Содержание

Введение

Эту книгу должны прочитать те, кто знакомится с языком PL/SQL впервые. Часть I посвящена введению в PL/SQL. Часть II охватывает основы языка программирования, которые состоят из функций, процедур, пакетов, триггеров и больших объектов. В части III рассмотрены некоторые более сложные вопросы, что должно помочь при разработке ваших проектов.

Приложения в части IV дают вам основы знаний по таким темам, как администрирование базы данных Oracle (DBA), программирование на SQL, создание скриптов на PHP, разработка приложений на JAVA, регулярные выражения и упаковка текста (wrapping) в PL/SQL. Для первичного ознакомления дано краткое описание иерархического профайлера PL/SQL (PL/SQL Hierarchical Profiler) и средства PL/Scope. Кроме того, в приложении дан обзор зарезервированных слов и встроенных функций, которые помогают решать стоящие перед вами задачи.

Часть I: Основы PL/SQL

В часть I освящены текущие возможности базы данных Oracle Database 10g release 2 и новые возможности, появившиеся в Oracle Database 11g. Она содержит руководство для быстрого освоения языка и его составных частей – семантика языка, типы, управляющие структуры и обработка ошибок.

- Глава 1. Обзор PL/SQL, разъяснены основы PL/SQL. Кроме того, рассмотрены текущие возможности базы данных Oracle Database 10g release 2 и дополнения, появившиеся в Oracle Database 11g.
- Глава 2. Основы PL/SQL, дан краткий обзор приемов написания программ на PL/SQL. Она рассчитана на использование в качестве введения в язык.
- Глава 3. Описание языка посвящено семантике языка PL/SQL. Эта глава охватывает лексические единицы, структуру блоков, типы переменных и область видимости переменных.
- Глава 4. Управляющие структуры, объяснены структуры PL/SQL, используемые при условных переходах и в циклах. Кроме того, рас-

смотрена часть языка, касающаяся курсоров (включая системные курсорные переменные) и массовых операций.

- Глава 5. Обработка ошибок, объяснено как в PL/SQL выполняется обработка ошибок. Показано, как обрабатывать исключения, определять ваши пользовательские исключения и работать со стеком ошибок.

Часть II. Программирование на PL/SQL

В Части II дано введение в функции, процедуры, пакеты и триггеры. Кроме того, рассматривается тема больших объектов в Oracle (LOB). Все это составляет инструментарий для создания эффективных приложений базы данных.

- Глава 6. Функции и процедуры. В разделе объяснено, как создавать функции и процедуры. Содержатся примеры моделей с передачей параметров по значению и по ссылке, показано, как создавать функции детерминированные, пригодные к распараллеливанию, конвейерные и работающие с кэшем результатов. Кроме того, в ней затрагивается семантика вызовов – нотация позиционная, по имени и смешанная, а также автономные программные единицы.
- Глава 7. Коллекции. В разделе разъяснены v-массивы (VARRAY), вложенные таблицы и ассоциативные массивы (ранее известные как таблицы PL/SQL). Кроме того, рассматривается интерфейс API для работы с коллекциями и операторы для работы с коллекционными множествами. В этой главе даны примеры работы с числовыми и строковыми индексами для ассоциативных массивов.
- Глава 8. Большие объекты. Здесь разъяснены большие объекты, показано, как работать с ними, считывать их из файловой системы и записывать в базу данных. Кроме того, вы узнаете, как считывать и записывать их в ваших приложениях на PL/SQL или в web-приложениях, загружать с использованием web-страницы или скрипта на PHP.
- Глава 9. Пакеты. В разделе Показано, как создавать библиотеки родственных функций и процедур, как применять повторно инициализируемые и повторно не инициализируемые пакеты, как применять модель прав определившего и вызывающего лица. Кроме того, в этой главе есть раздел, посвященный поиску проверке корректности и получению описания пакетов в каталоге данных. Вы узнаете, как контролировать зависимости и сравните два метода проверки корректности: по метке времени и по сигнатуре.
- Глава 10. Триггеры. Здесь разъяснено, как применять триггеры в базе данных. В примерах рассмотрены триггеры DDL, DML, комбинированные (новинка в базе данных Oracle 11g), замещающие

(instead-of) и системные тригеры. Кроме того, эта глава содержит полный набор примеров по использованию функций атрибутов событий, которые используются при работе с триггерами.

Часть III. Расширенные возможности программирования на PL/SQL

Глава посвящена введению в динамический SQL, взаимодействию между сеансами, внешним процедурам, типам объектов, библиотекам Java и разработке web-приложений.

- Глава 11. Динамический SQL. Здесь разъяснено, как использовать собственный динамический SQL (Native Dynamic SQL) и пакет `DBMS_SQL` предыдущих версий. Примеры в этой главе иллюстрируют вызовы программ с предложениями, построенными динамически и протестированными новым пакетом `DBMS_ASSERT`. Приведены примеры использования замещаемых переменных или переменных связи, включая создание динамического списка замещаемых переменных.
- Глава 12. Взаимодействие между сеансами, показано, как использовать `DBMS_ALERT` и `DBMS_PIPE` для создания связи между двумя параллельными сеансами.
- Глава 13. Внешние процедуры. В разделе разъяснено, как использовать внешние процедуры, как создавать их во внешних библиотеках на C и Java. Кроме того, эта глава охватывает тему гетерогенного сервера Oracle (Oracle Heterogeneous server) и как сконфигурировать файл `listener.ora` для поддержки внешних процедур.
- Глава 14. Объектные типы. Здесь объяснено, как определить и использовать динамические объектные типы, как альтернативный подход к использованию пакетов. Вы узнаете, как создавать объектные типы и реализовывать тела объектов, как создавать подтипы. В главе показано, как создавать запросы и получать доступ к столбцам объектного типа в вашей базе данных.
- Глава 15. Библиотеки Java. Здесь показано, как создавать и разворачивать библиотеки Java внутри базы данных, объяснено как работают оболочки на PL/SQL, используемые в качестве интерфейса к вашим методам классов Java. Вы также узнаете новый метод взаимодействия с базой данных в Oracle 11g.
- Глава 16. Разработка web-приложений. В разделе показано, как писать процедуры PL/SQL для web-приложений и серверные страницы PL/SQL (PL/SQL Server Pages, PSP), как сконфигурировать и развернуть чистые web-приложения PL/SQL, используя автономный сервер Oracle HTTP Server и базу данных Oracle XML Database.

Часть IV. Приложения

Часть IV содержит ряд материалов для знакомства новичков с базой данных Oracle и родственными технологиями, включая задачи администрирования в Oracle (Oracle DBA), программирование на SQL, написание скриптов PHP, разработка программ на Java, регулярные выражения и упаковка PL/SQL. Кроме того, рассмотрены иерархический профайлер PL/SQL (Hierarchical Profiler PL/SQL), средство PL/Scope, зарезервированные слова и ряд ключевых встроенных функций.

