Введение

Побуждение изложить собранный в этой книге материал возникло у меня вследствие приобретенного опыта разработки программного обеспечения Oracle, сотрудничества с другими разработчиками Oracle и оказания им помощи в построении надежных приложений на основе базы данных Oracle. В основном эта книга отражает то, чем мне приходится заниматься ежедневно, и в ней освещены вопросы, с которыми пользователи сталкиваются в повседневной работе.

Я постарался осветить наиболее важный, по моему мнению, вопрос, а именно — архитектура баз данных Oracle. Можно было бы написать аналогичным образом озаглавленную книгу, посвященную разработке приложений с использованием конкретного языка и архитектуры — например, приложение, использующее JavaServer Pages для взаимодействия с Enterprise JavaBeans, которые, в свою очередь, с помощью JDBC осуществляют обмен данными с Oracle. Однако для успешного построения такого приложения необходимо действительно разбираться в вопросах, освещенных в этой книге. В ней собран материал, который, по моему глубокому убеждению, должен быть всесторонне усвоен для успешной разработки с использованием Oracle, как программистом на Visual Basic, использующим ODBC, так и программистом на Java, использующим EJB и JDBC, или программистом на Perl, применяющим DBI Perl. В книге не отдается предпочтение какой-то конкретной архитектуре приложений и не выполняется сравнение трехуровневой архитектуры с архитектурой типа клиент-сервер. Вместо этого в ней описаны возможности базы данных и пояснены особенности ее работы. Поскольку база данных — ядро архитектуры любого приложения, книга должна заинтересовать широкий круг читателей.

Как следует из названия, эта книга посвящена архитектуре базы данных и работе самой базы данных. Я подробно освещаю архитектуру базы данных Oracle — файлы, структуры памяти и процессы, образующие как базу данных в целом, так и ее экземпляр. Затем я перехожу к освещению таких важных тем базы данных, как блокировка, управление параллелизмом, работа транзакций, повторение и отмена, и поясняю, почему эти вопросы важны. И, наконец, я рассматриваю физические структуры базы данных, такие как таблицы, индексы и типы данных, освещая при этом приемы их оптимального использования.

О чем эта книга

Одна из проблем, обусловленная наличием множества возможностей выполнения разработки, состоит в том, что иногда трудно выбрать решение, которое наиболее подходит для конкретных нужд. Всем требуется обеспечение максимальной гибкости (наличие максимального числа возможностей), но при этом желательно, чтобы все оставалось как можно более лаконичным — иначе говоря, простым. Oracle предлагает разработчикам практически неограниченную возможность выбора. Никто никогда не скажет: “Это нельзя выполнить в Oracle”. Скорее, может прозвучать вопрос: “Сколькими различными способами вы желаете выполнить это в Oracle?”. Надеюсь, что настоящая книга поможет делать правильный выбор.

Книга адресована тем, кто уважает свободу выбора, но при этом хотел бы располагать определенными инструкциями и сведениями о конкретной реализации свойств и функций Oracle. Например, СУБД Oracle имеет действительно эффективное средство, которое называется *параллельным выполнением*. В документации Oracle рассказано, как использовать это средство и как оно работает. Однако в ней ничего не сказано о том, *когда* следует и — что еще важнее — *когда не следует* его использовать. Нюансы реализации этого средства описаны недостаточно подробно и, если о них не знать, это может привести к неприятным последствиям (речь идет не о программных ошибках, а о предполагаемом способе работы средства и о том, для чего в действительности оно предназначено).

Поэтому в книге я стремился не только описать те или иные функциональные возможности, но и пояснить, когда и почему следует подумать о применении конкретной функции или реализации. Как мне кажется, важно понимать не только "как" выполняются те или иные действия, но и "когда" и "почему" — равно как и "когда нет" и "почему нет"!

Кому адресована эта книга

Эта книга адресована всем, кто занимается разработкой приложений на основе баз данных Oracle. Она предназначена для профессиональных разработчиков Oracle, которым требуется знание способов выполнения действий в базе данных. Практическая природа этой книги такова, что многие разделы должны также весьма заинтересовать администраторов баз данных. Для демонстрации ключевых средств в большинстве примеров применяется язык SQL*Plus, поэтому их вряд ли можно использовать для разработки действительно оригинального графического интерфейса пользователя, но зато читатели увидят, как работает база данных Oracle, какие действия позволяют выполнять ее основные средства, и когда следует (а когда не следует) их применять.

Книга адресована всем, кто стремится максимально использовать базу данных Oracle, затрачивая при этом как можно меньше усилий. Она будет интересна тем, кто ищет новые способы применения существующих функциональных средств, и тем, кто желает выяснить, как эти средства можно использовать при решении реальных задач (не просто ознакомиться с примерами применения того или иного свойства, но и выяснить, почему данное свойство используется в конкретном случае). Еще одна категория людей, которых может заинтересовать эта книга — руководители групп разработчиков, выполняющих проекты на Oracle. Исключительно важно, чтобы эти руководители в полной мере осознавали, почему знание базы данных имеет решающее значение для успешного выполнения проекта. Эта книга может стать серьезным подспорьем руководителям различных уровней, которые стремятся к тому, чтобы их подчиненные были в курсе новейших технологий, или хотят убедиться, что персонал уже обладает необходимыми знаниями. Чтобы прочтение книги было наиболее эффективным, понадобятся следующие условия.

- *Знание языка SQL*. Вовсе не обязательно быть наилучшим программистом SQL, но хорошие практические навыки не помешают.
- *Представление о PL/SQL*. Это требование не обязательно, но наличие таких знаний облегчит усвоение примеров. Например, в этой книге вы не найдете инструкций по применению цикла `FOR` или объявлению типа данных — эти вопросы подробно освещены в документации по Oracle и во множестве книг. Однако это не означает, что в процессе чтения данной книги нельзя почерпнуть ценные сведения о PL/SQL. Это не так. Вы ознакомитесь со многими средствами PL/SQL, узнаете о новых способах решения задач и о пакетах/средствах, о существовании которых, возможно, даже не подозревали.

- *Некоторое представление о языках третьего поколения, таких как C или Java.* Я уверен, что любой, кто способен читать и создавать код на языке третьего поколения, сможет успешно прочитать и понять примеры, приведенные в этой книге.
- *Знание руководства* Oracle Concepts *(Концепции Oracle).*

Последний пункт требует небольшого пояснения. Поскольку документация по Oracle достигает весьма значительных объемов, многих это обстоятельство несколько пугает. Если вы лишь приступаете к изучению СУБД Oracle или еще не прочли ни одного из таких документов, руководство *Oracle Concepts* как раз подойдет в качестве начального учебного пособия. Его объем — около 400 страниц (я знаю это, поскольку сам писал некоторые из страниц, а редактировал их все), и в нем освещены многие из основных концепций Oracle, о которых обязательно нужно знать. Возможно, вы не найдете в нем подробное описание всех нюансов (им посвящены остальные от 10 000 до 20 000 страниц документации), но оно поможет ознакомиться со всеми наиболее важными концепциями. В этом руководстве освещены следующие темы (ниже перечислены далеко не все).

- Структуры в базе данных и способ организации и хранения данных.
- Распределенная обработка.
- Архитектура памяти Oracle.
- Архитектура процессов Oracle.
- Объекты схемы, которые будут использоваться в приложениях (таблицы, индексы, кластеры и т.д.).
- Встроенные и определяемые пользователем типы данных.
- Хранимые процедуры SQL.
- Работа транзакций.
- Оптимизатор.
- Целостность данных.
- Управление параллелизмом.

Периодически я и сам буду возвращаться к этим темам. Они являются основополагающими, и отсутствие знаний по ним приведет к созданию приложений Oracle, подверженных сбоям. Поэтому я настоятельно рекомендую прочитать руководство и усвоить некоторые из этих тем.

Как структурирована эта книга

Для облегчения изучения материала большинство глав содержит четыре основных раздела (описаны в списке ниже). Хоть это и не жесткое разделение, тем не менее, оно помогает быстрее ориентироваться в нужной информации. Книга содержит 16 глав, каждая из которых представляет собой своего рода “мини-книгу” — практически самостоятельный компонент. Иногда я ссылаюсь на примеры или средства, описанные в других главах, но любую главу книги вполне можно читать независимо от остальных. Например, вовсе не обязательно сначала читать главу 10, посвященную таблицам базы данных, чтобы понять или применить материал, изложенный в главе 14, которая посвящена параллельному выполнению.

Формат и стиль многих глав практически идентичен.

