие параметры машины. Если пригнать этому рабочему (DBA) Lincoln Navigator (большой грузовик) и сказать, что он будет принимать участие в гонках с Indy 500, то что произойдет? Безусловно, администратор все проверит и удостоверится, что колеса могут двигаться максимально быстро, но он не сможет ничего сделать для того, чтобы машина ехала со скоростью, большей, чем 100 миль в час. В действительности после того, как машина была спроектирована и создана, администратор не так уж много может сделать (а теперь замените машину приложением).

Эта книга предназначена для разработчиков, которым необходимы более совершенные методы проектирования и построения масштабируемых систем с использованием платформы Oracle. Эта книга не для начинающих. Она рассчитана на разработчиков, которые знают, как вводить SQL-запросы, используя SQL Plus, и т.п. Эта книга не учит SQL — она рассказывает о том, что необходимо знать для создания «грамотных» SQL-запросов. Она не учит, как создавать код приложений, но рассказывает о том, что необходимо знать для написания «грамотных» приложений, основанных на Oracle.

Воспользуюсь еще одной аналогией для того, чтобы объяснить, как представлена информация в этой книге. Допустим, что разработчик — это доктор, а приложение — пациент. Пациенту оказывают помощь разные врачи:

- **Врач скорой помощи (ER)** Этот доктор производит так называемую «сортировку», отделяя безнадежных пациентов от тех, кому еще можно помочь, оказывает экстренную помощь, стремясь сделать все возможное для того, чтобы пациент остался жив. Он пыта-

ется вернуть к жизни больных, перенесших сердечный приступ, который произошел из-за того, что они долго курили, не соблюдали диету и не выполняли физических упражнений.

- **Хирург (OR)** Этот доктор получает пациентов после того, как врач скорой помощи оказал им первую помощь. Хирург пытается не только поддержать жизнь пациента, но и вернуть ему здоровье настолько, насколько это возможно. Он делает операцию по шунтированию сердца у больных, перенесших сердечный приступ, пытаясь очистить сосуды.
- **Физиотерапевт (PT)** Пациент попадает к физиотерапевту в послеоперационный период для реабилитации. Процесс восстановления здоровья обычно бывает довольно долгим и болезненным (не говоря уж о том, что и дорогостоящим).
- **Врачи профилактической медицины** Эти специалисты стараются делать все, чтобы пациент не попал в ситуацию, требующую вмешательства трех предыдущих врачей. Они советуют больному умеренно курить, соблюдать диету, выполнять физические упражнения. Кроме того, они разрабатывают специальную пошаговую программу для поддержания хорошей физической формы пациента. Если пациент будет выполнять все их рекомендации, он никогда не попадет ни к одному из трех предыдущих врачей (несчастные случаи, например автомобильная авария, в расчет не берутся).

Безусловно, все врачи необходимы — мало ли что может произойти. Но, вероятно, самым важным и необходимым является врач профилактической медицины. Только он пытается сделать все возможное для того, чтобы пациенту не пришлось обращаться к трем другим докторам.

По моему опыту, многие ориентируются на три первых типа докторов. Они живут с верой в «героя»-разработчика — доктора скорой помощи или хирурга. Возможно, именно по этой причине хорошая проработка и внедрение проекта кажутся неблагодарным занятием. Врачи скорой помощи и хирурги получают всю славу, спасая, буквально вырывая пациента из рук смерти (сохраняют систему, совершая чудо). Их вызывают в экстренных случаях, и в последнюю минуту они спасают жизнь пациенту, получая за это достойную плату. Физиотерапевты — это несчастные люди, которые получают систему после OR/ER-врачей, латают ее и становятся ответственными за ее работу.

Я вполне мог бы написать эту книгу с точки зрения доктора скорой помощи, поскольку, по сути, отношусь к тем самым «героям», которых часто называют «завоевателями» систем. Я действительно *мог бы* так поступить, но не стал.

Ведь подобный подход однозначно упускает возможность воспользоваться советами доктора профилактической медицины. Существует несколько книг, авторы которых предлагают именно профилактику. Это мои любимые книги: Guy Harrison "*Oracle SQL High Performance Tuning*", 2nd Edition (Prentice Hall, 2001) и Jonathan Lewis "*Practical Oracle 8i Building Efficient Databases*" (Addison Wesley, 2001). Эти книги, включая и мою собственную "*Expert One on One Oracle*" (Wrox Press, 2001), написаны для того, чтобы постараться исключить потребность в «героях». Запомните, что

пожарные — это те герои, услугами которых каждый из нас надеется никогда не воспользоваться.

Эта книга является руководством для тех, кто создает структуру и настаивает систему. В ней обсуждаются следующие вопросы:

- Настройка перед началом проектирования
- Проектирование с учетом определенных целей и проверка их реализации
- Испытание системы и исключение ошибок (необходимо убедиться в том, что каждый элемент работает надлежащим образом, поскольку многие проблемы являются следствием незнания того, как же в действительности работает база данных)
- Ошибки в процессе реальной работы. Рассматривается также роль ER/OR-докторов, но это не является главной целью книги. Знания, получаемые врачом профилактической медицины, должны помочь ограничить более яркие роли остальных докторов.

Большинство книг в основном сосредотачиваются на том, как быть «героем». Эта же книга делает акцент на том, как стать хорошим разработчиком. Вместо того чтобы обсуждать вопросы исправления некорректно работающей системы, в этой книге рассматриваются возможности построения стабильной системы. Ведь, в конце концов, тот, кому удастся создать стабильно работающую систему, и будет настоящим героем.

ГЛАВА 1

ПРАВИЛЬНЫЙ ПОДХОД К СОЗДАНИЮ ПРИЛОЖЕНИЙ

В этой главе будут рассмотрены проблемы, которые часто встречаются при работе с базой данных Oracle. Они касаются проектирования системы, ее разработки, тестирования, внедрения и поддержки. Подобные проблемы встречаются при работе не только с Oracle, но и с любой другой программной системой. Возможно, эта глава меньше всех остальных глав посвящена технической стороне вопроса, но тем не менее она является одной из самых важных в этой книге. Дело в том, что нередко встречаются ошибки, которые по своей природе являются в большей степени ошибками организации процесса работы, нежели техническими ошибками. Приведем несколько примеров:

- Тенденция непонимания между разработчиками и администраторами базы данных (DBA). В этой главе рассматриваются возможные последствия подобного непонимания и даются советы, как сделать взаимоотношения более продуктивными.
- Точка зрения, что создание тестовой среды — это слишком дорого. На самом деле, отсутствие тестовой среды в дальнейшем может обернуться гораздо большими затратами.
- Неполная эксплуатация возможностей базы данных. Это может быть обусловлено непониманием или желанием не попадать в зависимость от применения базы данных, или же это может быть связано с боязнью, неуверенностью и сомнениями. В любом случае часто используются приложения, работа с которыми не приносит ожидаемого результата.