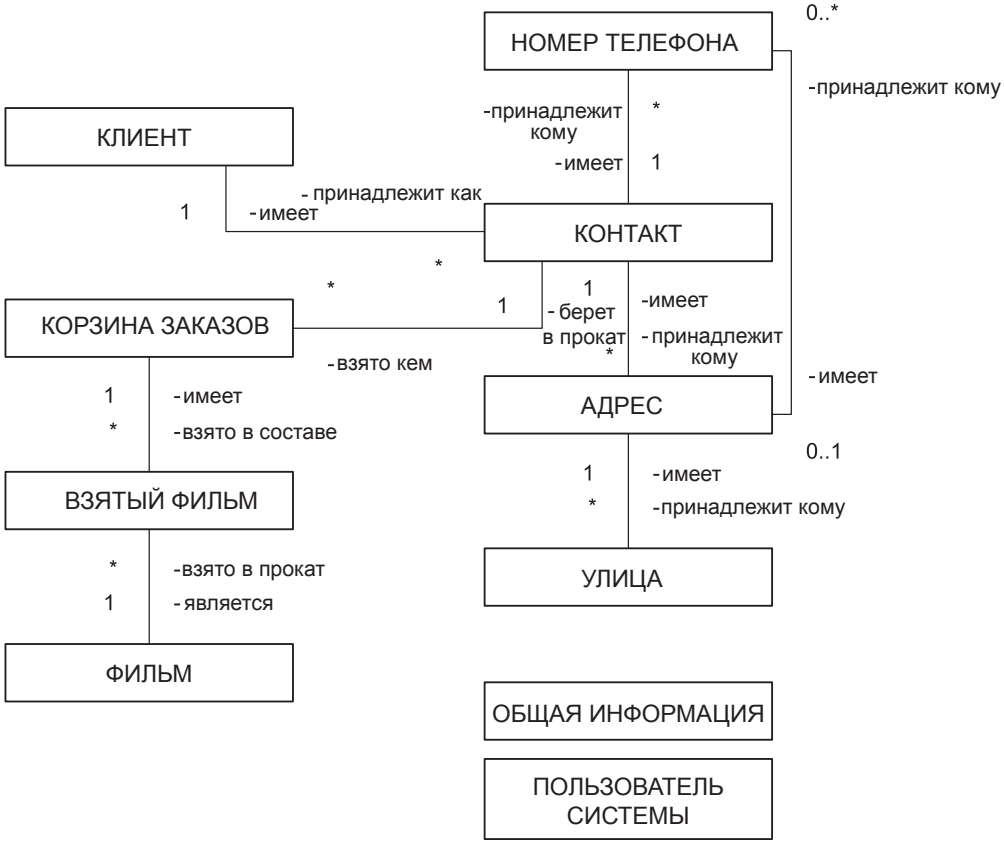
- Приложение A. Краткий курс администрирования в базе данных Oracle. Здесь разъяснено, как использовать интерфейс SQl*Plus, запускать и останавливать базу данных, запускать и останавливать процесс наблюдателя Oracle (Oracle listener).
- Приложение B. Краткий курс языка SQL в базе данных Oracle, начинается с описания применения языка SQL в Oracle, включая типы данных Oracle в SQl*Plus, затем рассмотрены команды языка SQL, необходимые для создания приложений базы данных: язык определения данных (Data Definition Language, DDL) язык манипулирования данными (Data Manipulating Language, DML), язык запроса данных (Data Query Language, DQL) и язык управления данными (Data Control Language, DCL).
- Приложение C. Краткий курс PHP. Здесь рассмотрены основы PHP, применение Zend Core для Oracle и как писать web-страницы на PHP, использующие базу данных Oracle 11g.
- Приложение D. Краткий курс Java для базы данных Oracle. В разделе рассмотрены основы языка программирования Java, порядок соединения с базой данных Oracle 11g с помощью Oracle JDBC, автономные приложения Java, которые будут работать с базой данных, включая использование больших объектов (`LOB`).
- Приложение E. Краткое описание регулярных выражений. Здесь разъяснено применение и использование регулярных выражений в базе данных Oracle 11g.
- Приложение F. Краткое описание упаковки кода PL/SQL (Wrapping PL/SQL `CODE`). В разделе показано как вы можете упаковать (закодировать) программы на PL/SQL, хранимые в базе данных для защиты программной логики от постороннего взгляда.
- Приложение G. Краткое описание иерархического профайлера PL/SQL (PL/SQL Hierarchical Profiler). Здесь объяснено как работает иерархический профайлер, даны примеры его использования.
- Приложение H. Средство PL/Scope. Здесь показано, как этот инструмент работает, дан краткий анализ его концепции.
- Приложение I. Зарезервированные и ключевые слова PL/SQL. Здесь объяснены зарезервированные и ключевые слова языка PL/SQL, показано, как определять их в каталоге данных.

- Приложение J. Встроенные функции PL/SQL. В разделе рассмотрено большое количество наиболее полезных встроенных функций, даны ключевые примеры применения этих функций, на эти примеры имеются ссылки из других частей книги.

Демонстрационная база данных: магазин видеофильмов

Большинство примеров в этой книге используют модель базы данных магазина видеофильмов, которую вы можете скачать с сайта издателя. Вы можете создать пользователя `plsql`, на которого мы ссылаемся в этой книге, используя скрипт `create_user.sql`. Вы можете создать модель базы данных, используя скрипт `create_store.sql`. Этот последний скрипт заполняет базу данных исходной информацией для обеспечения работы примеров, приводимых в данной книге.

На рисунке показаны связи между сущностями (ERD-диаграмма) этой модели



Часть I

ОСНОВЫ PL/SQL

ГЛАВА 1

ОБЗОР ЯЗЫКА ORACLE PL/SQL

Эта глава посвящена введению в язык процедур/структурированный язык запросов (Procedure Language/Structured Query Language, PL/SQL). В ней объясняются история, архитектура и структура блоков языка PL/SQL, дан обзор новых возможностей Oracle 10g и новых возможностей Oracle 11g. В этой главе имеются следующие разделы: История и происхождение, Архитектура, Структура основных блоков, Новые возможности Oracle 10g, Новые возможности Oracle 11g.

История и происхождение

Язык PL/SQL разработан корпорацией Oracle в конце 1980-х годов. Сначала PL/SQL имел ограниченные возможности, но все изменилось в начале 1990-х годов. PL/SQL снабдил базу данных Oracle встроенным интерпретирующим и не зависящим от операционной системы программным окружением. Операторы SQL естественно интегрированы в язык PL/SQL. Кроме того, вы можете вызывать PL/SQL напрямую из командной строки интерфейса SQI*Plus. Похожие прямые вызовы можно сделать в ваших внешних запросах к базе данных, написанных на иных языках программирования, как это показано в Приложениях C и D.