- Введение в средство или возможность.
- Обоснование применения (или не применения) средства или возможности. Здесь приводятся пояснения того, когда следует подумать об использовании этого средства, а когда — нет.
- Как использовать средство. Приведенная здесь информация — не просто копия материала, заимствованного из справочника по SQL. Она представляет собой пошаговые инструкции выполнения конкретных задач: отдельно описан требуемый результат, отдельно — действия, которые понадобится выполнить для его достижения, и отдельно — варианты выбора, которые необходимо сделать, чтобы приступить к работе. В этом разделе рассматриваются следующие вопросы.
  - Как реализовать средство.
  - Примеры применения.
  - Как отлаживать средство.
  - Предостережения относительно использовании средства.
  - Как обрабатывать ошибки (упреждающим образом).
- Резюме с кратким подведением итогов.

Главы будут содержать множество примеров и кода, которые доступны для загрузки на сайте издательства. В последующих разделах приведено краткое описание содержимого каждой из глав.

Глава 1. Разработка успешных приложений Oracle

В этой главе я описываю используемый мною подход к программированию баз данных. Все базы данных отличаются одна от другой и, чтобы успешно и своевременно разрабатывать приложения, управляемые базами данных, необходимо четко понимать, *что* конкретная база данных может делать, и *как* она это делает. Не зная, что может делать база данных, вы рискуете постоянно заново изобретать колесо, создавая средства, которые база данных уже предоставляет. Если не знать, как работает база данных, скорее всего, разработанные приложения будут работать неэффективно и вести себя непредсказуемым образом.

В этой главе рассматриваются с эмпирической точки зрения некоторые приложения, где недостаток знаний ведет к неудаче всего проекта. В соответствии с этим ориентированным на примеры подходом, в главе описаны основные средства и функции базы данных, которые должны быть поняты разработчиком. Основная идея состоит в том, что к базе данных нельзя относиться как к черному ящику, который просто выдает ответы и самостоятельно заботится о масштабируемости и производительности.

Глава 2. Обзор архитектуры

В этой главе освещены основные принципы архитектуры базы данных Oracle. Я начинаю с того, что привожу ряд понятных определений двух терминов, которые многие из тех, кому приходится иметь дело со средой Oracle, понимают не правильно — понятий “экземпляр” и “база данных”. Кроме того, кратко рассматривается системная глобальная область (System Global Area) и процессы, лежащие в основе экземпляра базы данных Oracle, а также исследуется, как именно выполняется простое действие “подключения к базе данных Oracle”.

Глава 3. Файлы

В этой главе подробно описаны восемь типов файлов, образующих базу данных Oracle и ее экземпляры. Мы рассмотрим, что собой представляют эти файлы — от про-

стого файла параметров до файла данных и файлов журнала повторения выполненных действий — каково их назначение и способ использования.

Глава 4. Структуры памяти

В этой главе освещены вопросы использования памяти базой данных Oracle, как в отдельных процессах (PGA), так и разделяемой памяти (SGA). Мы рассмотрим различия между ручным и автоматическим управлением памятью PGA и (в Oracle 10*g*) SGA, а также выясним, в каких случаях подходит каждый из этих методов управления памятью. После прочтения этой главы вы получите полное представление о том, как база данных Oracle использует память и управляет ею.

Глава 5. Процессы Oracle

В этой главе предлагается обзор типов процессов Oracle (серверных и фоновых процессов). Кроме того, в ней более подробно описаны различия между подключением к базе данных с помощью процесса разделяемого сервера и выделенного сервера. Мы рассмотрим также большинство фоновых процессов (такие как `LGWR`, `DBWR`, `PMON` и `SMON`), действующих во время запуска экземпляра базы данных Oracle, и их функции.

Глава 6. Блокировка и защелкивание данных

В различных базах данных задачи решаются по-разному (то, что подходит для SQL Server, может не годиться для Oracle), и понимание особенностей реализации блокировки и управления параллелизмом — абсолютно необходимое условие успешной работы приложения. В этой главе рассматривается общий подход, используемый в Oracle для решения этих задач, типы блокировок, которые могут быть применены (DML, DDL и защелки), а также проблемы, которые могут возникать при неправильной реализации блокировки (взаимоблокировка, блокировка и эскалация).

Глава 7. Параллелизм и многоверсионность

В этой главе рассматривается мое любимое средство Oracle — многоверсионность, а также его влияние на управление параллелизмом и на всю структуру приложения. В этой главе будет показано, что все базы данных отличаются друг от друга, а их конкретная реализация может влиять на структуру приложений. Мы начнем с обзора различных уровней изоляции транзакций, определенных в стандарте ANSI SQL, и их отражения в реализации базы данных Oracle (а также соответствие этому стандарту других СУБД). Затем мы проанализируем последствия, которые может иметь многоверсионность — средство, позволяющее СУБД Oracle обеспечивать неблокирующие чтения базы данных.

Глава 8. Транзакции

Транзакции — это фундаментальное средство всех СУБД. Они относятся к тем возможностям, которые отличают базу данных от файловой системы. Но, несмотря на это, зачастую их понимают неправильно, и многие разработчики даже не подозревают, что неумышленно отказываются от их применения. В этой главе рассматриваются рекомендуемые способы использования транзакций в Oracle, а также некоторые плохие привычки, которые могут быть приобретены при разработке приложений в других СУБД. В частности, будут показаны последствия атомарности и ее влияние на операторы в Oracle. Кроме того, мы рассмотрим операторы управления транзакциями (`COMMIT`, `SAVEPOINT` и `ROLLBACK`), ограничения целостности, распределенные транзакции (двухфазная фиксация) и автономные транзакции.

Глава 9. Повтор и отмена

Вообще говоря, разработчикам не обязательно разбираться в нюансах реализации повтора (redo) и отмены (undo) действий в такой же степени, как администраторам баз данных, но разработчики должны знать, какую роль это играет в базе данных. После определения понятия повтора мы подробно рассмотрим действия, выполняемые оператором `COMMIT`. Мы выясним, как установить объем сгенерированной информации redo и как с помощью конструкции `NOLOGGING` существенно уменьшить объем информации redo, генерируемой определенными операциями. Мы исследуем также генерацию redo в связи с такими вопросами, как очистка блоков и конкуренция за журналы.

В разделе главы, посвященном отмене, рассматривается роль данных undo и операции, которые генерируют наибольшее/наименьшее количество информации undo. И, наконец, мы коснемся пресловутой ошибки `ORA-01555: snapshot too old` (ORA-01555: устаревший снимок), возможные причины ее возникновения и способы ее предотвращения.

Глава 10. Таблицы базы данных

В настоящее время Oracle поддерживает множество типов таблиц. В этой главе рассмотрены различные типы таблиц — традиционные таблицы ("обычные", используемые по умолчанию), индекс-таблицы, кластеризованные индекс-таблицы, кластеризованные хеш-таблицы, вложенные таблицы, временные таблицы и объектные таблицы — и описано когда, как и почему их следует применять. В большинстве случаев вполне достаточно традиционной таблицы, но эта глава поможет определить ситуации, в которых более подходящим может оказаться какой-то другой тип.

Глава 11. Индексы

Индексы — критически важный аспект структуры приложения. Для правильной реализации требуются глубокие знания о данных, их распределении и способе использования. Слишком часто при разработке приложений к индексам относятся как к чему-то второстепенному, что ведет к снижению производительности.

В этой главе подробно рассматриваются различные типы индексов, в том числе индекс со структурой B-дерева (B*Tree), битовый индекс, индекс на основе функций и индекс предметной области, и описано, когда следует, а когда не следует их применять. В разделе, посвященном часто задаваемым вопросам и мифам об индексах, я отвечу также на ряд обычно задаваемых вопросов вроде "Работают ли индексы в представлениях?" и "Почему индекс не используется?".

Глава 12. Типы данных

Существует множество доступных типов данных. В этой главе описан каждый из 22 встроенных типов данных, особенности их реализации, а также способы и ситуации их применения. Вначале приведен краткий обзор поддержки национальных языков (National Language Support — NLS), наличие общего представления о которой обязательно для полного понимания простых строковых типов данных Oracle. Затем мы перейдем к рассмотрению вездесущего типа `NUMBER` и рассмотрим новые возможности хранения чисел в базе данных, появившиеся в Oracle 10*g*. Мы рассмотрим (в основном, с исторической точки зрения) типы `LONG` и `LONG RAW`. Основное назначение этого материала — показать, как работать в приложениях с унаследованными столбцами типа `LONG` и преобразовывать их в тип `LOB`. Затем мы рассмотрим различные типы данных, предназначенные для хранения значений дат и времени, и уделим внимание способам

манипулирования различными типами данных для получения нужного результата. Будут описаны также особенности поддержки часового пояса.