Все эти вопросы рассматриваются в данной главе.

Работа в команде

Работа в команде не имеет ничего общего с технологией или программным обеспечением. Это только взаимодействие людей. Обратите внима-

ние, что большинство проблем, возникающих в процессе разработки программного обеспечения, по большей части связаны с политическими аспектами, а не с технологией. Сколько раз я наблюдал, как все труды разработчиков сводились на нет не технической сложностью поставленной задачи, а проблемами, связанными с распределением ресурсов, с организацией рабочего процесса и с политикой.

Нередко отношения между членами команды разработчиков и команды администраторов базы данных (DBA), которые должны оказывать поддержку друг другу, строятся по принципу «мы против них». В подобных условиях администраторы считают необходимым защитить базу данных от разработчиков. С другой стороны, разработчики считают, что администраторы им мешают. Порой, пытаясь заставить трудиться эти две группы вместе, я ощущаю себя в большей степени брачным адвокатом, чем экспертом базы данных.

Мы должны всегда помнить, что работа в команде подобна улице с двухсторонним движением. Разработчики часто жалуются, что администраторы требуют от них слишком много объяснений по поводу предоставления права доступа или разрешений. В действительности, это разумное требование. Раздача прав доступа заслуживает внимательного и обдуманного подхода. Нет ничего неправильного в том, что администратор может попросить разработчика, чтобы тот объяснил необходимость, например, предоставления его учетной записи права CREATE VIEW (см. ниже). Но, с другой стороны, крайне неразумно запрещать использование таких средств базы данных, как представления базы данных, хранимые процедуры и триггеры.

Ниже приведен один из примеров подобных отношений (он взят с сайта AskTom):

> «Использование хранимых процедур — плохо ли это?
>
> В чем заключается отрицательная сторона применения хранимых процедур? Вызывает ли это какие-то побочные явления? Я новичок в администрировании базы данных Oracle. Один из программистов SQL попросил меня предоставить ему системные права на CREATE PROCEDURE...»
>
> В своем ответе я объяснил несомненные выгоды от использования хранимых процедур, указал администратору, каким образом они могут помочь ему в настройке приложений (без детального изучения работы приложения). Я подчеркнул, что хранимые процедуры являются механизмом безопасности и позволяют увеличить производительность. Я посоветовал администратору разрешить использование хранимых процедур, поскольку, размещая «плохой» SQL в клиентских приложениях, разработчики могут доставить гораздо больше проблем. К тому же я указал, что размещение SQL-операторов в хранимых процедурах дает администраторам возможность контролировать разработчиков.
>
> Последовавшие ответы были довольно агрессивны и лишний раз демонстрировали наличие пропасти в отношениях между разработчиками и администраторами базы данных. Точку зрения администраторов можно описать следующим образом: «Моя работа заключается в защите базы данных». В то время как разработчики

> считают: «Моя работа заключается в написании программ, а их задача — позволить мне делать это». Один разработчик высказал мнение, что в обязанности администраторов входит также проверка каждой строчки кода, который используется в базе данных. А администратор заявил, что это дело разработчика.
>
> В действительности, это непосильная для администратора задача — просмотреть каждую строчку кода в базе данных. Более того, администратор — не разработчик, и он не сможет понять, о чем идет речь, даже если просмотрит исходный код.

Очевидно, что разработчики и администраторы выполняют различные задачи. Работа администратора заключается не только в защите базы данных, и он не должен заниматься обслуживанием разработчика.

Подобно чрезмерно заботливому родителю, работа администратора, который придерживается позиции «защитить базу данных от разработчика», не будет продуктивна. С другой стороны, разработчики не программируют назло администраторам (вопреки расхожему мнению администраторов). Разработчики всего лишь пытаются сделать свое дело. Их основная задача — построение функционального приложения, удовлетворяющего требованиям конечного пользователя, производительного, масштабируемого, удобного для эксплуатации и разработанного в разумных стоимостных пределах. До тех пор пока эти две команды не согласятся вместе идти к конечной цели, шансы на успех будут невелики. Пока между ними существует виртуальная стена, многие из задач не смогут быть реализованы.

Роли администраторов базы данных и разработчиков

Обычно администраторы базы данных знают о самой базе и ее работе больше разработчиков, в то время как разработчики знают больше администраторов о том, как создавать программное обеспечение. Это общее понимание их должностных обязанностей.

Администраторы должны разбираться в архитектуре базы данных, знать, как и когда установить обновление и как работает база данных. Они не смогут успешно выполнить процедуру резервного копирования и восстановления данных (например, если их этому не обучали) без глубокого понимания и постоянного изучения архитектуры базы данных Oracle. Если администратор не понимает архитектурных взаимоотношений между управляющими файлами базы данных, файлами с данными и журналами базы данных, он будет совершать ошибки при резервном копировании и восстановлении. Жизнь администратора тесно связана с работой базы данных.

Разработчики являются в основном программистами или аналитиками, которые воспринимают базу данных как инструмент, используемый для достижения очередной цели. В большинстве случаев они посвящают много времени работе, не связанной с базой данных, такой как создание интерфейса конечного пользователя.

И тем не менее есть точки пересечения. Если две эти команды будут трудиться сообща, то разработчики будут знать больше о работе базы данных, а администраторы смогут помочь им в процессе проектирования.

Я составил два списка того, что должны и чего не должны делать разработчики и администраторы баз данных. Это поможет им закрыть вопрос взаимодействия раз и навсегда.

Что должны и чего не должны делать администраторы

Администраторы! Не считайте, что ваша основная цель — защитить базу данных от разработчиков. База данных является их инструментом, и вы должны помогать и советовать им, как лучше ее использовать. Не считайте разработчика вашим противником.