В базе данных Oracle 8 появилась возможность использовать объектные типы данных. Это перевело базу данных Oracle из чисто реляционной модели в объектно-реляционную (или расширенную реляционную) модель. Эти типы имели ограниченную ценность в качестве наборов (коллекций) скалярных переменных, пока они не получили возможность создавать экземпляры объектов в Oracle 9i release 2. Способность создавать экземпляры объектов SQL сделала внутренние объекты Oracle совместимыми с объектными типами C++, Java или C#. Объектные типы SQL применяются в PL/SQL и рассматриваются в главе 15.

Язык PL/SQL сделал шаг вперед, введя полноценные возможности объектно-ориентированного программирования, появившихся в Oracle 9i release 2. PL/SQL больше не является чисто процедурным языком. В настоящее время это одновременно процедурный и объектно-ориентированный язык программирования.

Следующий шаг был сделан в базе данных Oracle 11g путем изменения PL/SQL из интерпретирующего языка в язык, компилирующий в собственный (native) код. Вы можете спросить: "Не уничтожает ли это преимущества языка, не зависящего от операционной системы?". Ответ на этот вопрос полностью отрицательный. Сейчас вы можете на первом этапе писать процедуры на PL/SQL в форме, не зависящей от операционной системы. Затем вы можете развернуть их и позволить Oracle выполнить компиляцию в собственный код. Oracle 11g автоматизирует этот процесс на всех платформах.

Версии PL/SQL
Первые версии языка PL/SQL имели самостоятельную нумерацию, не связанную с нумерацией версий базы данных. Например, PL/SQL 1.0 поставлялся с базой данных Oracle 6. PL/SQL 2.x поставлялся с базой данных Oracle 7.x. Начиная с Oracle 8, версии PL/SQL корреспондируют с номерами релизов базы данных, например PL/SQL 11.1 в базе данных Oracle 11g release 1.

PL/SQL можно вызывать из внешних программ, и это основной шлюз для внешних библиотек. Наименование «внешняя библиотека» ни о чем не говорит, так как библиотеки Java могут храниться в базе данных. Oracle вызывает внешние процедуры внешних библиотек через PL/SQL независимо от того, где они хранятся. Программы PL/SQL служат оболочками для внешних библиотек. Оболочки являются интерфейсами, которые скрывают преобразование типов между базой данных и внешними программами.

Можно расширить функциональность базы данных Oracle 11g путем создания хранимых функций и процедур на PL/SQL, C, C++ или Java. Программы на Java могут напрямую сохраняться в базе данных Oracle 11g всех версий, кроме Oracle Express Edition. Глава 12 демонстрирует, как создавать и запускать внешние процедуры. Глава 14 показывает, как создавать и развертывать библиотеки Java в базе данных.

PL/SQL продолжает развиваться и становится все более мощным. Это очень удобно для тех, кто хорошо знает PL/SQL, а эволюция Java от версии к версии очень выгодна программистам, владеющим Java. Программирование на PL/SQL бросает вызов новичкам в этом языке, потому что на нем пишут очень многие знатоки базы данных Oracle. По мере того, как вы нарабатываете опыт в этом языке, вы узнаете, как применить PL/SQL для решения задач возрастающей сложности.

Архитектура

Язык PL/SQL мощный инструмент с многочисленными возможностями. PL/SQL позволяет вам, однажды написав код, развернуть его на базе

Является ли программирование на PL/SQL похоже на черную магию?

В начале PL/SQL 1.0 был инструментом для создания отчетов. В настоящее время оператор `CASE` в SQL охватывает большую часть той первоначальной функциональности. В середине 1990-х разработчики описывали программирование на версии 2.x как черную магию. Тогда это прозвище было справедливым. Об этом языке было мало написано, а доступность примеров кода в Интернете было ограничено, потому что Интернет, в привычном сегодня виде, еще не существовал.

Но и сегодня некоторые рассматривают программирование на PL/SQL как черную магию. Им больше нравится написать код, не зависящий от базы данных, на Java или других языках. Политически корректно говорить о желании отклонить решения на PL/SQL, невзирая на их преимущества. Почему до сих пор для многих Oracle PL/SQL остается черной магией, если на сегодня опубликовано столько книг, посвященных PL/SQL?

Вы скажете, что это из-за курсоров, но курсоры существуют в любой программе, соединяющей через интерфейс вызовы Oracle (Oracle Call Interface, OCI) или Java Database Connectivity (JDBC). Если не из-за курсоров, то можно попробовать обвинить синтаксис, типы данных, определяемые пользователем, или нюансы функций и процедур. А они действительно отличаются от тех, которые используются в других языках программирования? Если вы отвечаете отрицательно на этот вопрос, то готовы начать изучение мира PL/SQL. Если вы отвечаете положительно на этот вопрос или полагаете, что имеются иные темные стороны в языке, вы не готовы к его изучению.

С чего лучше начать изучение? Простой ответ – читать эту книгу. Более серьезный ответ – понять, что термины Oracle, которые используются и в языке PL/SQL, однозначные. Например, переменная – это всегда переменная некоторого типа, а функция или процедура – это всегда подпрограмма, которая обрабатывает формальные параметры, полученные по ссылке или по значению и, которая может возвращать или не возвращать в результате операнд корректного типа. Это типичные примеры простых правил, которые справедливы и для каждой компоненты языка.

данных, ближайшей к вашим данным. PL/SQL может упростить разработку приложений, оптимизировать выполнение и повысить степень использования ресурсов базы данных.

Этот язык программирования нечувствительный к регистру, как и SQL. Поэтому можно позволяет использовать многочисленные приемы наиболее удобного форматирования. Чтобы не приводить аргументы в пользу того или иного стиля, рекомендуем вам выработать свой собственный стиль, соответствующий стандартам вашей организации и впоследствии

применять этот стиль в работе. В этой книге для кода PL/SQL применяется следующее правило: ключевые слова пишутся прописными буквами, а переменные, имена столбцов и вызовы хранимых процедур – строчными.

PL/SQL разработан с применением моделирующих концепций структурного программирования, статической типизации данных, модульности, обработки исключений и параллельного (конкурентного) процессинга, которые заимствованы из языка программирования Ada. Язык программирования Ada, разработанный для министерства обороны США, был предназначен для использования во встроенных военных системах, применяемых на самолетах и в ракетах, которые должны быть системами реального времени и предельно надежными. Язык программирования Ada заимствовал многие значимые элементы синтаксиса из языка программирования Pascal, включая операторы присвоения, сравнения и ограничители в виде одинарных кавычек.