Затем мы перейдем к типам данных LOB. Мы рассмотрим способы их хранения и значение каждого из множества параметров настройки, таких как IN ROW, CHUNK, RETENTION, CACHE и т.д. При работе с типами данных LOB важно понимать, как они реализованы и как хранятся по умолчанию — особенно, когда дело доходит до настройки их извлечения и хранения. Глава завершается описанием типов ROWID и UROWID. Это специальные патентованные типы Oracle, которые представляют адрес строки. Мы рассмотрим ситуации, в которых их следует применять в таблице в качестве типа данных столбца (что почти никогда не случается).

Глава 13. Секционирование

Секционирование предназначено для облегчения управления очень большими таблицами и индексами за счет реализации концепции "разделяй и властвуй" — в основном оно сводится к разбиению таблицы или индекса на множество мелких и более управляемых частей. Это область, в которой администратор базы данных и разработчик должны сотрудничать для обеспечения максимальной доступности и производительности приложения. В главе рассматривается секционирование и таблиц, и индексов. Будет описано секционирование с использованием локальных индексов (характерная особенность хранилищ данных) и глобальных индексов (обычная черта систем OLTP).

Глава 14. Параллельное выполнение

В этой главе представлена концепция параллельного выполнения в Oracle. Мы начнем с рассмотрения ситуаций, в которых параллельная обработка полезна, и в которых к ней следует прибегать, и ситуаций, когда она не нужна. После того, как вы получите общее представление по этому вопросу, будут показаны особенности реализации параллельного запроса — средства, которое для большинства людей ассоциируется с параллельной обработкой. Затем будет описан язык PDML (Parallel DML), позволяющий выполнять модификации с использованием параллельного выполнения. Будет показано, как PDML физически реализован, и почему эта реализация приводит к ряду ограничений, связанных с PDML.

Затем мы перейдем к рассмотрению параллельного DDL. По моему мнению, именно здесь параллельное выполнение проявляется во всей своей красе. Как правило, администраторы баз данных располагают небольшими окнами обслуживания, в рамках которых должны выполнять огромные объемы операций. Параллельный DDL предоставляет администраторам баз данных возможность в полной мере задействовать все доступные ресурсы компьютера, завершая крупные, сложные операции значительно быстрее, чем при их последовательном выполнении.

Глава завершается рассмотрением процедурного параллелизма — средств параллельного выполнения кода приложения. Здесь будут освещены две технологии. Первая из них — параллельные конвейерные функции, или способность Oracle динамически и параллельно выполнять хранимые функции. Вторая технология — "самодельный" параллелизм (do-it-yourself — DIY), когда приложение проектируется так, чтобы оно могло выполняться параллельно.

Глава 15. Загрузка и выгрузка данных

Первая половина этой главы посвящена загрузчику SQL*Loader (SQLLDR) и различным способам использования этого инструмента для загрузки и изменения данных в базе. Рассматриваются такие вопросы, как загрузка данных с разделителями, обновле-

ние существующих и вставка новых строк, выгрузка данных и вызов SQLLDR из хранимой процедуры. SQLLDR — это тщательно продуманный и очень важный инструмент, но с его практическим применением связано множество вопросов. Во второй части главы внимание сосредоточено на внешних таблицах — альтернативном и очень эффективном средстве массовой загрузки и выгрузки данных.

Глава 16. Шифрование данных

В этой главе рассматриваются возможности по шифрованию данных в базах Oracle. В нее включено обсуждение "самодельного" шифрования с применением встроенного пакета базы данных `DBMS_CRYPTO`, но упор на нем не делается. Вместо этого приводятся объяснения, почему не стоит использовать этот пакет. Основное внимание в этой главе сосредоточено на деталях реализации TDE (Transparent Data Encryption — прозрачное шифрование данных) в СУБД Oracle. В главе показано, как осуществляется шифрование на уровне столбца и табличного пространства, и что это означает для разработчиков и администраторов баз данных. Здесь описаны далеко не все возможные конфигурации (для этого существует документация Oracle), а лишь детали практической реализации вместе с их влиянием в конкретной ситуации.

Исходный код и обновления

В процессе ознакомления с примерами, приведенными в этой книге, вы можете предпочесть вводить весь код вручную. Многие так поступают именно потому, что это хороший способ ознакомления с используемыми технологиями создания кода.

Но, как бы то ни было, все исходные коды, приведенные в этой книге, доступны на сайте издательства. Если вы предпочитаете вводить код вручную, файлы исходных кодов можно использовать для проверки ожидаемых результатов — их следует применять в первую очередь, когда есть подозрение, что код был введен с ошибкой. Если же не хотите вводить код вручную, то загрузка исходного кода с веб-сайта издательства просто обязательна! В любом случае файлы кода существенно облегчат обновление и отладку.

От издательства

Вы, читатель этой книги, и есть главный ее критик и комментатор. Мы ценим ваше мнение и хотим знать, что было сделано нами правильно, что можно было сделать лучше и что еще вы хотели бы увидеть изданным нами. Нам интересно услышать и любые другие замечания, которые вам хотелось бы высказать в наш адрес.

Мы ждем ваших комментариев и надеемся на них. Вы можете прислать нам бумажное или электронное письмо, либо просто посетить наш веб-сервер и оставить свои замечания там. Одним словом, любым удобным для вас способом дайте нам знать, нравится или нет вам эта книга, а также выскажите свое мнение о том, как сделать наши книги более интересными для вас.

Посылая письмо или сообщение, не забудьте указать название книги и ее авторов, а также ваш обратный адрес. Мы внимательно ознакомимся с вашим мнением и обязательно учтем его при отборе и подготовке к изданию последующих книг.

Наши координаты:

E-mail: `info@williamspublishing.com`

WWW: `http://www.williamspublishing.com`

Информация для писем из:

России: 127055, г. Москва, ул. Лесная, д. 43, стр. 1

Украины: 03150, Киев, а/я 152

Настройка среды

В этом разделе будет показано, как настроить среду, чтобы в ней можно было выполнять приведенные в книге примеры кода. В частности, рассматриваются следующие вопросы.

- Корректная настройка демонстрационной схемы SCOTT/TIGER.
- Необходимая действующая среда.
- Конфигурирование средства AUTOTACE в SQL*Plus.
- Установка пакета Statspack.
- Установка и запуск runstats и других нестандартных утилит, используемых в примерах книги.
- Соглашения по кодированию, используемые в этой книге.

Все сценарии, не относящиеся к поставляемым компанией Oracle, доступны для загрузки на веб-сайте издательства.

Настройка схемы SCOTT/TIGER

Схема SCOTT/TIGER часто уже присутствует в базе данных. Как правило, она включена в обычную установку, но не является обязательным компонентом базы данных. Схему примера SCOTT можно установить для любой учетной записи базы данных — использование учетной записи SCOTT не обусловлено никакими особыми причинами. При желании таблицы EMP/DEPT можно установить непосредственно для своей учетной записи базы данных.

Большинство примеров в этой книге построено на таблицах в схеме SCOTT. Чтобы опробовать такие примеры, нужно иметь эти таблицы. Если вы работаете с разделяемой базой данных, целесообразно установить собственную копию этих таблиц от имени учетной записи, отличной от SCOTT, что позволит избежать побочных эффектов, возникающих из-за того, что другие пользователи затрагивают те же самые данные.

Выполнение сценария

Чтобы создать демонстрационные таблицы схемы SCOTT, выполните перечисленные ниже шаги.

- Ввести команду `cd [ORACLE_HOME]/sqlplus/demo`
- После подключения от имени любого пользователя запустить на выполнение сценарий `demobld.sql`

На заметку! В Oracle 10g и последующих версиях демонстрационные подкаталоги должны устанавливаться из сопровождающего компакт-диска (Companion CD). Ниже будут приведены необходимые компоненты файла `demobld.sql`.

Сценарий `demobld.sql` создаст и заполнит данными пять таблиц. По завершении он автоматически выйдет из SQL*Plus, поэтому не удивляйтесь, когда окно SQL*Plus исчезнет с экрана после запуска сценария — так было задумано.