Кроме того, не отказывайтесь от использования тех или иных средств без весомых на то причин. Очень часто подобные решения основаны на опасении и неуверенности или же на однократном неудачном опыте применения. Рассмотрим общие ограничения:

- Не разрешены представления. Причиной такого ограничения может стать единичный негативный опыт работы администратора с представлением. В результате чего объявляется, что использование представлений недопустимо. Но в таком случае можно и SQL объявить вне закона, если вы столкнетесь с плохо организованным запросом.
- Не разрешены хранимые процедуры. Это действительно удивляет, поскольку использование хранимых процедур является оптимальным вариантом работы для администратора. В этом случае администратор будет точно знать, какой модуль зависит от какого объекта базы данных. Будет легче настроить неправильно выполняющийся модуль, если проблема связана с SQL (нужно лишь прочитать код из словаря данных и выполнить его с применением SQL). Предположим, что администраторам базы данных необходимо настроить приложение с помощью Java 2 Enterprise Edition (J2EE). Так как администраторы не являются разработчиками и не могут программировать с помощью J2EE, они даже не будут знать, с чего им начать. Если часть приложения находится в базе данных, то настройка доступа к базе данных становится намного легче.
- Не разрешено добавление новых средств, появившихся после версии 6. Это ограничение очень распространено среди администраторов-ветеранов, которые осторожно относятся ко всему новому: к PL/SQL, триггерам, контекстам приложений, проверке на уровне структурных единиц, контролю доступа к структурным единицам и т.п. Они хотят запретить чтение в базе данных (если бы они еще могли объявить вне закона соединения!). Я подозреваю, что так себя ведут администраторы, не желающие ни за что отвечать. Работа администратора становится незначительной в том случае, если разработчики применяют язык манипулирования данными (DML). Но компания в данном случае много теряет, поскольку платит большие деньги за работу команды базы данных, но не использует ее в полном объеме.
- Не принимаются новые возможности для версии N. Идея, которая лежит в основе подобного поведения, следующая: пусть другие решат основные проблемы, работая с новыми возможностями, мы же начнем использовать эти средства через три или четыре года. Но за это время потери могут составить гораздо больше, так как исполь-

зование новых возможностей помогло бы сэкономить массу времени и усилий (а в дальнейшем и денег). Локально управляемые табличные пространства (представленные в Oracle 8i, выпуск 1) — классический тому пример. Применение этого средства выгодно, однако многие администраторы базы данных объявили его вне закона. Причина этого имеет вовсе не технический характер, она вызвана боязнью, неуверенностью и сомнениями.

«Я прочитал о преимуществах использования локально управляемых табличных пространств и спросил моего старшего администратора базы данных, не могли бы мы изменить табличные пространства новых хранилищ данных на локально управляемые. Он ответил, что при работе с локально управляемыми пространствами возникают проблемы с производительностью...»

Администратор прав, проблемы с производительностью есть, но они обусловлены как раз теми табличными пространствами, которые использует администратор (управляемые словарем). Отсутствие корректной информации ведет к отрицанию новых возможностей.

С другой стороны, не стоит увлекаться новыми средствами, даже если они кажутся уникальными. Возьмем, к примеру, применение расширенного языка разметки (XML) в базе данных. Вовсе не означает, что все должно быть сохранено в XML.

Теперь представим список того, что администратор базы данных должен делать:

- Рассматривайте разработчика как человека, которого вы можете научить и которому вы можете передать ваши знания о базе данных. Через какое-то время вы с удивлением заметите, что разработчик действительно начинает поступать правильно. Если показать людям верный путь, а еще лучше, научить их искать правильный путь самостоятельно, то они будут пользоваться этими знаниями. Люди хотят поступать правильно, но они лишены информации. Распространяйте ваши знания.
- Оценивайте и тестируйте новые возможности по мере их появления. Не проходите мимо них. Не руководствуйтесь единичным негативным опытом применения нового средства. Каждое средство предназначено для определенных целей. Возможно, что негативный опыт оказался результатом неподходящего применения той или иной технологии. Молоток является хорошим инструментом для забивания гвоздей, но абсолютно непригоден для закручивания шурупов.
- Используйте методики и процедуры, основываясь на реальных фактах. Не занимайте позицию: «Я слышал, что они очень медленно работают» или «Я слышал, что при работе возникает много ошибок». Слухи допустимы во дворе, но абсолютно не приемлемы в работе. Это подобно вере в миф: «Если вы достигли 99.9% удачных обращений к кэшу, то ваша работа выполнена» или «Таблица должна находиться в одном экстенте». В этой книге я придерживаюсь реальных фактов. И вам следует поступать также.

Что должны и чего не должны делать разработчики

- Не пытайтесь работать рядом с администраторами базы данных, работайте вместе с ними. Если вы предоставите верную и неоспоримую аргументацию, то они будут вас слушать. Администраторы начнут блокировать вас в том случае, если вы попытаетесь что-то сделать без их ведома. Ведь вы становитесь слишком самостоятельны, а они этого боятся.
- Не считайте, что администраторы работают против вас. В большинстве случаев есть причины на то, чтобы размещать методы и процедуры в определенных местах. Работайте над изменением правил, с которыми вы не согласны, но не пытайтесь нарушать их.
- Просите администраторов, чтобы они объясняли вам свои доводы. Если вы предлагаете использовать локально управляемые табличные пространства, а ваш главный администратор базы данных говорит: «Нет, потому что есть проблемы с производительностью», попросите его предоставить вам информацию, связанную с производительностью. Объясните, что вы хотели бы изучить и понять, почему работа с локально управляемыми табличными пространствами дает такие плохие характеристики производительности (на самом деле это не так).
- Вы должны знать то, о чем говорите. В противном случае вы немедленно утратите доверие. Используйте научный метод: начните работу с гипотезы. Протестируйте гипотезу, чтобы подтвердить или опровергнуть ее. Убедитесь, что этого теста достаточно для подтверждения вашей точки зрения. Попробуйте воспроизвести ее по-другому. Подготовьте результат и критически осмыслите его. При соблюдении этих условий вы будете твердо стоять на ногах, доказывая правильность вашего подхода.

Администраторы базы данных и разработчики должны трудиться вместе как единая команда. Они должны встречаться и регулярно обсуждать все вопросы. Они не должны считать себя двумя независимыми подразделениями, перебрасывающими друг другу работу. Подход «они против нас» сделает невозможным создание быстрой, надежной и доступной системы. На мой взгляд, имеет смысл работать единой командой, используя специфические знания и обмениваясь ими друг с другом.

Чтение документации

Документация к базе данных Oracle насчитывает более 100 руководств (108 вместе с Oracle 9*i* выпуска 2). Это свыше 46 000 страниц текста. Такой объем информации запугает кого угодно и отобьет охоту к изучению. В действительности, люди больше симулируют испуг, чем боятся на самом деле. Необходимо лишь определиться, какая информация нужна, и найти требуемый раздел. Для начала выясним, какие части документации являются ключевыми.

> «Я изучил большой объем информации на вашем сайте. Один из моментов, о котором вы постоянно говорите, — это чтение общего руководства (Concepts guide). Пару недель назад я скопировал

на мой портативный компьютер PDF-версию этого руководства и начал его изучать. Я очень рад, что поступил так! Многие ранее непонятные вопросы теперь стали ясны. Я работаю консультантом разработчиков и администраторов базы данных и теперь знаю больше них».

Это лучшая награда для меня. Я часто получаю вопросы, начинающиеся со слов «Пожалуйста, не отсылайте меня к документации». Подобные просьбы я игнорирую, если ответ на вопрос хорошо изложен в документации. Нередко я начинаю свой ответ со слов: «Итак, решение своей проблемы вы найдете в *общем руководстве* (Concepts guide) в разделе...» Прочитайте *общее руководство* от корки до корки. Пусть даже в голове останется только 10% прочитанного, но знаний о базе данных Oracle и ее работе станет на 90% больше, чем у тех, кто не обращался к этой книге. И в дальнейшем, если вдруг возникнет какая-то проблема, специалист просто скажет себе: «Я помню, что читал об этом. Поищу решение в *общем руководстве*».