Эти решения позволили производить прямую вставку операторов SQL в блоки кода PL/SQL. Это важно, потому что SQL включает те же операторы языка Pascal, ограничители строк и декларативные скалярные типы данных. Pascal и Ada имеют декларируемые скалярные типы данных. Декларируемые типы данных не изменяются во время исполнения и известны как «сильные» типы данных. «Сильные» типы данных являются важными для тесной интеграции языков Oracle SQL и PL/SQL. PL/SQL поддерживает динамические типы данных путем сопоставления их во время исполнения типам, определенным в каталоге базы данных Oracle 11g. Операторы нахождения соответствий и ограничители строк означают упрощение синтаксического разбора, потому что предложения SQL естественно включены в программные единицы PL/SQL.

Примечание Примитивы в языке программирования Java описывают скалярные переменные, которые хранят только одно значение в данный момент.

Первая команда разработчиков PL/SQL тщательно продумала выбранные решения. База данных Oracle со временем завоевала авторитет благодаря этим решениям. Одно из этих решений, которое кажется превосходным, позволяет привязывать переменные PL/SQL к каталогу базы данных. Это своего рода наследование типов во время исполнения. Вы можете использовать псевдотипы `%TYPE` и `%ROWTYPE` для наследования переменных «сильного» типа, определенных в каталоге базы данных (этот вопрос рассматривается в главах 3 и 9).

Привязывание переменных PL/SQL к объектам каталога базы данных является эффективной формой структурирования. Это может минимизировать объем изменений, которые нужно внести в программы на PL/SQL. Наконец, это ограничивает частоту переработки кода в результате изменений базовых типов (например, изменение `VARCHAR2` в `DATE`) и не переопределять размеры переменных. Например, не нужно модифицировать код при изменении размера строк в столбце со строками переменной длины.

Кроме того, корпорация Oracle приняла и другое стратегическое решение, когда ограничила количество базовых типов данных SQL, и ввела понятие подтипа в каталоге базы данных. Возможность создавать подтипы базовых типов позволила корпорации Oracle создать дерево объектов с многоуровневой иерархией, которое продолжает расти и развиваться. Объектно-ориентированный подход к проектированию позволяет корпорации Oracle преобразовывать реляционную модель в объектно-реляционную (известную как расширенная реляционная модель). PL/SQL позволяет получить все выгоды от введения подтипов к базовым типам переменных SQL.

Интерпретатор PL/SQL существует как ресурс внутри окружения SQL*Plus. Окружение SQL*Plus является одновременно и интерактивное, и вызываемое. Каждый раз, когда вы присоединяетесь к базе данных Oracle 11g, база данных создает новый сеанс. В этом сеансе можно запускать на выполнение операторы SQL и PL/SQL из окружения SQL*Plus.

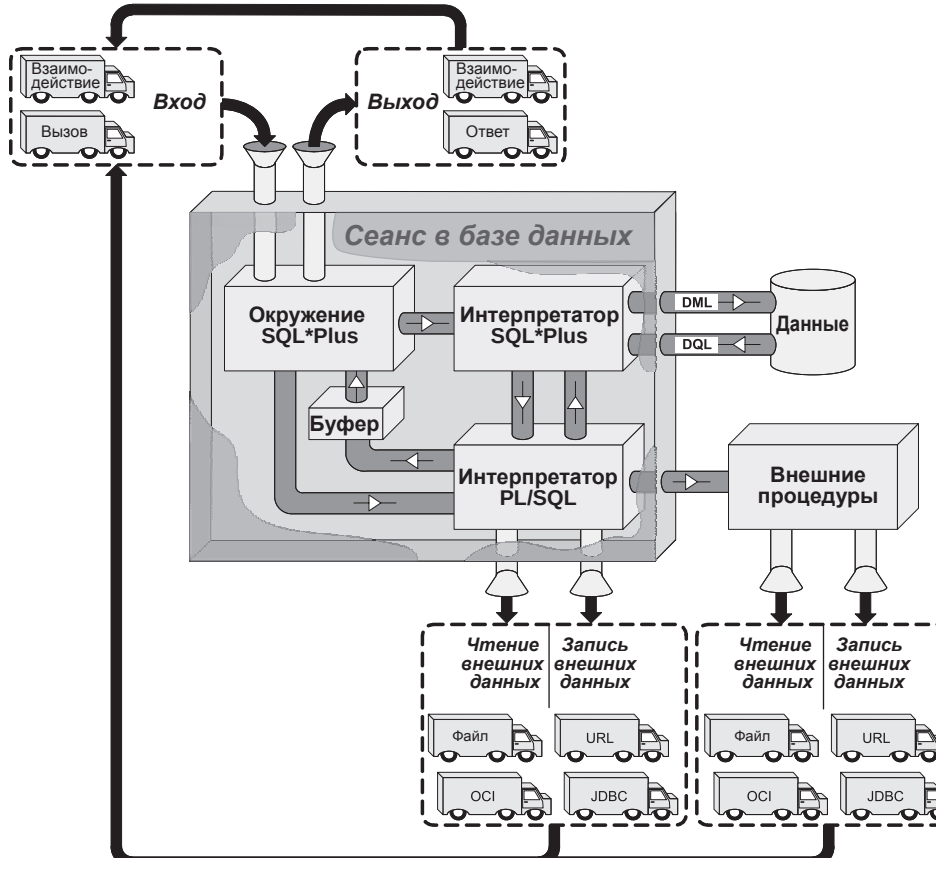


Рис. 1.1 *Архитектура обработки данных в базе*

Далее программная единица PL/SQL может запустить на выполнение операторы SQL или внешние процедуры, как это показано на Рис. 1.1. Предложения SQL также могут вызывать хранимые функции и процедуры PL/SQL. Операторы SQL напрямую взаимодействуют с реальными данными.

Прямые вызовы PL/SQL можно сделать через интерфейс вызовов Oracle (Oracle Call Interface, OCI) или Java Database Connectivity (JDBC). Это важно, потому что вы можете задавать границы транзакций в хранимых процедурах PL/SQL. Это чрезвычайно упрощает мириады задач, которые часто появляются в приложениях на слое абстрагирования данных.

Кроме того, PL/SQL поддерживает конструирование предложений SQL во время исполнения. Предложения SQL, создаваемые во время исполнения, называются динамическим SQL. Вы можете использовать два подхода к динамическому SQL: один – собственный динамический SQL (Native Dynamic SQL, NDS), другой – пакет `DBMS_SQL`. В базу данных Oracle 11g введены новые возможности NDS и повышена скорость выполнения. В этом релизе нужно использовать только пакет `DBMS_SQL`, если вы не знаете количество столбцов, к которым нужно обратиться в динамическом запросе SQL. В Главе 11 демонстрируется динамический SQL и разбирается как NDS, так и пакет `DBMS_SQL`.

Итак, у вас имеется высокоуровневый обзор языка PL/SQL. В следующем разделе дан краткий обзор структуры блоков PL/SQL.

Структура основных блоков

PL/SQL блочный язык программирования. Программные единицы могут быть именованными или неименованными блоками. Неименованные блоки известны как анонимные и будут так именоваться дальше в этой книге. Стиль кодирования на PL/SQL отличается от стиля программирования, принятого в языках C, C++ и Java. Например, фигурные скобки не применяются для ограничения блоков в PL/SQL.