В стандартных демонстрационных таблицах никаких ограничений ссылочной целостности не определено. Однако некоторые приведенные в книге примеры полагаются на ссылочную целостность. Поэтому после выполнения сценария `demobld.sql` рекомендуется выполнить также следующие операторы:

```
alter table emp add constraint emp_pk primary key(empno);
alter table dept add constraint dept_pk primary key(deptno);
alter table emp add constraint emp_fk_dept
                          foreign key(deptno) references dept;
alter table emp add constraint emp_fk_emp foreign key(mgr) references emp;
```

На этом установка демонстрационной схемы завершена. Если когда-либо потребуется удалить эту схему для освобождения места на диске, понадобится просто запустить сценарий `[ORACLE_HOME]/sqlplus/demo/demodrop.sql`. Он удалит пять таблиц и выйдет из SQL*Plus.

Создание схемы без сценария

Если доступ к файлу `demobld.sql` отсутствует, для выполнения приведенных в книге примеров достаточно будет выполнить следующий код:

```
CREATE TABLE EMP
(EMPNO NUMBER(4) NOT NULL,
 ENAME VARCHAR2(10),
 JOB VARCHAR2(9),
 MGR NUMBER(4),
 HIREDATE DATE,
 SAL NUMBER(7, 2),
 COMM NUMBER(7, 2),
 DEPTNO NUMBER(2)
);

INSERT INTO EMP VALUES (7369, 'SMITH', 'CLERK', 7902,
TO_DATE('17-DEC-1980', 'DD-MON-YYYY'), 800, NULL, 20);
INSERT INTO EMP VALUES (7499, 'ALLEN', 'SALESMAN', 7698,
TO_DATE('20-FEB-1981', 'DD-MON-YYYY'), 1600, 300, 30);
INSERT INTO EMP VALUES (7521, 'WARD', 'SALESMAN', 7698,
TO_DATE('22-FEB-1981', 'DD-MON-YYYY'), 1250, 500, 30);
INSERT INTO EMP VALUES (7566, 'JONES', 'MANAGER', 7839,
TO_DATE('2-APR-1981', 'DD-MON-YYYY'), 2975, NULL, 20);
INSERT INTO EMP VALUES (7654, 'MARTIN', 'SALESMAN', 7698,
TO_DATE('28-SEP-1981', 'DD-MON-YYYY'), 1250, 1400, 30);
INSERT INTO EMP VALUES (7698, 'BLAKE', 'MANAGER', 7839,
TO_DATE('1-MAY-1981', 'DD-MON-YYYY'), 2850, NULL, 30);
INSERT INTO EMP VALUES (7782, 'CLARK', 'MANAGER', 7839,
TO_DATE('9-JUN-1981', 'DD-MON-YYYY'), 2450, NULL, 10);
INSERT INTO EMP VALUES (7788, 'SCOTT', 'ANALYST', 7566,
TO_DATE('09-DEC-1982', 'DD-MON-YYYY'), 3000, NULL, 20);
INSERT INTO EMP VALUES (7839, 'KING', 'PRESIDENT', NULL,
TO_DATE('17-NOV-1981', 'DD-MON-YYYY'), 5000, NULL, 10);
INSERT INTO EMP VALUES (7844, 'TURNER', 'SALESMAN', 7698,
TO_DATE('8-SEP-1981', 'DD-MON-YYYY'), 1500, 0, 30);
INSERT INTO EMP VALUES (7876, 'ADAMS', 'CLERK', 7788,
TO_DATE('12-JAN-1983', 'DD-MON-YYYY'), 1100, NULL, 20);
```

```
INSERT INTO EMP VALUES (7900, 'JAMES', 'CLERK', 7698,
TO_DATE('3-DEC-1981', 'DD-MON-YYYY'), 950, NULL, 30);
INSERT INTO EMP VALUES (7902, 'FORD', 'ANALYST', 7566,
TO_DATE('3-DEC-1981', 'DD-MON-YYYY'), 3000, NULL, 20);
INSERT INTO EMP VALUES (7934, 'MILLER', 'CLERK', 7782,
TO_DATE('23-JAN-1982', 'DD-MON-YYYY'), 1300, NULL, 10);

CREATE TABLE DEPT
(DEPTNO NUMBER(2),
 DNAME VARCHAR2(14),
 LOC VARCHAR2(13)
);

INSERT INTO DEPT VALUES (10, 'ACCOUNTING', 'NEW YORK');
INSERT INTO DEPT VALUES (20, 'RESEARCH', 'DALLAS');
INSERT INTO DEPT VALUES (30, 'SALES', 'CHICAGO');
INSERT INTO DEPT VALUES (40, 'OPERATIONS', 'BOSTON');
```

Если вы создадите схему, выполнив приведенные выше команды, не забудьте вернуться к предыдущему подразделу и выполнить команды, создающие ограничения.

Настройка среды

Большинство из приведенных в этой книге примеров предназначено для выполнения полностью в среде SQL*Plus. Поэтому настройки и конфигурирования требует только SQL*Plus. Однако у меня имеется совет относительно использования этой среды. Почти во всех примерах тем или иным образом применяется `DBMS_OUTPUT`. Для работы `DBMS_OUTPUT` должна быть выполнена следующая команда SQL*Plus:

```
SQL> set serveroutput on
```

Необходимость постоянного ввода этой команды очень скоро станет раздражающей обузой. К счастью, имеется возможность создать файл `login.sql` — сценарий, выполняемый при каждом запуске SQL*Plus. Вдобавок можно определить переменную среды `SQLPATH`, которая позволит находить этот сценарий независимо от того, в каком каталоге он хранится.

Для всех примеров в этой книги используется сценарий `login.sql` такого вида:

```
define _editor=vi
set serveroutput on size 1000000
set trimspool on
set long 5000
set linesize 100
set pagesize 9999

column plan_plus_exp format a80
column global_name new_value gname

set termout off
define gname=idle
column global_name new_value gname

select lower(user) || '@' || substr( global_name, 1, decode( dot, 0,
length(global_name), dot-1) ) global_name
  from (select global_name, instr(global_name,'.') dot from global_name );

set sqlprompt '&gname> '
set termout on
```

Ниже описаны действия, выполняемые этими командами.

- `define _editor=vi`. Определяет текстовый редактор, который SQL*Plus будет использовать по умолчанию. Допускается указывать любой текстовый редактор (но не текстовый процессор), такой как Notepad или emacs.
- `set serveroutput on size 1000000`. Включает `DBMS_OUTPUT` по умолчанию (таким образом, не понадобится постоянно вводить команду `set serveroutput on`). Также устанавливает максимально доступный размер используемого по умолчанию буфера.
- `set trimspool on`. При буферизации текста из строк будут удаляться пробелы, а размер строк будет переменным. Если этот параметр установлен в `off` (по умолчанию), ширина буферизованных строк будет равна значению параметра `linesize`.
- `set long 5000`. Устанавливает количество байт, отображаемых при выборе столбцов `LONG` и `CLOB`.
- `set linesize 100`. Устанавливает ширину строк, отображаемых SQL*Plus, равной 100 символам.
- `set pagesize 9999`. Устанавливает параметр `pagesize`, управляющий частотой вывода заголовков SQL*Plus, в достаточно большое значение (в результате для каждой страницы будет выводиться один набор заголовков).
- `column plan_plus_exp format a80`. Определяет заданную по умолчанию ширину строки вывода плана выполнения, получаемого с помощью `AUTOTRACE`. В общем случае значения `a80` вполне достаточно для отображения всего плана.

Следующий фрагмент сценария `login.sql` определяет вид приглашения SQL*Plus:

```
define gname=idle
column global_name new_value gname
select lower(user) || '%' || substr( global_name, 1, decode( dot, 0,
length(global_name), dot-1) ) global_name
  from (select global_name, instr(global_name,'.') dot from global_name );
set sqlprompt '&gname> '
```

Директива `column global_name new_value gname` указывает SQL*Plus о необходимости помещения последнего значения, полученного для любого столбца по имени `global_name`, в переменную подстановки `gname`. Затем из базы данных выбирается `global_name` и объединяется с указанным при входе в систему именем пользователя. В результате приглашение интерфейса приобретает следующий вид:

```
ops$tkyte%ora11gr2>
```

Это позволяет видеть имя пользователя и базы данных.

Настройка средства AUTOTRACE в среде SQL*Plus

`AUTOTRACE` — это средство SQL*Plus, которое выводит план выполнения запущенных запросов и сведения об использованных ими ресурсах. В этой книге `AUTOTRACE` применяется очень часто. Существует несколько способов конфигурирования этого средства.