Руководство к руководствам

Несмотря на то, что существует более 100 книг с документацией, руководство к руководствам получилось на удивление сжатым. Руководства, рассмотренные ниже, — это та документация, которую следует прочитать от корки до корки (независимо от принадлежности к группе разработчиков или администраторов базы данных). К остальной документации можно обращаться по необходимости.

Общее руководство

Общее руководство (Concepts guide) — это та часть документации Oracle, которую необходимо читать при появлении каждого нового выпуска базы данных. Она содержит большой объем информации по следующим разделам:

- **Что такое Oracle?** Введение в базу данных, структура памяти, распределенная база данных, управление параллельным доступом, непротиворечивость данных, безопасность, администрирование и другие вопросы.
- **Структура базы данных** Углубленный взгляд на хранение данных. Кто, когда, где, зачем и почему разбивал на блоки, работал с экстентом и табличными пространствами, сегментировал базу данных.
- **Экземпляр Oracle** Что собой представляет экземпляр Oracle: каковы процессы запуска и остановки, как приложения взаимодействуют с Oracle, как выглядит архитектура памяти и процесса и как управлять ресурсами базы данных.
- **Данные** Краткий обзор множества объектов схемы (таких, как таблицы, представления, индексы и т.п.), все опции, доступные для каждого типа объектов, все типы данных: и предопределенные в базе данных, и определяемые пользователем.

- **Доступ к данным** SQL, PL/SQL, взаимодействие Java с базой данных, управление и поддержка зависимостей между объектами схемы, триггеры и управление транзакциями.
- **Параллельные операции** Как и когда применяются параллельные операции. Параллельные запросы, параллельный DML, параллельные операции для администрирования и т.д.
- **Защита данных** Возможно, это один из самых главных разделов. В нем рассматриваются такие темы, как согласование и совмещение работы в Oracle, целостность данных, безопасность и средства защиты данных. Ряд тем посвящен использованию привилегий и ролей, а также проведению проверки в случае появления проблемы.

Одним из достоинств общего руководства (Concepts guide) (помимо того, что оно бесплатно) является то, что оно содержит указания и ссылки на остальную документацию. Очень часто тема заканчивается словами «Смотрите также» и ссылками на документацию Oracle, где вопрос излагается более подробно. Таким образом, общее руководство (Concepts guide) — это книга, занимающая верхний уровень в описании возможностей Oracle и содержащая метаданные для остальной документации. После прочтения общего руководства пользователь сможет плавно перейти к изучению необходимой ему информации, углубляясь в ту или иную тему.

Руководство по новым возможностям

Руководство по новым возможностям (New features guide) — это обзор новых средств, вошедших в пару последних выпусков. В руководстве приводится краткое описание каждого нового средства с указателем на документацию, в которой содержатся более подробные сведения. В нем также дается перечень того, что включено в каждый вариант из семейства Oracle (Персональный, Стандартный, Корпоративный). В Oracle 8*i* и более ранних версиях это руководство называлось *Getting to Know* (Получение знаний).

Во многих документах содержится глава «Что нового?». В *руководстве по новым возможностям* (New features guide) рассматривается большинство новых средств, относящихся к администрированию, разработке, производительности, масштабируемости, работоспособности и т.п. Конкретный же документ содержит более подробный список «Что нового?». Например, в руководстве по администрированию (Administrators guide) будет содержаться глава «Что нового в администрировании?», а в руководстве для разработчиков приложений (Application developers guide) будет глава «Что нового в разработке приложений?».

Руководство для разработчиков приложений

Существует несколько руководств по различным средствам базы данных Oracle, начинающихся с *руководства для разработчиков приложений* (Application developers guide). К таким средствам относятся, например, организация очередей (Advanced queuing), LOB (большие объекты), особенности реляционных объектов (Object relational features) и управление рабочей областью (Workspace management). Кроме того, всем разработ-

чикам рекомендуется прочитать *руководство по основным принципам* (Fundamentals guide). В нем рассматриваются такие вопросы, как:

- Программные среды, к которым можно получить доступ
- Проектирование схемы базы данных
- Поддержание целостности данных посредством ограничений
- Индексирование данных: что учесть, как SQL-операторы будут обрабатываться компьютером, использование динамического SQL, применение PL/SQL, реализация безопасности и т.д.

Общее руководство (Concepts guide) рассказывает о новых средствах, в то время как *руководство для разработчиков приложений* (Application developers guide) объясняет, как этими средствами пользоваться.

Справочное руководство пользователя PL/SQL

PL/SQL является одним из наиболее важных языков для разработчиков, поскольку в процессе проектирования разработчик получает возможность доступа к базе данных. *Справочное руководство пользователя PL/SQL* (PL/SQL users guide and reference) рассматривает основные принципы PL/SQL, ошибки обработки, синтаксис, пакеты/процедуры и многое другое.

Справочное руководство по настройке производительности

Одним из наиболее часто используемых документов является *справочное руководство по настройке производительности* (в Oracle 8*i* и более ранних версиях оно называлось «*Руководство по проектированию и настройке производительности*» (Designing and tuning for performance guide)). Первая часть руководства обращена к разработчикам. В ней рассказывается о том, как оптимизировать работу, как корректно собрать статистику, как работают различные физические структуры и где их лучше использовать (например, в каком случае следует применять индексные таблицы). В этом документе описаны основные средства, которыми пользуется разработчик: Explain Plan, SQL_TRACE, TKPROF, Autotrace и Statspack (пакет сбора статистики)!

Вторая часть руководства предназначена для администраторов. В ней рассматриваются темы, посвященные построению производительной базы данных, конфигурации памяти, взаимодействию с операционной системой и использованию ресурсов, конфигурированию общих и специализированных серверов, вопросам сбора статистики и применения производительных представлений и других средств. Все это должно быть прочитано перед началом настройки и вводом в действие. Даже если администратор проработал с Oracle уже 100 лет, все равно он найдет для себя что-то новое и полезное в этом руководстве.

Концепция дублирования и восстановления

Дублирование и восстановление — это те функции, нарушения работы которых администратор не должен допустить. Даже если используются средства автоматического дублирования, все равно необходимо прочитать *руководство по концепции дублирования и восстановления* (Backup and recovery concepts). Четкое и глубокое понимание процессов дублирования и восстановления никогда не будет лишним.