В некоторых ситуациях эффективны программы из анонимных блоков. Обычно вы применяете анонимные блоки при создании скриптов для первоначального заполнения базы данных или выполнения разовых работ. Кроме того, они эффективны, если нужно произвести вложение некоторого кода в исполняемую секцию другого блока SQL. Базовая структура анонимного блока должна содержать секцию исполняемых команд. Кроме того, можно ввести в секции объявлений переменных и обработки исключений анонимные блоки. Следующий пример иллюстрирует анонимный блок:

-
```
[DECLARE]
список объявлений переменных
BEGIN
выполняемые команды
```

```
[EXCEPTION]
команды обработки исключений
END;
/
```

Блок объявлений позволяет определить типы данных, структуры и переменные. Определение переменной означает, что вы присваиваете ей имя и тип данных. Кроме того, вы можете объявить переменную, присвоив ей имя, тип данных и значения. В одновременно определяете и присваиваете значение переменной при ее объявлении.

Некоторые типы объектов не могут быть определены как локальные переменные, но должны быть определены как типы в каталоге базы данных, как это обсуждается в Главе 14. Структуры – это составные переменные, такие как коллекции, структуры типа «запись» или системные курсорные переменные (system reference cursor). Структурами также могут быть локально именованные функции, процедуры и курсоры. Курсоры работают как небольшие функции. Курсоры имеют имена, сигнатуры и тип возвращаемого результата – выходные столбцы из запроса или оператора `SELECT`. Зарезервированное слово `DECLARE` начинает блок объявлений, а зарезервированное слово `BEGIN` завершает его.

Блок исполняемых команд позволяет обрабатывать данные. Блок исполняемых команд может содержать присвоения значений переменным, сравнения, условные операторы и итерации. Кроме того, блок исполняемых команд существует при доступе через курсор и другие именованные программные единицы. Функции, процедуры и некоторые типы объектов являются именованными программными единицами. Кроме того, можно производить вложение программ из анонимных блоков внутрь блока исполняемых команд. Зарезервированное слово `BEGIN` начинает блок исполняемых команд, а опциональное зарезервированное слово `EXCEPTION` или обязательное зарезервированное слово `END` завершает его. Вы должны иметь не менее одного предложения в блоке исполняемых команд. Следующее предложение минимального анонимного блока включает оператор `NULL`:

❐
```
BEGIN
NULL;
END;
/
```

Этот пример не выполняет ничего, кроме исполнения фазы компиляции без ошибок. В любом языке компиляция включает синтаксический разбор (парсинг). Отсутствие предложения в этом блоке вызовет синтаксическую ошибку, как это показано в Главе 5.

Блок обработки исключений позволяет обрабатывать исключения. Здесь можно перехватывать и обрабатывать их. Блок обработки исключений позволяет выполнить альтернативную обработку; во многих отношениях он действует как комбинация блоков catch и finally в языке программирования Java (в Приложении D дана информация по Java). Заре-

зервированное слово EXCEPTION начинает секцию, а зарезервированное слово END завершает ее.

Совет Нужно соблюдать – не менее одного предложения в любом блоке – в отношении блока с условным выражением (таким, как оператор IF), и в отношении блока с циклами.

Структура программы из именованных блоков – блочная структуру, потому что они сохраняются в базе данных. В них есть секция объявлений, известная как заголовок (header). Имя, список формальных параметров и тип возвращаемого значения именованных блоков определяются в заголовке. Имя и список формальных параметров известны как сигнатура подпрограммы. Область между заголовком и блоком исполняемых команд действует как блок объявлений именованного блока. Это правило справедливо и для текстовой части объектных типов, рассматриваемых в Главе 14.

Вот прототип функции с именованным блоком.

-
```
FUNCTION имя_функции
[(параметр1 [IN] [OUT] [NOCOPY] тип_данных_sql | тип_данных_plsql
, параметр2 [IN] [OUT] [NOCOPY] тип_данных_sql | тип_данных_plsql
, параметр(n+1) [IN] [OUT] [NOCOPY] тип_данных_sql | тип_данных_plsql)]

RETURN [тип_данных_sql | тип_данных_plsql)]

[AUTHID {DEFINER | CURRENT_USER}]
[DETERMINISTIC | PARALLEL_ENABLED]
[PIPELINED]
[RESULT_CACHE [RELIES ON имя_таблицы]] IS
список объявлений переменных
BEGIN
выполняемые команды
[EXCEPTION]
команды обработки исключений
END;
/
```

В Главе 6 обсуждаются правила, которым подчиняются функции. Функции могут работать как подпрограммы с параметрами, передаваемыми по значению, и параметрами, передаваемыми по ссылке. Подпрограммы с параметрами, передаваемыми по значению, определяют формальные параметры, используя только режим IN. Это означает, что переменная, которая была передана в подпрограмму, не может быть изменена в процессе выполнения программы. Подпрограммы с параметрами, передаваемыми по ссылке, определяют формальные параметры, используя режимы IN и OUT или только OUT.

Oracle 11g продолжает передавать копии переменных вместо ссылок на переменные, если не дано предписания NOCOPY. Oracle применяет передачу параметров по ссылке таким способом для того, чтобы гарантировать согласованность переменных в режиме IN OUT. Эта модель гаран-

тирует, что переменные не изменяются вплоть до успешного завершения выполнения подпрограммы. Вы можете переопределить этот способ, принимаемый по умолчанию, путем использования предписания NOCOPY. Корпорация Oracle не рекомендует использовать предписание NOCOPY, потому что это может привести к частичным изменениям значений фактических параметров. В конце концов, база данных выбирает, действовать ей в соответствии с заданным предписанием и передать ссылку.

Функции могут просматривать данные, используя оператор SELECT, но не могут выполнить операторы языка манипулирования данными (DML) INSERT, UPDATE или DELETE. Все остальные правила применяются к хранимым функциям аналогично правилам, применяемым к анонимным блокам. Функции, которые определяют формальные параметры или возвращаемые типы, используя типы данных PL/SQL, не могут быть вызваны из командной строки SQL. Однако можно вызывать функции, которые используют типы данных SQL, из командной строки SQL.

Значением по умолчанию для AUTHID является DEFINER, что известно как права определившего лица (definer rights). Это означает, что любой пользователь, имеющий право запускать на выполнение хранимую процедуру, запускает ее с теми же правами, что и пользователь, давший определение этой процедуры. Альтернатива в виде CURENT_USER позволяет пользователям с правами на запуск вызывать хранимую программу и запускать ее только в рамках данных, которые доступны этим пользователям/схемам. Это известно как права вызывающего лица (invoker rights) и описывает процесс вызова программ из публичного источника в рамках прав индивидуальных пользователей и их данных.