Начальная установка

Я предпочитаю выполнять настройку `AUTOTRACE` следующим образом:

- Ввести команду `cd [ORACLE_HOME]/rdbms/admin`

- Войти в SQL*Plus с учетной записью SYSTEM
- Ввести команду @utlxplan
- Ввести команду CREATE PUBLIC SYNONYM PLAN_TABLE FOR PLAN_TABLE;
- Ввести команду GRANT ALL ON PLAN_TABLE TO PUBLIC;

При желании вместо PUBLIC в команде GRANT можно указать конкретного пользователя. Определяя таблицу PLAN_TABLE как общедоступную, вы позволяете любому пользователю выполнять трассировку с помощью SQL*Plus (что, по моему мнению, неплохо). Это избавляет всех пользователей от необходимости установки собственной таблицы планов. Альтернативой может быть запуск команды @utlxplan в каждой схеме, из которой требуется использовать AUTOTRACE.

Далее понадобится создать и назначить роль PLUSTRACE:

- Ввести команду cd [ORACLE_HOME]/sqlplus/admin
- Войти в SQL*Plus с учетной записью SYS или SYSDBA
- Ввести команду @plustrce
- Ввести команду GRANT PLUSTRACE TO PUBLIC;

Как и ранее, вместо PUBLIC в команде GRANT можно указать имя конкретного пользователя.

Управление отчетом

Можно автоматически получать отчет по пути выполнения, использованном оптимизатором SQL, и статистикой по выполнению операторов. Отчет генерируется после успешного выполнения операторов SQL DML (т.е, SELECT, DELETE, UPDATE, MERGE и INSERT). Он полезен для отслеживания и настройки производительности перечисленных операторов.

Отчетом можно управлять посредством настройки системной переменной AUTOTRACE.

- SET AUTOTRACE OFF. Отчет AUTOTRACE не генерируется. Это значение установлено по умолчанию.
- SET AUTOTRACE ON EXPLAIN. Отчет AUTOTRACE будет отображать только путь выполнения оптимизатора.
- SET AUTOTRACE ON STATISTICCS. Отчет AUTOTRACE будет отображать только статистику по выполнению SQL-операторов.
- SET AUTOTRACE ON. Отчет AUTOTRACE будет содержать путь выполнения оптимизатора и статистику по выполнению SQL-операторов.
- SET AUTOTRACE TRACEONLY. Подобна SET AUTOTRACE ON, но подавляет вывод сведений о запросах пользователей, если таковые существуют.

Настройка пакета Statspack

Пакет Statspack должен устанавливаться при подключении к базе данных с учетной записью SYSDBA (CONNECT / AS SYSDBA). Чтобы его установить, необходимо иметь возможность подключаться с ролью SYSDBA. Во многих средах установка Statspack требует участия администратора базы данных.

При наличии возможности подключения установка Statspack не представляет сложности. Нужно просто запустить сценарий @spcreate.sql. Он находится в каталоге [ORACLE_HOME]\rdbms\admin и должен быть запущен после подключения от имени SYSDBA через SQL*Plus.

Перед запуском сценария spcreate.sql необходимо знать три единицы информации:

- Пароль, который вы хотите использовать для создаваемой схемы PERFSTAT
- Табличное пространство по умолчанию для схемы PERFSTAT
- Временное табличное пространство, которое должно использовать для схемы PERFSTAT

Запуск этого сценария дает примерно следующий вывод:

```
$ sqlplus / as sysdba
SQL*Plus: Release 11.2.0.1.0 Production on Fri May 28 10:52:52 2010
Copyright (c) 1982, 2009, Oracle. All rights reserved.
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

sys%ORA11GR2> @spcreate

Choose the PERFSTAT user's password
-----------------------------------
Not specifying a password will result in the installation FAILING
Enter value for perfstat_password:
... <дальнейший вывод для краткости опущен> ...
```

В процессе выполнения сценарий запросит необходимую информацию. В случае опечатки или случайного прерывания установки перед повторной установкой Statspack следует с помощью сценария spdrop.sql, находящегося в $ORACLE_HOME/rdbms/admin, удалить пользователя и установленные представления. Программа установки Statspack создаст файл spcpkg.lis. Его необходимо просмотреть на предмет выяснения любых возможных ошибок, имевших место во время установки. Установка пользователя, представлений и кода PL/SQL должна выполняться без каких-либо проблем, но только при указании допустимых имен табличных пространств (и отсутствии пользователя PERFSTAT).

Специальные сценарии

В этом разделе описываются требования (если есть), выполнение которых необходимо для различных сценариев, используемых в книге. Приводится также их код.

Runstats

Runstats — это инструмент, который я разработал для сравнения двух различных методов выполнения одних и тех же действий и выбора лучшего из них. Достаточно передать утилите два различных метода, а она выполнит все остальное. Этот сценарий производит три следующих ключевых измерения.

- *Время в формате часов, минут и секунд или истекшее время*. Эта информация полезна, но не является самой важной.
- *Статистика системы*. Отображает в расположенных рядом столбцах количество повторений различных операций (например, вызовов синтаксического анализатора) каждым из методов и различия между ними.
- *Сведения о защелках*. Это основная часть данного отчета.

Как будет показано в книге, защелка представляет собой облегченную блокировку. Блокировки — это механизмы сериализации, которые подавляют параллелизм. Приложения, подавляющие параллелизм, сужают возможности масштабирования, мо-

гут поддерживать меньше пользователей и требуют больше ресурсов. Одна из целей каждой разработки — построение приложений, обладающих потенциальной возможностью масштабирования — приложений, которые с равным успехом могут обслуживать и 1, 1000 и 10000 пользователей. Чем к меньшему числу защелок приводит данный подход, тем лучше. В большинстве случаев я предпочитаю подход, применение которого ведет к увеличению времени выполнения, но снижает показатель защелок до 10%. Такой подход обеспечит значительно более высокую масштабируемость, чем подход с большим количеством защелок.

Утилиту Runstats лучше использовать в изоляции, т.е. в однопользовательской базе данных. Мы будем измерять статистические показатели и вычислять уровни защелкивания (блокировки) данных, порождаемые применяемыми подходами. Поэтому весьма нежелательно, чтобы во время выполнения этих вычислений другие сеансы влияли на загрузку системы или уровень защелкивания. Для проведения подобных тестов удобно применять небольшую тестовую базу данных. Например, для этих целей я часто использую свой настольный ПК или лэптоп.

На заметку! Я убежден, что все разработчики должны иметь в своем распоряжении полностью контролируемую тестовую базу данных, в которой они смогут проверять свои идеи, не обращаясь постоянно за помощью к администратору базы данных. Вне всяких сомнений, база данных должна быть установлена на настольном компьютере каждого разработчика, даже если условия лицензионного соглашения для персональной версии базы данных разработчика предполагает "использование только в целях разработки и тестирования, но не для развертывания". В этом случае разработчик ничем не рискует! Кроме того, я провел несколько неформальных опросов на ряде конференций и семинаров и выяснил, что почти каждый администратор баз данных начинал свою профессиональную деятельность в качестве разработчика. Опыт и знания, которые разработчик может получить, управляя собственной базой данных — имея возможность видеть, как она в действительности работает — в перспективе сулят значительные преимущества.

Чтобы использовать утилиту Runstats, необходимо настроить доступ к нескольким представлениям `V$`, создать таблицу для хранения статистики и создать пакет Runstats. Потребуется доступ к четырем таблицам `V$` (динамическим таблицам производительности): `V$STATNAME`, `V$MYSTAT`, `V$TIMER` и `V$LATCH`. Я использую следующее представление:

```
create or replace view stats
as select 'STAT...' || a.name name, b.value
     from v$statname a, v$mystat b
    where a.statistic# = b.statistic#
   union all
   select 'LATCH.' || name, gets
     from v$latch
   union all
   select 'STAT...Elapsed Time', hsecs from v$timer;
```

На заметку! Действительными именами объектов, к которым должен быть открыт доступ, являются `V_$STATNAME`, `V_$MYSTAT` и т.д. — имена объектов начинаются с `V_$`, а не `V$`. Имя `V$` — это синоним, указывающий на действительное представление с именем, начинающимся с `V_$`. Таким образом, `V$STATNAME` — синоним, указывающий на представление `V_$STATNAME`. У вас должен быть открыт доступ к этому представлению.

Полномочия `SELECT` для представлений `V$STATNAME`, `V$MYSTAT` и `V$LATCH` могут быть выданы непосредственно вам (в этом случае вы можете самостоятельно создавать представление), либо кто-то другой с полномочиями `SELECT` для этих объектов

может создать представление и затем предоставить вам полномочия SELECT на этом представлении.

После этого останется создать небольшую таблицу для сбора статистики:

```
create global temporary table run_stats
( runid varchar2(15),
  name varchar2(80),
  value int )
on commit preserve rows;
```

И, наконец, необходимо создать сам пакет для Runstats. Он содержит три простых вызова API-интерфейса:

- RS_START (запуск Runstats) — процедура, которая должна вызываться в начале теста Runstats;
- RS_MIDDLE — процедура, которая должна вызываться в середине теста;
- RS_STOP — процедура, завершающая выполнение сценария и выводящая отчет.