Но только чтения этого руководства будет недостаточно. Нередко поступают вопросы от людей, которые прочитали это руководство, но не понимают, почему им не удается провести восстановление после возвращения в исходное состояние управляющих файлов прошлой недели. У них отсутствуют базовые знания о том, как взаимодействуют файлы и что необходимо для восстановления в данной ситуации. Администратор должен прочитать *руководство по концепции дублирования и восстановления* (Backup and recovery concepts). Если не все понятно с первого раза, необходимо читать снова и снова. Затем нужно проверить свои знания. И если вы не можете даже найти день обновления, то ваши знания ничего не стоят.

Руководство по управлению восстановлением

В *руководстве по управлению восстановлением* (Recovery manager reference) рассказывается о средствах, которые используются для резервного копирования базы данных. Можно забыть о старых сценариях и распрощаться с tar, cpio, dd и ocopy. В этом руководстве рассматриваются возможности восстановления на уровне блоков, своевременное восстановление, дублирование сохраняемых методов, быстрое резервирование и другие средства. Оно заслуживает вашего внимания.

Руководство для администратора

В *руководстве для администраторов* (Administrators guide) всегда можно найти что-то новое. Из этой книги можно узнать о том, что в составе базы данных есть менеджер ресурсов (появившийся в версии 8*i* и усовершенствованный в версии 9*i*), а также о том, как настроить аудит (появился в Oracle 9*i*).

Новые возможности Oracle, касающиеся администрирования базы данных, содержатся только в этом документе, так как они не настолько общие, чтобы размещать их в *руководстве по новым возможностям* (New features guide).

Путеводитель по чтению

Путеводитель — это документация, которую обязательно нужно прочитать. Я предлагаю три набора: один общий, второй для тех, кто считает себя администратором, и третий для тех, кто считает себя разработчиком.

Книги, необходимые и администраторам, и разработчикам

При выходе каждой новой версии базы данных администраторы и разработчики обязаны прочитать:

- *Общее руководство* (Concepts guide)
- *Руководство по новым возможностям* (New features guide)

Книги, необходимые разработчикам

Разработчики должны также прочитать следующую документацию:

- *Руководство для разработчиков приложений* (Основы) (Application developers guide (Fundamentals))
- *Справочное руководство пользователя PL/SQL* (PL/SQL users guide and reference)

- *Справочное руководство по настройке производительности* (Performance tuning guide and reference) (Руководство по проектированию и настройке производительности в версии 8*i* и более ранних)

Разработчикам следует прочитать первую часть справочного *руководства по настройке производительности* и просмотреть на досуге остальные главы. В дополнение к руководству по Oracle 9*i* я предлагаю просмотреть справочник *Performance Method* (9*i*, выпуск 1) или *Performance Planning* (9*i*, выпуск 2). В этом руководстве, всего на 60-ти страницах, представлена информация, которая необходима для успешной работы по таким темам, как масштабируемость, архитектура системы, принципы проектирования приложения и т.п.

Книги, необходимые администраторам

Администраторы должны также прочитать следующие книги:

- *Концепция дублирования и восстановления* (Backup and recovery concepts)
- *Руководство по управлению восстановлением* (Recovery manager reference)
- *Концепция дублирования и восстановления* (Нет, это не опечатка. Я действительно поместил это руководство здесь дважды, поскольку это тот вопрос, в котором администратор не имеет права ошибаться. И это тот вопрос, в котором администраторы ошибаются особенно часто. Это руководство необходимо прочитать и понять.)
- *Руководство для администратора* (Administrators guide)
- *Справочное руководство по настройке производительности* (Performance tuning guide and reference)

Книги, рекомендуемые для прочтения

После прочтения руководств, рекомендованных выше, можно перейти к специализированной документации. Например, разработчик, которому необходимы знания по XML, может найти не менее трех руководств по XML. Для тех, кого интересуют возможности Java, существуют руководства по этой теме. Если же администратору базы данных необходимо понять, каким образом установить и сконфигурировать стабильно работающую среду, он также сможет воспользоваться соответствующей документацией.

Если глубоко и подробно изучать документацию, всегда можно найти полезные вещи. Я говорю об этом вовсе не потому, что работаю в Oracle. Просто на сегодняшний день большинство моих знаний об Oracle я почерпнул именно из документов. Я читаю *общее руководство* (Concept guide) для пополнения своих знаний. Настоятельно рекомендую всем обращаться по адресу в Интернете http://otn.oracle.com и работать со ссылками на документацию. Там можно найти абсолютно все.

Избегайте синдрома черного ящика

Без фундаментальных знаний базы данных Oracle и ее функционирования разработчики неизбежно будут совершать ошибки. Долгое время

базу данных считали черным ящиком — таким же заменяемым предметом потребления, как, например, аккумуляторы в радио. В соответствии с этим подходом пользователь базы данных любой ценой старается избежать зависимости от нее и поэтому отказывается от эксплуатации новых возможностей. Получается, что на деле большинство функций не применяются. Это означает, что пользователь идет по пути «делать все самому», т.е. пишет больше кода, чем это необходимо для работы, и теряет время.

Компании, выбравшей такой путь, будет не легко. Это стоит денег как с точки зрения оплаты более длительного процесса разработки, так и с точки зрения упущенных возможностей.

Выбор между независимостью и зависимостью от базы данных

Многие могут поспорить, но тем не менее: зависимость от базы данных (не имеет значения, какая база данных используется, речь идет не только об Oracle) должна быть целью, к которой нужно стремиться, а не избегать ее. Если руководство предприятия считает возможным вкладывать в базу данных денежные средства и хочет, чтобы программное обеспечение разрабатывалось за минимально короткое время, то единственный путь — это полностью использовать все возможности базы данных.

Дело в том, что, если не считать самых простых приложений, независимость от базы данных — удовольствие чрезвычайно дорогое и поглощающее огромное количество ресурсов. Да, простой отчет может быть независимым от базы данных. Но возможно ли это в случае масштабируемой системы транзакций? Нет, если только не применяется подход компании PeopleSoft или SAP. Подобно продуктам этих компаний, ваше приложение не будет использовать SQL после простого «закрытого чтения» (может быть применена сортировка подобно VSAM-файлам на мэйнфрейме, файл читается только закрытым — никаких соединений, никакого анализа, только закрытое чтение). Оно не будет пользоваться никакими расширениями производителей и функциями ANSI SQL, так как они реализуются не всеми производителями. Невозможно будет применять базу данных для управления параллельным доступом и для анализа (поскольку все сделано по-разному). В принципе, написание базы данных на этом и закончится. Фактически, SAP поступил следующим образом — написал свою собственную базу данных.