Не нужно использовать фразу DETERMINISTIC (детериминированная), если функции зависят от состояния переменных уровня сеанса. Фразы DETERMINISTIC наилучшим образом подходят для индексов, базирующихся на функциях, и материализованных представлений (materialized views).

Фраза PARALEL_ENABLED (распараллеленная) должна быть задействована для функций, которые вы планируете вызывать из предложений SQL и которые могут использовать возможности параллельных запросов. Нужно внимательно присмотреться к этой фразе при использовании хранилищ данных.

Фраза PIPELINED (конвейерная) предоставляет возможности повысить производительность, если функции возвращают коллекции, например, вложенные таблицы или v-массивы VARRAY. Кроме того, при применении фразы PIPELINED возрастет производительность в случае возвращения системных курсорных переменных.

Фраза RESULT_CACHE означает, что функция кэшируется только однажды в системной глобальной области (System Global Area, SGA) и доступна всем сеансам. Это новая возможность в базе данных Oracle 11g. Функции, доступные всем сеансам (кросс-сеансовые), работают только в режиме IN формальных параметров.

Глава 6 содержит детали применения этих фраз. Кроме того, приведены примеры их использования.

Вот прототип процедуры с именованным блоком.

```
PROCEDURE имя_процедуры
[(параметр1 [IN] [OUT] [NOCOPY] тип_данных_sql | тип_данных_plsql
, параметр2 [IN] [OUT] [NOCOPY] тип_данных_sql | тип_данных_plsql
, параметр(n+1) [IN] [OUT] [NOCOPY] тип_данных_sql | тип_данных_plsql)
[AUTHID {DEFINER | CURRENT_USER}]
список объявлений переменных
BEGIN
выполняемые команды
[EXCEPTION]
команды обработки исключений
END;
/
```

В Главе 6 обсуждаются правила, которые применяются к процедурам. Они действуют во многих отношениях как функции, но не могут возвращать данные. Это означает, что нельзя использовать их в качестве операндов справа. В отличие от функций, процедуры должны быть вызваны посредством блоков PL/SQL. Процедуры могут как просматривать данные, так и изменять их. Процедуры, кроме того, являются основой для создания подпрограмм, которые передают значения из и во внешние языки C, C++, Java и PHP.

В этом разделе были представлены и обсуждены основные структуры программных единиц PL/SQL. В следующем мы рассмотрим современные возможности базы данных Oracle 10g и новые возможности базы данных Oracle 11g.

Новые возможности Oracle 10g

В базу данных Oracle 10g было введено несколько изменений. Не все из них были доступны в период подготовки предыдущих изданий этой книги, потому что они начали поставляться только со вторым релизом базы данных.

В число новых возможностей базы данных, введенные в Oracle 10g, входят:

- Встроенные пакеты
- Сообщения времени исполнения
- Условная компиляция
- Правила обработки числового типа данных
- Оптимизированный компилятор PL/SQL
- Регулярные выражения
- Альтернативные кавычки
- Операторы, работающие с множествами

- Трассировка стека ошибок
- Упаковка хранимых программ PL/SQL

В данном разделе рассматриваются новейшие возможности, введенные в Oracle 10g. Кроме того, освящены функции базы данных Oracle 11g, которые обсуждаются ниже в этой главе.

Встроенные пакеты

Начиная с Oracle10g release 2, вы можете получить доступ к нескольким новым или улучшенным встроенным пакетам. Выделим три:

- `DBMS_SCHEDULER` заменяет встроенный DBMS_JOB и предоставляет новую функциональность при включении в расписание и выполнение пакетных заданий.
- `DBMS_CRYPTO` включает возможность шифрования и расшифровки больших объектов и поддерживает локализацию при использования нескольких таблиц кодировки.
- `DBMS_MONITOR` предоставляет трассировку, поддерживающую API и сбор статистики по сеансу.

Сообщения времени исполнения

Начиная с Oracle 10g release 1, можно анализировать производительность ваших программ на PL/SQL путем задания параметра `plsql_WARNINGS` в программах, которые вы разрабатываете. Вы можете определить этот параметр в сеансе или в базе данных. Последнее рекомендуется из-за ограничений, накладываемых на базу данных. Вы задаете этот параметр, используя следующую команду:

```
ALTER SESION SET plsql_warnings = 'enable:all';
```

Условная компиляция

Начиная с Oracle 10g release 2, можно использовать условную компиляцию. Условная компиляция позволяет включать отладочную логику или специальную логику, которая запускается, если установлены переменные уровня сеанса. Следующая команда определяет переменную PL/SQL времени компиляции `DEBUG` равной 1:

```
ALTER SESSION SET plsql_CCFLAGS = 'debug:1';
```

Эта команда устанавливает переменную PL/SQL времени компиляции, равной 1.

Нужно помнить, что флаг времени компиляции нечувствителен к регистру. Кроме того, можно присвоить значения `true` (истина) и `false` (ложь) переменным времени компиляции, и они будут действовать подобно булевым переменным. Если вы хотите использовать более одного флага условной компиляции, нужно использовать следующий синтаксис:

```
❒ ALTER SESSION SET plsql_CCFLAGS = 'имя1:значение1
[,имя(n+1):значение(n+1)]';
```

Параметры условной компиляции хранятся, как пары имя-значение в параметре базы данных `plsql_CCFLAGS`. Следующая программа использует директивы `$IF`, `$THEN`, `$ELSE`, `$ELSIF`, `$ERROR` и `$END` для создания блока кода с условной компиляцией:

```
❒ BEGIN
  $IF $$DEBUG = 1 $THEN
  dbms_output.PUT_LINE('Включен уровень отладки 1.');
  $END
  END;
  /
```

Блоки с условной компиляцией отличаются от обычных блоков кода с `if-then-else`. Примечательно, что директива `$END` закрывает блок вместо `END IF` и точки с запятой, как это показано в Главе 4. Кроме того, нужно обратить внимание на то, что символ $$ отмечает переменную PL/SQL времени условной компиляции.

Правила, которым подчиняется условная компиляция, устанавливается парсером SQL. Нельзя использовать условную компиляцию с объектными типами SQL. Это ограничение также накладывается на вложенные таблицы и `VARRAY` (скалярные таблицы). Условная компиляция различается в функциях и процедурах. Логика меняется в зависимости от того, имеет функция или процедура список формальных параметров. Вы можете использовать условную компиляцию после открытия скобок, окружающих список формальных параметров, например:

```
❒ CREATE OR REPLACE FUNCTION тип_условия
  (условное_число $IF $$DEBUG = 1 $THEN SIMPLE_NUMBER $ELSE NUMBER $END)
  RETURN NUMBER IS
  BEGIN
  RETURN условное_число;
  END;
  /
```

Альтернативный вариант – использовать их после ключевых слов AS или IS в функциях и процедурах без списка параметров. Кроме того, они могут быть использованы и внутри списка формальных параметров, и после AS и IS в функциях и процедурах с параметрами.