Спецификация этого пакета выглядит следующим образом:

```
ops$tkyte%ORA11GR2> create or replace package runstats_pkg
  2  as
  3      procedure rs_start;
  4      procedure rs_middle;
  5      procedure rs_stop( p_difference_threshold in number default 0 );
  6  end;
  7  /
Package created.
Пакет создан.
```

Параметр p_difference_threshold служит для управления объемом данных, выводимых по завершении работы пакета. Пакет Runstats собирает статистические данные и информацию о защелках для каждого запуска теста, а затем выводит отчет о количестве используемых каждым тестом (подходом) ресурсов и различиях между ними. Этот входной параметр можно применять для вывода статистических сведений и информации о защелках только тех тестов, различие между которыми превышает это число. По умолчанию это значение равно нулю, и выводятся все результаты.

Теперь рассмотрим тело пакета, процедуру за процедурой. Пакет начинается с объявления глобальных переменных. Они будут использоваться для записи времени выполнения каждого теста:

```
ops$tkyte%ORA11GR2> create or replace package body runstats_pkg
  2  as
  3
  4  g_start number;
  5  g_run1 number;
  6  g_run2 number;
  7
```

Затем следует процедура RS_START. Она просто очищает таблицу со статистическими сведениями, а затем заполняет ее статистикой “перед” и информацией о количестве защелок. После этого она захватывает текущее значение таймера, которое можно применять для вычисления затраченного времени с точностью до сотых долей секунды:

```
  8  procedure rs_start
  9  is
 10  begin
 11      delete from run_stats;
```

```
12
13      insert into run_stats
14      select 'before', stats.* from stats;
15
16      g_start := dbms_utility.get_time;
17  end;
18
```

Следующая процедура — RS_MIDDLE. Она просто записывает время, затраченное на выполнение первого теста, в переменную G_RUN1. Затем она вставляет текущий набор статистических данных и информацию о количестве защелок. Если вычесть эти значения из значений, которые ранее были сохранены во время выполнения процедуры RS_START, можно выяснить количество защелок, использованных первым методом, количество использованных курсоров (статистических показателей) и т.п.

И, наконец, процедура записывает время начала следующего теста:

```
19  procedure rs_middle
20  is
21  begin
22      g_run1 := (dbms_utility.get_time-g_start);
23
24      insert into run_stats
25      select 'after 1', stats.* from stats;
26      g_start := dbms_utility.get_time;
27
28  end;
29
```

Последней процедурой в пакете является RS_STOP. Ее задача состоит в выводе совокупного времени процессора на каждый запуск, а также вывода разницы между статистическими показателями по каждому из двух запусков (выводятся только те, что превышают пороговые значения):

```
30  procedure rs_stop(p_difference_threshold in number default 0)
31  is
32  begin
33      g_run2 := (dbms_utility.get_cpu_time-g_start);
34
35      dbms_output.put_line
36      ( 'Run1 ran in ' || g_run1 || ' cpu hsecs' );
37      dbms_output.put_line
38      ( 'Run2 ran in ' || g_run2 || ' cpu hsecs' );
39         if ( g_run2 <> 0 )
40         then
41      dbms_output.put_line
42      ( 'run 1 ran in ' || round(g_run1/g_run2*100,2) ||
43        '% of the time' );
44         end if;
45      dbms_output.put_line( chr(9) );
46
47      insert into run_stats
48      select 'after 2', stats.* from stats;
49
50      dbms_output.put_line
51      ( rpad( 'Name', 30 ) || lpad( 'Run1', 12 ) ||
52        lpad( 'Run2', 12 ) || lpad( 'Diff', 12 ) );
53
```

```
    for x in
    ( select rpad( a.name, 30 ) ||
           to_char( b.value-a.value, '999,999,999' ) ||
           to_char( c.value-b.value, '999,999,999' ) ||
           to_char( ( (c.value-b.value)-(b.value-a.value)),
                                  '999,999,999' ) data
       from run_stats a, run_stats b, run_stats c
      where a.name = b.name
        and b.name = c.name
        and a.runid = 'before'
        and b.runid = 'after 1'
        and c.runid = 'after 2'

        and abs( (c.value-b.value) - (b.value-a.value) )
             > p_difference_threshold
      order by abs( (c.value-b.value)-(b.value-a.value))
    ) loop
       dbms_output.put_line( x.data );
    end loop;

    dbms_output.put_line( chr(9) );
    dbms_output.put_line
    ( 'Run1 latches total versus runs -- difference and pct' );
    dbms_output.put_line
    ( lpad( 'Run1', 12 ) || lpad( 'Run2', 12 ) ||
      lpad( 'Diff', 12 ) || lpad( 'Pct', 10 ) );

    for x in
    ( select to_char( run1, '999,999,999' ) ||
           to_char( run2, '999,999,999' ) ||
           to_char( diff, '999,999,999' ) ||
           to_char( round( run1/decode( run2, 0,
                to_number(0), run2) *100,2 ), '99,999.99' ) || '%' data
      from ( select sum(b.value-a.value) run1, sum(c.value-b.value) run2,
                    sum( (c.value-b.value)-(b.value-a.value)) diff
               from run_stats a, run_stats b, run_stats c
              where a.name = b.name
                and b.name = c.name
                and a.runid = 'before'
                and b.runid = 'after 1'
                and c.runid = 'after 2'
                and a.name like 'LATCH%'
              )
    ) loop
       dbms_output.put_line( x.data );
    end loop;
end;

end;
/
Package body created.
Тело пакета создано.
```

Теперь все готово к использованию пакета Runstats. В качестве примера рассмотрим применение Runstats для выяснения того, какой метод эффективнее: одна групповая операция `INSERT` или построчная обработка. Начнем с определения двух таблиц, в

которые нужно вставить 1 000 000 строк (сценарий создания BIG_TABLE представлен в этом разделе ниже):

```
ops$tkyte%ORA11GR2> create table t1
  2  as
  3  select * from big_table.big_table
  4  where 1=0;
Table created.
Таблица создана.

ops$tkyte%ORA11GR2> create table t2
  2  as
  3  select * from big_table.big_table
  4  where 1=0;
Table created.
Таблица создана.
```

Теперь применим первый метод вставки записей: использование одного SQL-оператора. Начнем с вызова процедуры RUNSTATS_PKG.RS_START:

```
ops$tkyte%ORA11GR2> exec runstats_pkg.rs_start;
PL/SQL procedure successfully completed.
Процедура PL/SQL успешно завершена.

ops$tkyte%ORA11GR2> insert into t1
  2  select *
  3    from big_table.big_table
  4   where rownum <= 1000000;
1000000 rows created.
1000000 строк создано.

ops$tkyte%ORA11GR2> commit;
Commit complete.
Фиксация завершена.
```

Затем можно применить второй метод — построчную вставку данных:

```
ops$tkyte%ORA11GR2> exec runstats_pkg.rs_middle;
PL/SQL procedure successfully completed.
Процедура PL/SQL успешно завершена.

ops$tkyte%ORA11GR2> begin
  2          for x in ( select *
  3                       from big_table.big_table
  4                      where rownum <= 1000000 )
  5          loop
  6                    insert into t2 values X;
  7          end loop;
  8          commit;
  9  end;
 10  /
PL/SQL procedure successfully completed.
Процедура PL/SQL успешно завершена.
```

И, наконец, сгенерируем отчет:

```
ops$tkyte%ORA11GR2> exec runstats_pkg.rs_stop(1000000)
Run1 ran in 411 cpu hsecs
Run2 ran in 6192 cpu hsecs
run 1 ran in 6.64% of the time
```

```
Name                                    Run1          Run2          Diff
STAT...opened cursors cumulati           213     1,000,365     1,000,152
STAT...execute count                     213     1,000,372     1,000,159
LATCH.shared pool                      2,820     1,006,421     1,003,601
STAT...recursive calls                 3,256     1,014,103     1,010,847
STAT...physical read total byt   122,503,168   124,395,520     1,892,352
STAT...cell physical IO interc   122,503,168   124,395,520     1,892,352
STAT...physical read bytes       122,503,168   124,395,520     1,892,352
STAT...db block changes              110,810     2,087,125     1,976,315
STAT...file io wait time           5,094,828       438,102    -4,656,726
LATCH.cache buffers chains           571,469     5,510,416     4,938,947
STAT...undo change vector size     3,885,808    67,958,316    64,072,508
STAT...redo size                 120,944,520   379,497,588   258,553,068

Run1 latches total versus runs -- difference and pct
Run1        Run2        Diff        Pct
804,522   6,840,599   6,036,077    11.76%

PL/SQL procedure successfully completed.
Процедура PL/SQL успешно завершена.
```

Это подтверждает, что пакет `RUNSTATS_PKG` установлен, и показывает, почему при разработке приложений следует, где только возможно, использовать один оператор SQL вместо порции процедурного кода!