Так что, если не стоит задача создания продукта, который будет работать в качестве готового программного обеспечения с множеством различных баз данных, то независимость базы данных не должна являться конечной целью. В действительности, большая часть программного обеспечения создана именно для работы со стандартной корпоративной базой данных. Представляется сомнительной в подобных обстоятельствах необходимость в независимости от базы данных. Создать приложение быстро и всего лишь с помощью нескольких строчек кода — это то, к чему необходимо стремиться. А работа с условием независимости от базы данных или, что еще хуже, пренебрежение (преднамеренное или как результат недостаточных знаний) возможностями базы данных не должно быть целью.

Лучшим способом, позволяющим достичь некоторого уровня мобильности приложения для множества баз данных, является кодирова-

ние всех компонентов базы данных, используемых приложением, в хранимых процедурах. В связи с этим возникает вопрос: если кодирование производится в хранимых процедурах, а каждый производитель имеет свой собственный язык, не возникнет ли зависимость от производителя? И да, и нет. Видимые компоненты приложения в безопасности. Логика приложения (существующая вместе с логикой данных) тоже в безопасности. Логика данных кодируется в процессе работы с базой данных.Так как все это скрыто в хранимой процедуре, пользуясь (на самом деле это всегда нужно использовать) расширенными средствами каждого производителя, можно получить данные более высокого уровня (см. ниже).

Если приложение разработано и внедрено, то оно навсегда остается внедренным в базу данных. Если приложение перемещается в другую базу данных, то его необходимо адаптировать, обновив с помощью новых возможностей и функций. Просто так перенести не получится.

Опасности синдрома черного ящика

Ниже приводятся причины, по которым не следует рассматривать базу данных как черный ящик:

- **Невозможность получения корректного ответа** Средства управления параллельным доступом — это главное отличие между базами данных. В зависимости от того, с какой базой данных работают приложения, они получат различные результаты, несмотря на одинаковые входные данные и порядок ввода запроса.
- **Невозможность получения высокой производительности** Производительность будет всего лишь долей того, что могло и должно было бы быть.
- **Невозможность быстрого создания программного обеспечения** Придется тратить много времени на то, чтобы делать все самому.
- **Неэффективное вложение средств** Если есть возможность потратить большую сумму денег на программное обеспечение базы данных, то их нужно тратить. Однако хочется с иронией отметить ситуацию, когда меняют производителя программного обеспечения по той причине, что его приложения работают недостаточно хорошо, в то время как причиной плохой работы является, в первую очередь, неполное использование возможностей программ этого производителя.

Данный список не упорядочен. В зависимости от того, кем является пользователь, пункты могут иметь для него различное значение. Рассмотрим несколько примеров.

Невозможность получения высокой производительности

Предположим, необходимо разработать приложение, осуществляющее сортировку работников или материалов в перечне материалов в определенном порядке. В Oracle любые иерархии строятся с помощью SQL-оператора CONNECT BY. Наиболее эффективным является следующее решение (используется стандартная в Oracle таблица SCOTT.EMP):

```
scott@ORA920.US.ORACLE.COM> select rpad ('*',2*level,'*') || ename ename
  2  from emp
  3  start with mgr is null
  4  connect by prior empno = mgr
  5  /
ENAME
------------------------------
**KING
****JONES
******SCOTT
********ADAMS
******FORD
********SMITH
****BLAKE
******ALLEN
******WARD
******MARTIN
******TURNER
******JAMES
****CLARK
******MILLER

14 rows selected.
```

Оператор CONNECT BY не универсален. Многие базы данных не поддерживают этот синтаксис. В других базах данных необходимо написать некоторую процедуру и поместить результаты во временную таблицу. Можно было бы и в Oracle сделать так же, но зачем? Это будет медленнее и займет больше ресурсов, одним словом, это неверный подход к решению задачи. А подход должен быть верным. Значит, необходимо использовать конструкцию CONNECT BY и «спрятать» ее в хранимую процедуру. Ее можно реализовать по-разному в различных базах данных, например в Microsoft SQL Server, если потребуется (по моему опыту, это бывает редко).

Продолжим. Допустим, что необходимо получить в качестве результата информацию о работниках: имя работника, департамент и заработную плату. Также требуется получить общую сумму заработной платы по департаментам и процент заработной платы конкретного работника в сумме департамента и общей сумме (например, работник X в департаменте Y получает 10% заработной платы от суммы заработной платы его департамента и 1% от общей суммы заработной платы компании). Правильным подходом к решению этой задачи в Oracle является использование аналитических функций:

```
scott@ORA920.US.ORACLE.COM> column pct_dept format 99.9
scott@ORA920.US.ORACLE.COM> colums pct_overall format 99.9
scott@ORA920.US.ORACLE.COM> break on deptno skip 1

scott@ORA920.US.ORACLE.COM> select deptno,
  2        ename,
  3        sal,
  4        sum(sal) over (partition by deptno order by sal,ename) cum_sal,
  5        round(100*ratio_to_report(sal)
  6             over (partition by deptno), 1 ) pct_dept,
  7        round(100*ratio_to_report(sal) over ( ) , 1 ) pct_overall
  8   from emp
  9  order by deptno, sal
 10 /
```

```
DEPTNO ENAME      SAL  CUM_SAL  PCT_DEPT  PCT_OVERALL
------ --------- ----- -------- --------- -----------
    10 MILLER     1300     1300      14.9         4.5
       CLARK      2450     3750      28.0         8.4
       KING       5000     8750      57.1        17.2

    20 SMITH       800      800       7.4         2.8
       ADAMS      1100     1900      10.1         3.8
       JONES      2975     4875      27.4        10.2
       FORD       3000     7875      27.6        10.3
       SCOTT      3000    10875      27.6        10.3

    30 JAMES       950      950      10.1         3.3
       MARTIN     1250     2200      13.3         4.3
       WARD       1250     3450      13.3         4.3
       TURNER     1500     4950      16.0         5.2
       ALLEN      1600     6550      17.0         5.5
       BLAKE      2850     9400      30.3         9.8

14 rows selected.
```

Однако многие реляционные базы данных не имеют такого средства, как аналитические функции, так что, в сущности, это зависимая от базы данных технология. Существует другой способ решения этой задачи, который будет работать в большинстве баз данных. Этот подход предполагает использование соединений, представлений и т.п.

```
scott@ORA920.US.ORACLE.COM> select emp.deptno,
  2       emp.ename,
  3       emp.sal,
  4       sum(emp4.sal) cum_sal,
  5       round(100*emp.sal/emp2.sal_by_dept,1) pct_dept,
  6       round(100*emp.sal/emp3.sal_overall,1) pct_overall,
  7   from emp,
  8       (select deptno, sum(sal) sal_by_dept
  9          from emp
 10          group by deptno ) emp2,
 11        (select sum(sal) sal_overall
 12           from emp ) emp3,
 13        emp emp4
 14   where emp.deptno = emp2.deptno
 15     and emp.deptno = emp4.deptno
 16     and (emp.sal > emp4.sal or
 17          (emp.sal = emp4.sal and emp.ename >= emp4.ename))
 18   group by emp.deptno, emp.ename, emp.sal,
 19        round (100*emp.sal/emp2.sal_by_dept,1),
 20        round (100*emp.sal/emp3.sal_overall,1)
 21   order by deptno, sal
 22 /

DEPTNO ENAME      SAL  CUM_SAL  PCT_DEPT  PCT_OVERALL
------ --------- ----- -------- --------- -----------
    10 MILLER     1300     1300      14.9         4.5
       CLARK      2450     3750      28.0         8.4
       KING       5000     8750      57.1        17.2
```