Условная компиляция может иметь место только после ключевого слова `BEGIN` в триггерах и программных единицах с анонимными блоками. Прошу заметить, что вы не можете инкапсулировать замещаемую переменную или переменную связи внутрь блока с условной компиляцией.

Глава 4 содержит примеры использования метода условной компиляции.

Правила обработки числового типа данных

Начиная с Oracle 10g release 1, база данных использует машинную арифметику для BINARY_INTEGER, INTEGER, INT, NATURAL, NATURALN, PLS_INTEGER, POSITIVE, POSITIVEN и SIGNTYPE. Это означает, что теперь они используют такую же точность, как и тип данных BINARY_INTEGER. В предыдущих версиях базы данных эти типы обрабатывались как тип данных NUMBER, они использовали ту же математическую библиотеку C, что и тип данных NUMBER. Новые версии этих типов данных могут участвовать в сравнении с бесконечностью или с NaN (не число, not a number).

Оборотная сторона этого изменения: сейчас они используют числовую точность, а не десятичную. По этой причине финансовые приложения должны использовать тип данных NUMERIC.

BINARY_FLOAT обычной точности и BINARY_DOUBLE удвоенной точности также присутствуют в базе данных Oracle 10g. Они используются для математических и научных расчетов.

Оптимизированный компилятор PL/SQL

Начиная с Oracle 10g release 1, база данных оптимизирует вашу компиляцию PL/SQL. Это задается по умолчанию и применяется как к интерпретируемому p-коду, так и к собственному скомпилированному коду PL/SQL (natively compiled PL/SQL CODE). Можно отменить или изменить степень активности оптимизатора путем перезадания параметра plsql_OPTIMIZE_LEVEL. В Таблице 1.1 объяснены три возможных значения этого параметра.

Вы можете отключить оптимизацию на время сеанса путем применения следующей команды

```
ALTER SESSION SET уровень_оптимизации_plsql = 0;
```

Кроме того, можно задать уровень оптимизации для процедуры. Например

```
ALTER PROCEDURE некоторая процедура COMPILE уровень_оптимизации_
plsql=1;
```

После задания уровня оптимизации, можно использовать фразу REUSE SETTINGS для предыдущей установки, по аналогии с:

```
ALTER PROCEDURE некоторая процедура COMPILE REUSE SETTINFS;
```

Рекомендуется установка по умолчанию. Оптимизированный код всегда исполняется быстрее не оптимизированного.

Примечание Уровень plsql_OPTIMIZE_LEVEL должен всегда быть установлен равным 2 или выше для автоматического внедрения подпрограмм (automatic subprogram inlining), которое имеет место в базах данных Oracle 10g и 11g.

Таблица 1.1 *Доступные значения параметра* `plsql_OPTIMIZE_LEVEL`

Уровень оптимизации	Смысл оптимизации
0	Не оптимизировать
1	Умеренная оптимизация, может удаляться неиспользуемый код или исключения
2 (по умолчанию)	Активная оптимизация, может изменять порядок исполнения исходного кода.

Регулярные выражения

Начиная с Oracle 10g release 1, база данных поддерживает набор функций, работающих с регулярными выражениями. Вы можете использовать их как в операторах SQL, так и в программных единицах PL/SQL. К таким функциям относятся:

- `REGEXP_LIKE` просматривает строку для отыскания соответствия образцу, заданному регулярным выражением.
- `REGEXP_INSTR` – ищет начальную позицию, с которой начинается совпадение с образцом, заданным регулярным выражением.
- `REGEXP_REPLACE` – производит замену одной подстроки на другую в случае нахождения соответствия образцу, заданному регулярным выражением.

Это важные функции. В Приложении E обсуждаются, рассматриваются и демонстрируются регулярные выражения, которые используются в функциях базы данных Oracle 11g для работы с регулярными выражениями.

Альтернативные кавычки

Начиная с Oracle 10g release 1, база данных позволяет заменять обычную одиночную кавычку другим символом. Это полезно, если у вас много апострофов в строке и они требуют индивидуального дублирования вторым апострофом. Старый способ:

```
SELECT 'It''s a bird, no plane, no it can''t be ice cream!' AS phrase
FROMdual;
```

Новый способ:

```
SELECT q'('It's a bird, no plane, no it can't be ice cream!)' AS phrase
FROMdual;
```

В обоих случаях на выходе мы получим следующий результат:

```
PHRASE
--------------------------------------------------------------------------
It's a bird, no plane, no it can't be ice cream!
```

Можно использовать новый синтаксис и сэкономить время, но старый способ тоже можно применять. Старый метод шире распространен и более мобилен.

Операторы, работающие с множествами

Начиная с Oracle 10g release 1, база данных поддерживает для вложенных таблиц операторы, работающие с множествами. Это операторы MULTISET EXCEPT, MULTISET INTERSECT, MULTISET UNION и MULTISET UNION DISTINCT. MULTISET UNION работает как оператор UNION ALL: возвращает две копии всего, находящегося в области пересечения (intersection) двух множеств (set) и одну копию собственных компонентов. MULTISET UNION DISTINCT работает подобно оператору UNION: возвращает одну копию путем выполнения операции сортировки. В Главе 7 эти операторы рассматриваются при обсуждении коллекций.

Трассировка стека ошибок

Начиная с Oracle 10g release 1, база данных позволяет сформировать отформатированную трассировку стека. Трассировка стека формирует список ошибок, начиная с исходного вызова, до места, где была сгенерирована ошибка. Для формирования трассировки стека используйте функцию DBMS_UTILITY.FORMAT_ERROR_BACKTRACE. Кроме того, вы можете вызвать FORMAT_CALL_STACK или FORMAT_ERROR_STACK из того же пакета для обработки сгенерированных исключений.

Вот простой пример:

```
DECLARE
локальное_исключение EXCEPTION;
FUNCTION вложенная_локальная_функция
RETURN BOOLEAN IS
возвращаемое_значение BOOLEAN:=FALSE;
BEGIN
RAISE локальное_исключение
RETURN возвращаемое_значение;
END;
BEGIN
IF вложенная_локальная_функция THEN
dbms_output.PUT_LINE('Исключения не сгенерированы');
END IF;
EXCEPTION
WHEN others THEN
dbms_output.PUT_LINE('DBMS_UTILITY.FORMAT_CALL_STACK');
dbms_output.PUT_LINE('----------------------------------------------')
;
dbms_output.PUT_LINE(dbms_utility.format_call_stack);
dbms_output.PUT_LINE('DBMS_UTILITY.FORMAT_ERROR_BACKTRACE');
dbms_output.PUT_LINE('----------------------------------------------')
;
```

```
dbms_output.PUT_LINE(dbms_utility.format_error_backtrace);
dbms_output.PUT_LINE('DBMS_UTILITY.FORMAT_ERROR_STACK');
dbms_output.PUT_LINE('----------------------------------------------')
;
dbms_output.PUT_LINE(dbms_utility.format_error_stack);
END;
/
```