Mystat

Сценарий `mystat.sql` и связанный с ним сценарий `mystat2.sql` служат для отображения увеличения некоторых "статистических показателей" Oracle после выполнения некоторых операций. Сценарий `mystat.sql` просто записывает начальные значения показателей:

```
set echo off
set verify off
column value new_val V
define S="&1"

set autotrace off
select a.name, b.value
from v$statname a, v$mystat b
where a.statistic# = b.statistic#
and lower(a.name) like '%' || lower('&S')||'%'
/
set echo on
```

Сценарий `mystat2.sql` отображает разницу между начальным и конечным значением (`&V` заполняется при запуске первого сценария `mystat.sql` — он использует для этого средство `NEW_VAL` из SQL*Plus, которое содержит последнее значение `VALUE`, выбранное предыдущим запросом):

```
set echo off
set verify off
select a.name, b.value V, to_char(b.value-&V,'999,999,999,999') diff
from v$statname a, v$mystat b
where a.statistic# = b.statistic#
and lower(a.name) like '%' || lower('&S')||'%'
/
set echo on
```

Например, чтобы выяснить, сколько информации redo сгенерировано определенной операцией UPDATE, можно выполнить следующий код:

```
big_table%ORA11GR2> @mystat "redo size"
big_table%ORA11GR2> set echo off

NAME                                 VALUE
-----------------------------  ----------
redo size                               496

big_table%ORA11GR2> update big_table set owner = lower(owner)
  2  where rownum <= 1000;
1000 rows updated.
1000 строк обновлено.

big_table%ORA11GR2> @mystat2
big_table%ORA11GR2> set echo off

NAME                                   V  DIFF
-----------------------------  ----------  ----------------
redo size                           89592            89,096
```

Как видите, оператор UPDATE для 1000 строк сгенерировал 89 096 байтов информации redo.

SHOW_SPACE

Процедура SHOW_SPACE выводит подробную информацию об использовании пространства сегментами базы данных. Ее интерфейс выглядит следующим образом:

```
ops$tkyte%ORA11GR2> desc show_space
PROCEDURE show_space

 Argument Name       Type               In/Out         Default?
 ------------------  -----------------  ------------  -------------
 P_SEGNAME           VARCHAR2           IN
 P_OWNER             VARCHAR2           IN             DEFAULT
 P_TYPE              VARCHAR2           IN             DEFAULT
 P_PARTITION         VARCHAR2           IN             DEFAULT
```

Ниже описаны аргументы этой процедуры.

- P_SEGNAME. Имя сегмента (например, имя таблицы или индекса).
- P_OWNER. По умолчанию принимается текущий пользователь, но эту процедуру можно использовать для просмотра какой-то другой схемы.
- P_TYPE. По умолчанию этот аргумент принимает значение TABLE и представляет тип просматриваемого объекта. Например, оператор select distinct segment_type from dba_segments выводит список допустимых типов сегментов.
- P_PARTITION. Имя секции при просмотре пространства для секционированного объекта. Процедура SHOW_SPACE одновременно может отображать пространство только для одной секции.

При условии, что сегмент хранится в табличном пространстве ASSM, результат выполнения этой процедуры выглядит следующим образом:

```
big_table@ORA11GR2> exec show_space('BIG_TABLE');
Unformatted Blocks .....................               0
FS1 Blocks (0-25)  .....................               0
FS2 Blocks (25-50) .....................               0
FS3 Blocks (50-75) .....................               0
```

```
FS4 Blocks (75-100).....                    0
Full Blocks      .....                  14,469
Total Blocks.....                       15,360
Total Bytes.....                   125,829,120
Total MBytes.....                          120
Unused Blocks.....                         728
Unused Bytes.....                    5,963,776
Last Used Ext FileId.....                    4
Last Used Ext BlockId.....                43,145
Last Used Block.....                         296
PL/SQL procedure successfully completed.
Процедура PL/SQL успешно завершена.
```

Элементы вывода объясняются ниже.

- `Unformatted Blocks` (Неформатированных блоков). Количество блоков, которые выделены для таблицы и хранятся ниже маркера максимального уровня заполнения (high-water mark — HWM), но не используются. Сумма неформатированных и неиспользуемых блоков равна общему количеству блоков, выделенных для таблицы, но не используемых для хранения данных в объекте ASSM.
- `FS1 Blocks - FS4 Blocks` (Блоков FS1 — Блоков FS4). Количество форматированных блоков с данными. Диапазоны, указанные после имени, представляют степень пустоты каждого блока. Например, диапазон (0–25) представляет количество блоков, пустых на 0–25%.
- `Full Blocks` (Полных блоков). Количество блоков, заполненных настолько, что они больше не являются кандидатами для выполнения последующих вставок данных.
- `Total Blocks` (Всего блоков), `Total Bytes` (Всего байт), `Total Mbytes` (Всего Мбайт). Общий объем дискового пространства, выделенный для сегмента, измеряемый, соответственно, в блоках базы данных, байтах и мегабайтах.
- `Unused Blocks` (Неиспользуемых блоков), `Unused Bytes` (Неиспользуемых байт). Эти значения представляют объем дискового пространства, которое никогда не используется. Эти блоки выделены для сегмента, но в настоящее время хранятся выше HWM-маркера сегмента.
- `Last Used Ext BlockId` (Идентификатор блока последнего использованного экстента). Идентификатор файла, который содержит последний экстент с данными.
- `Last Used Ext BlockId` (Идентификатор блока последнего использованного экстента). Идентификатор блока начала последнего экстента; идентификатор блока внутри последнего использованного файла.
- `Last Used Block` (Последний использованный блок). Смещение идентификатора последнего использованного блока в последнем экстенте.

Когда процедура `SHOW_SPACE` применяется для просмотра сведений об объектах в табличных пространствах с ручным управлением пространством сегментов, вывод подобен следующему:

```
big_table@ORA11GR2> exec show_space( 'BIG_TABLE' )
Free Blocks.....                             1
Свободных блоков
Total Blocks.....                      147,456
Всего блоков
Total Bytes.....                 1,207,959,552
Всего байт
```

```
Total MBytes.....................          1,152
Всего Мбайт
Unused Blocks....................          1,616
Неиспользуемых блоков
Unused Bytes.....................     13,238,272
Неиспользуемых байт
Last Used Ext FileId.............              7
Идентификатор файла последнего использованного экстента
Last Used Ext BlockId............        139,273
Идентификатор блока последнего использованного экстента
Last Used Block..................          6,576
Последний использованный блок

PL/SQL procedure successfully completed.
Процедура PL/SQL успешно выполнена.
```

Этот отчет отличается от предыдущего только строкой `Free Blocks` (Свободных блоков) в начале. Это количество блоков в первой группе списков свободных блоков сегмента. Сценарий выводит сведения только об этой группе списков свободных блоков. Чтобы сценарий мог отображать сведения о нескольких группах списков свободных блоков, его понадобится изменить.