20	SMITH	800	800	7.4	2.8
	ADAMS	1100	1900	10.1	3.8
	JONES	2975	4875	27.4	10.2
	FORD	3000	7875	27.6	10.3
	SCOTT	3000	10875	27.6	10.3
30	JAMES	950	950	10.1	3.3
	MARTIN	1250	2200	13.3	4.3
	WARD	1250	3450	13.3	4.3
	TURNER	1500	4950	16.0	5.2
	ALLEN	1600	6550	17.0	5.5
	BLAKE	2850	9400	30.3	9.8

14 rows selected.

Такой метод работает и является независимым от базы данных. Однако вряд ли в компаниях, где будет использоваться эта функциональность, работает всего 14 человек. Обычно счет идет на сотни и тысячи. Попробуем увеличить масштаб примера и посмотрим, как это повлияет на производительность. Применим эту процедуру к наборам данных разных размеров:

Строк в таблице	Процессор/ аналитические функции	Процессор/ универсальные функции	Разница
2000	0.05	2.13	42 раза
4000	0.09	8.57	95 раз
8000	0.19	35.88	188 раз

Из таблицы видно, что при реализации универсального метода по мере увеличения количества данных показатели производительности значительно ухудшаются. При каждом двукратном увеличении объема данных время, требуемое на аналитический процесс, также увеличивается в два раза, а в случае применения универсальной функции времени требуется еще больше.

Таким образом, если бы я был конечным пользователем, я однозначно не выбрал бы универсальный метод. Этот пример показывает, почему аналитические средства, на упаковке которых написано «мы работаем с вашей родной базой данных», более предпочтительны, чем средства, которые хвастаются: «мы универсально работаем на 15 базах данных». Средства и приложения, которые предназначены для конкретной базы данных, будут более производительны по сравнению с универсальными решениями. Единственные, кто будет рад применению универсальных решений, — это поставщики аппаратных средств, ведь пользователю придется наращивать мощность процессора.

Существуют и другие возможные решения этой проблемы. Можно, например, использовать временные таблицы для размещения данных. Однако все эти подходы обладают теми же недостатками, что и описанные ранее: они являются неверным решением для Oracle (возможно, для других баз данных они являются правильными), и все они требуют написания большого объема кода, что является трудоемким процессом.

А вот другой пример. Я работал с заказчиком, который отказался от применения битового индекса. Его мнение было следующим: «Не во всех программах он работает. Я не хочу помещать его в мою систему, так как индекс, создаваемый в Oracle, может не работать в других базах данных. Мне хотелось бы использовать универсальный сценарий, который будет без труда работать во всех базах данных». При использовании битового индекса выполнение запроса занимало бы меньше минуты, а не часы. Но заказчик решил «наказать» все реализации, поскольку не во всех из них была эта специфическая возможность. Не знаю, как другие, но лично я выбрал бы битовый индекс в Oracle, чем работу вообще без индекса.

Невозможность получения корректного ответа

Если пользователь воспринимает базу данных как черный ящик, возникают трудности не только с достижением высокой производительности, но и с получением корректного ответа. В предыдущем разделе результаты были достаточно очевидны; при рассмотрении производительности обработки данных удалось сразу выявить правильный и неправильный подходы к решению задачи. Однако не всегда сразу можно определить, где была допущена ошибка.

Рассмотрим пример, касающийся управления параллельным доступом (многовариантность, согласованность чтения, блокировка и т.п.), поскольку это те вопросы, на которых защитники идеологии черного ящика часто проваливаются. Применим простую транзакцию к родительской/дочерней таблице. Цель этой задачи — сохранить результат суммирования строк дочерней таблицы в родительской таблице, например, сохранить сумму индивидуальных зарплат работников в таблице департамента. В примере мы используем две простые таблицы:

```
ops$tkyte@ORA920> create table dept
  2  ( deptno int primary key,
  3    sum_of_salary number
  4  );
Table created.

ops$tkyte@ORA920> create table emp
 2  ( empno int primary key,
 3    deptno references dept,
 4    salary number
 5  );
Table created.

ops$tkyte@ORA920> insert into dept ( deptno ) values (1);
1 row created.

ops$tkyte@ORA920> insert into dept ( deptno ) values (2);
1 row created.
```

После выполнения транзакций в дочерней таблице EMP можно применить оператор UPDATE к родительской таблице DEPT для того, чтобы синхронизировать столбец SUM_OF_SALARY. Например, включим в транзакцию оператор UPDATE, идущий последним:

```
ops$tkyte@ORA920> insert into emp (empno, deptno, salary)
  2 values (100, 1, 55);
```

```
1 row created.

ops$tkyte@ORA920> insert into emp (empno, deptno, salary)
  2  values (101, 1, 50);
1 row created.

ops$tkyte@ORA920> update dept
  2     set sum_of_salary =
  3     ( select sum (salary)
  4         from emp
  5           where emp.deptno = dept.deptno )
  6   where dept.deptno = 1;
1 row updated.

ops$tkyte@ORA920> commit;
Commit complete.
```

Это выглядит просто — только вставить дочернюю запись и обновить сумму записей в родительской таблице. Можно ли допустить ошибку в таком простом примере? Если мы сейчас выполним запрос к схеме, то получим:

- ops$tkyte@ORA920> select * from emp;

```
     EMPNO         DEPTNO         SALARY
----------- -------------- --------------
       100              1             55
       101              1             50
```

```
ops$tkyte@ORA920> select * from dept;

     DEPTNO  SUM_OF_SALARY
----------- --------------
          1            105
          2
```

Если добавить строки к дочерней таблице для DEPTNO 1 или DEPTNO 2 и запустить обновление, то все будет хорошо. Но не был рассмотрен вопрос, что произойдет при одновременном (параллельном) доступе к этой таблице. Что, если два пользователя работают с дочерней таблицей EMP в одно и то же время? Один пользователь будет добавлять нового работника в DEPTNO 2. Второй пользователь будет переводить EMPNO 100 из DEPTNO 1 в DEPTNO 2. Рассмотрим случай, когда эти транзакции выполняются синхронно. Проследим за событиями:

Время	Работа сессии 1	Работа сессии 2
T1	Ввод в EMP (EMPNO, DEPTNO, SALARY) значений (102,2,60); добавляется новый работник в DEPTNO 2	
T2		Обновление EMP (set DEPTNO=2 where EMPNO=100); перевод работника из одного департамента в другой
T3		Обновление DEPT для департаментов 1 и 2 после модификации записей в обоих департаментах

Время	Работа сессии 1	Работа сессии 2
T4	Обновление DEPT для департамента 2, департамент изменен. Это будет блокировано, поскольку в сессии 2 есть заблокированные строки. Однако компонент запроса UPDATE начнет свою работу в случае, если результирующий набор уже построен. Механизм согласованного чтения в Oracle вернет к оператору UPDATE, что будет зафиксировано в базе данных в момент времени T4.	
T5		Завершение транзакции, сессия 1 разблокируется
T6	Завершение транзакции	

Можно легко смоделировать последовательность событий в транзакциях, используя две сессии и переключаясь между ними на экране.