Соответствующий код с комментариями приведен ниже. Эта утилита представляет собой простой дополнительный уровень, действующий поверх API-интерфейса `DBMS_SPACE` в базе данных.

```
create or replace procedure show_space
( p_segname   in varchar2,
  p_owner     in varchar2 default user,
  p_type      in varchar2 default 'TABLE',
  p_partition in varchar2 default NULL )
-- Эта процедура использует идентификатор authid текущего пользователя,
-- чтобы этот пользователь мог запрашивать представления DBA_*, используя
-- полномочия, определенные в роли ROLE, и чтобы его установку можно было
-- выполнять один раз для каждой базы данных, а не для каждого пользователя.
authid current_user
as
    l_free_blks               number;
    l_total_blocks            number;
    l_total_bytes             number;
    l_unused_blocks           number;
    l_unused_bytes            number;
    l_LastUsedExtFileId       number;
    l_LastUsedExtBlockId      number;
    l_LAST_USED_BLOCK         number;
    l_segment_space_mgmt      varchar2(255);
    l_unformatted_blocks      number;
    l_unformatted_bytes       number;
    l_fs1_blocks number; l_fs1_bytes    number;
    l_fs2_blocks number; l_fs2_bytes    number;
    l_fs3_blocks number; l_fs3_bytes    number;
    l_fs4_blocks number; l_fs4_bytes    number;
    l_full_blocks number; l_full_bytes  number;

    -- Встроенная процедура для вывода аккуратно
    -- форматированных чисел с простыми метками.
    procedure p( p_label in varchar2, p_num in number )
    is
```

```
    begin
        dbms_output.put_line( rpad(p_label,40,'.') ||
        to_char(p_num,'999,999,999,999') );
    end;
begin
  -- Этот запрос выполняется динамически, чтобы данная процедура могла
  -- быть создана пользователем, имеющим доступ к DBA_SEGMENTS/TABLESPACES
  -- посредством роли, как обычно.
  -- ПРИМЕЧАНИЕ: во время выполнения вызывающая процедура
  -- ДОЛЖНА иметь доступ к этим двум представлениям!
  -- Этот запрос определяет, является ли данный объект объектом ASSM
  begin
    execute immediate
      'select ts.segment_space_management
         from dba_segments seg, dba_tablespaces ts
        where seg.segment_name = :p_segname
         and (:p_partition is null or
             seg.partition_name = :p_partition)
         and seg.owner = :p_owner
         and seg.tablespace_name = ts.tablespace_name'
        into l_segment_space_mgmt
       using p_segname, p_partition, p_partition, p_owner;
  exception
     when too_many_rows then
       dbms_output.put_line
       ( ' This must be a partitioned table, use p_partition => ');
       -- Эта таблица должна быть секционированной, используйте p_partition =>
       return;
  end;
  -- Если объект - табличное пространство ASSM, для получения информации
  -- о пространстве необходимо использовать этом вызов API-интерфейса.
  -- В противном случае применяется API-интерфейс FREE_BLOCKS для работы
  -- с сегментами, управляемыми вручную.
  if l_segment_space_mgmt = 'AUTO'
  then
    dbms_space.space_usage
    ( p_owner, p_segname, p_type, l_unformatted_blocks,
      l_unformatted_bytes, l_fs1_blocks, l_fs1_bytes,
      l_fs2_blocks, l_fs2_bytes, l_fs3_blocks, l_fs3_bytes,
      l_fs4_blocks, l_fs4_bytes, l_full_blocks, l_full_bytes,  p_partition);
    p( 'Unformatted Blocks ', l_unformatted_blocks );
    p( 'FS1 Blocks (0-25) ', l_fs1_blocks );
    p( 'FS2 Blocks (25-50) ', l_fs2_blocks );
    p( 'FS3 Blocks (50-75) ', l_fs3_blocks );
    p( 'FS4 Blocks (75-100)', l_fs4_blocks );
    p( 'Full Blocks ', l_full_blocks );
  else
    dbms_space.free_blocks(
      segment_owner => p_owner,
      segment_name => p_segname,
      segment_type => p_type,
      freelist_group_id => 0,
      free_blks => l_free_blks);
    p( 'Free Blocks', l_free_blks );
  end if;
```

```
    -- Затем используется API-вызов unused_space для получения остальной информации.
    dbms_space.unused_space
     ( segment_owner => p_owner,
       segment_name => p_segname,
       segment_type => p_type,
       partition_name => p_partition,
       total_blocks => l_total_blocks,
       total_bytes => l_total_bytes,
       unused_blocks => l_unused_blocks,
       unused_bytes => l_unused_bytes,
       LAST_USED_EXTENT_FILE_ID => l_LastUsedExtFileId,
       LAST_USED_EXTENT_BLOCK_ID => l_LastUsedExtBlockId,
       LAST_USED_BLOCK => l_LAST_USED_BLOCK );
       p( 'Total Blocks', l_total_blocks );
       p( 'Total Bytes', l_total_bytes );
       p( 'Total MBytes', trunc(l_total_bytes/1024/1024) );
       p( 'Unused Blocks', l_unused_blocks );
       p( 'Unused Bytes', l_unused_bytes );
       p( 'Last Used Ext FileId', l_LastUsedExtFileId );
       p( 'Last Used Ext BlockId', l_LastUsedExtBlockId );
       p( 'Last Used Block', l_LAST_USED_BLOCK );
end;
/
```

BIG_TABLE

Во многих примерах, приведенных в этой книге, используется таблица BIG_TABLE. В зависимости от используемой системы, эта таблица может содержать от одной до 4 миллионов записей, и ее размер может изменяться от 200 Мбайт до 800 Мбайт. Во всех случаях ее структура остается неизменной. Для создания таблицы BIG_TABLE я написал сценарий, который выполняет перечисленные ниже действия.

- Создает пустую таблицу на основе представления ALL_OBJECTS. Это словарное представление применяется для заполнения таблицы BIG_TABLE.
- Устанавливает для этой таблицы режим NOLOGGING. Это не обязательно и было сделано исключительно в целях повышения производительности. Применение режима NOLOGGING для тестовой таблицы совершенно безопасно. Однако его не следует использовать в рабочей системе, поскольку это приведет к отключению ряда средств, таких как Oracle Data Guard (Защита данных Oracle).
- Заполняет таблицу, используя содержимое представления ALL_OBJECTS в качестве начальных значений, а затем выполняя итеративную вставку таблицы в саму себя и при каждой итерации увеличивая ее размер примерно вдвое.
- Создает ограничение первичного ключа таблицы.
- Собирает статистику.

Чтобы построить таблицу BIG_TABLE, можно запустить следующий сценарий в командной строке SQL*Plus, передав ему количество строк таблицы в качестве параметра:

```
create table big_table
as
select rownum id, a.*
  from all_objects a
 where 1=0
/
```

```
alter table big_table nologging;
declare
    l_cnt number;
    l_rows number := &1;
begin
    insert /*+ append */
    into big_table
    select rownum, a.*
      from all_objects a
     where rownum <= &1;
    l_cnt := sql%rowcount;
    commit;
    while (l_cnt < l_rows)
    loop
        insert /*+ APPEND */ into big_table
        select rownum+l_cnt,
               OWNER, OBJECT_NAME, SUBOBJECT_NAME,
               OBJECT_ID, DATA_OBJECT_ID,
               OBJECT_TYPE, CREATED, LAST_DDL_TIME,
               TIMESTAMP, STATUS, TEMPORARY,
               GENERATED, SECONDARY, NAMESPACE, EDITION_NAME
          from big_table
         where rownum <= l_rows-l_cnt;
        l_cnt := l_cnt + sql%rowcount;
        commit;
    end loop;
end;
/
alter table big_table add constraint
big_table_pk primary key(id);
exec dbms_stats.gather_table_stats( user, 'BIG_TABLE', estimate_percent => 1);
```

Я собрал базовые статистические сведения о таблице. Для индекса, ассоциированного с первичным ключом, статистика собирается автоматически при его создании.

Соглашения в отношении кода

Одно из используемых в этой книге соглашений в отношении кода касается именования переменных в коде PL/SQL. Например, взгляните на следующее тело пакета:

```
create or replace package body my_pkg
as
   g_variable varchar2(25);
   procedure p( p_variable in varchar2 )
   is
      l_variable varchar2(25);
   begin
      null;
   end;
end;
/
```

Этот фрагмент кода содержит три переменных: глобальную переменную пакета `G_VCARIABLE`, формальный параметр процедуры `P_VARIABLE` и, наконец, локальную переменную `L_VARIABLE`. Имена переменным назначаются по названиям диапазонов, в которых они содержатся. Имена всех глобальных переменных начинаются с символов `G_`, имена параметров — с символов `P_`, а имена локальных переменных — с `L_`.

Основная цель такого именования — различение переменных PL/SQL и имен столбцов таблицы базы данных. Например, процедура вроде показанной ниже:

```
create procedure p( ENAME in varchar2 )
as
begin
   for x in ( select * from emp where ename = ENAME ) loop
      Dbms_output.put_line( x.empno );
   end loop;
end;
```

будет всегда выводить все строки таблицы `EMP`, в которых поле `ENAME` не является пустым. SQL встречает выражение `ename = ENAME` и (естественно) сравнивает столбец `ENAME` с самим собой. Можно было бы использовать выражение `ename = P.ENAME` — т.е. уточнить ссылку на переменную PL/SQL именем процедуры, но об этом легко забыть, что приведет к ошибке.

Поэтому я просто всегда называю свои переменные по названиям диапазонов. В результате я всегда могу легко отличить параметры от локальных и глобальных переменных, не говоря уже о том, что это исключает любые сомнения по поводу имен столбцов и имен переменных.