Ниже представлен пример с двумя сессиями. Командная строка SQLPlus показывает, с какой сессией идет работа в настоящий момент. Можно открыть одновременно два окна SQLPlus при выполнении этого примера:

❒
```
Session 1> insert into emp ( empno, deptno, salary )
  2  values ( 102, 2, 60 )
1 row created.

Session 2> update emp
  2     set deptno = 2
  3   where empno = 100;
1 row updated.

Session 2> update dept
  2     set sum_of_salary = ( select sum (salary)
  3                           from emp
  4                          where emp.deptno = dept.deptno )
  5   where dept.deptno in ( 1, 2 );
2 rows updated.

Session 1> update dept
  2     set sum_of_salary = ( select sum (salary)
  3                           from emp
  4                          where emp.deptno = dept.deptno )
  5   where dept.deptno =  2;
```

Теперь сессия 1 будет блокирована. Это происходит после попытки модифицировать строку, которая уже заблокирована сессией 2. Однако чтение фрагмента оператора UPDATE (части запроса) уже произведено. Oracle уже зафиксировал этот результат, используя механизм, называемый «согласованным чтением» (см. главу 5). Вкратце, обновление таблицы DEPT сессией 2 не видит вставку строки сессией 1, и оператор обновления в сессии 1 не будет видеть обновление таблицы EMP в сессии 2. Продолжим пример:

❒
```
Session 2> commit;
Commit complete.
```

В этом месте сессия 1 будет разблокирована. В окне сессии 1 немедленно появится сообщение «Обновлена 1 строка». Затем сессия 1 будет завершена:

❒ Session 1> commit;
Commit complete.

Session 1> select * from dept;

DEPTNO	SUM_OF_SALARY
1	50
2	60

Session 1> select deptno, sum(salary) from emp group by deptno;

DEPTNO	SUM(SALARY)
1	50
2	115

Очевидно, что это неверно. Значение для DEPTNO 2 неправильное. Как такое могло произойти? Если запустить этот пример на SQL-сервере, сценарий будет слегка отличаться (исполнение будет производиться в другом порядке), но суммирование будет реализовано. На SQL-сервере последовательность событий будет такой:

Время	Работа сессии 1	Работа сессии 2
T1	Добавление в EMP (EMPNO, DEPTNO, SALARY) значений (102,2,60); добавление нового работника в DEPTNO 2	
T2		Обновление EMP (set DEPTNO=2 where EMPNO =100); перевод работника из одного департамента в другой
T3		Обновление DEPT для департаментов 1 и 2 после модификации записей в обоих департаментах. Этот оператор блокирует чтение EMP. Строки, вставленные в момент времени T1, заблокированы, и SQL-сервер ждет, когда будет реализована блокировка.
T4	Обновление DEPT для департамента 2 (департамент изменен). Этот оператор также блокирует чтение строк, обновленных в момент времени T2.	
T5	Сервер определяет, что обе сессии находятся в тупиковых условиях. Одна из них выбирается в качестве жертвы, и делается откат транзакции (например, сессии 1). Откат.	Операторы разблокированы
T6		Завершение транзакции

Из примера видно, что, используя блокировку и механизм управления параллельным доступом, SQL-сервер не позволит транзакциям выполняться совместно. Только одна транзакция будет выполнена, а по остальным будет произведен откат, и результат будет верный с точки зрения SQL-сервера. Так может быть это ошибка Oracle? Вовсе нет.

В Oracle есть средство, называемое «многовариантность и согласованное чтение» (это отличает Oracle от остальных реляционных баз данных). Его методы работы отличаются от работы SQL-сервера (я бы сказал, что они гораздо лучше, так как предлагают большую возможность параллельной работы, дают корректные ответы без ожидания, но все это не относится к рассматриваемому здесь примеру). Поскольку используется модель параллелизма, вторая сессия при обращении к данным, которые были изменены в процессе обновления, не увидит их (не сможет прочитать). Следовательно, обновление не увидит добавления записи. И хотя очередность действий в транзакциях имеет небольшое различие, результаты могут сильно отличаться. (Информацию о согласованном чтении и многовариантности Oracle можно найти в *общем руководстве* (Concepts guide).)

Мораль этого примера заключается в том, что фундаментальные модели согласованности и управления параллелизмом разных баз данных радикально отличаются друг от друга. Последовательность операторов в одной базе данных может, а иногда и будет, приводить к результату, отличному от результатов в другой базе данных. Это «иногда» относится к тому оператору, с которым возникают сложности. До тех пор пока не будет изучена документация и освоена база данных, будут возникать проблемы с целостностью данных, что будет затруднять отладку. При чтении кода невозможно выявить условия, описанные в этом примере. При применении транзакций к целевой базе данных всегда необходимо помнить, что транзакции, работающие в одной базе, не всегда будут так же работать в другой. При таком подходе обеспечивается страховка от подобных проблем.

Если члены команды разработчиков не имеют представления о работе механизма согласованности и управления параллелизмом в Oracle или, что еще хуже, считают, что он работает подобно механизму SQL-сервера или DB2, то в этом случае самый вероятный исход — это испорченные данные, неверный анализ и некорректные ответы.

Невозможность быстрого создания программного обеспечения

Если разработчик знает все об использовании базы данных и ее возможностях, то процесс создания приложений займет гораздо меньше времени. Обратимся вновь к примеру применения аналитических функций (см. выше). При использовании универсального метода не только производительность была хуже, но и потребовалось больше времени на его разработку! Было сложно найти решение этой задачи. Если бы я использовал временную таблицу для побитового получения результатов, мне пришлось бы потратить много больше времени на написание объемного процедурного кода.

В качестве другого примера рассмотрим ситуацию, когда необходимо создать приложение, способное проверить внесенные изменения. История строки от начала до конца должна сохраняться в базе данных. Можно пойти двумя путями:

- Проектирование, запись, отладка и затем сопровождение собственной реализации
- Использование единственной команды базы данных, реализующей такую же функциональность