Этот скрипт напечатает следующее

```
DBMS_UTILITY.FORMAT_CALL_STACK
------------------------------------------------------
----- PL/SQL Call Stack -----
object line object
handle number
name
20909240 18 ANONYMOUS block

DBMS_UTILITY.FORMAT_ERROR_BACKTRACE
-----------------------------------------------------------------
ORA-06512: at line 7
ORA-06512: at line 11

DBMS_UTILITY.FORMAT_ERROR_STACK
-------------------------------------------------------
ORA-06510: PL/SQL: unhandled user-defined exception

DBMS_UTILITY.FORMAT_CALL_STACK
------------------------------------------------------
----- Стек вызовов PL/SQL -----
имя номер объект
описателя строки
объекта
20909240 18 анонимный блок

DBMS_UTILITY.FORMAT_ERROR_BACKTRACE
-----------------------------------------------------------------
ORA-06512: в строке 7
ORA-06512: в строке 11

DBMS_UTILITY.FORMAT_ERROR_STACK
-------------------------------------------------------
ORA-06510: PL/SQL: необработанная пользовательская ошибка
```

Вероятно, будет наиболее полезна FORMAT_ERROR_BACKTRACE, – фиксирует строку, в которой была первоначально сгенерирована ошибка, и проходит сквозь все вызовы до тех пор, пока не достигнет начального вызова. Одновременно показываются номер строки и имя программы, если в стеке событий участвуют именованные блоки. В Главе 5 дана дальнейшая информации об обработке ошибок.

Упаковка хранимых программ на PL/SQL

Начиная с Oracle 10g release 2, база данных поддерживает возможность упаковки (кодирования) (wrap), или сокрытия, хранимых программ на PL/SQL. Это реализуется путем применения процедуры CREATE_WRAPPED пакета DBMS_DDL. Применяйте ее согласно примеру:

```
BEGIN
dbms_ddl.create_wrapped(
'CREATE OR REPLACE FUNCTION hello_world RETURN STRING AS '
||'BEGIN'
||' RETURN ''Hello World!'';'
||'END;');
END;
/
```

Когда функция создана, вы можете вызвать ее, используя следующее форматирование столбца и запрос в SQL*Plus:

```
SQL>COLUMN message FORMAT A20 HEADING «Сообщение»
SQL>SELECT hello_world AS message FROMdual;

Сообщение
------------------------------
Hello World!
```

Вы можете получить информацию о функции путем просмотра ее сигнатуры и типа возвращаемого результата:

```
SQL>DESCRIBE hello_world
FUNCTION hello_world RETURNS VARCHAR2

ФУНКЦИЯ hello_world возвращает VARCHAR2
```

Любые попытки изучить детали программного кода функции дадут нечитабельный результат. Вы можете проверить это путем запроса текста хранимой функции в столбце TEXT таблицы USER_SOURCE, например, следующим образом:

```
SQL>COLUMN text FORMAT A80 HEADING «Исходный текст»
SQL>SET PAGESIZE 49999
SQL>SELECT text FROMuser_source WHERE name='HELLO_WORLD';
```

Возвращается следующий результат:

```
FUNCTION hello_world wrapped
a000000
369
abcd
. . . . . et cetera . . . .
```

Это очень удобная утилита для сокрытия деталей программного кода от посторонних глаз. Мы вернемся к этому вопросу в Приложении F.

Новые возможности Oracle 11g

К новым возможностям PL/SQL, введенным в Oracle 11g, относятся:

- Автоматическое встраивание подпрограмм (inlining)
- Оператор `CONTINUE`
- Межсеансовый кэш результатов работы функций PL/SQL (Cross-Session PL/SQL Function Result Cache)
- Расширение динамического SQL
- Вызовы в SQL в смешанной нотации – по именам и по позиции
- Многопроцессный пул соединений (multiprocess connection pool)
- Иерархический профайлер PL/SQL (PL/SQL Hierarchical Profiler)
- Собственный компилятор PL/SQL (PL/SQL Native Compiler) генерирует собственный код (native `CODE`)
- PL/Scope
- Расширение регулярных выражений
- Тип данных `SIMPLE_INTEGER`
- Прямое обращение к последовательности (sequence) в операторах SQL.

Эти усовершенствования кратко рассматриваются в следующих разделах. В Главе 3 рассматривается тип данных `SIMPLE_INTEGER`. В Главе 4 обсуждается оператор continue. В Главе 6 демонстрируется межсеансовый кэш результатов функций PL/SQL, а также смешанная, по именам и по позиции нотация вызовов. В Главе 9 рассматривается вопрос об автоматическом встраивании подпрограмм (inlining) и собственный компилятор PL/SQL (PL/SQL Native Compiler). В Главе 16 обсуждаются разработка web-приложений и многопроцесный пул соединений (multiprocess connection pool). Вы можете найти дальнейшую информацию по регулярным выражениям, иерархическому профайлеру PL/SQL и средству PL/Scope в Приложениях E, G и H соответственно.

Автоматическое встраивание подпрограмм

Встраивание (inlining) подпрограммы заменяет вызов внешней подпрограммы копией этой подпрограммы. Это почти всегда повышает производительность программы. Начиная с базы данных Oracle 11g, вы можете предложить компилятору встраивать подпрограммы, используя директиву PL/SQL `PRAGMA_INLINE`. Вы должны задать параметр `PRAGMA`, если установили параметр `plsql_OPTIMIZE_LEVEL` равным 2.

Предположим, что у вас в схеме имеется хранимая функция `ADD_NUMBERS`, тогда вы можете предписать программной единице PL/SQL встроить вызов функции `ADD_NUMBERS`. Это полезно, если вы вызываете функцию `ADD_NUMBERS` в цикле, как в данном примере:

```
CREATE OR REPLACE PROCEDURE INLINE_demo
(a NUMBER
,b NUMBER ) IS
PRAGMA INLINE(add_numbers,'YES');
BEGIN
FOR i IN 1..10000 LOOP
dbms_output.PUT_LINE(add_function(8,3));
END LOOP;
END;
/
```

База данных автоматически выбирает встраивание, если вы устанавливаете параметр `plsql_OPTIMIZE_LEVEL` равным 3. Это освобождает вас от определения тех случаев, когда необходимо встраивать вызовы функций. Однако, это всего лишь рекомендации компилятору. Рекомендуем позволить интерпретатору оптимизировать ваш код во время компиляции.

Оператор `CONTINUE`

Добавление оператора `CONTINUE` в язык PL/SQL вызывает смешанные эмоции. Существует мнение, что оператор `CONTINUE` приводит к неоптимальному программированию, но в большинстве случаев он упрощает структуру циклов.

Оператор `CONTINUE` дает сигнал немедленно прекратить выполнение текущей итерации цикла и вернуться на первый оператор в цикле. Следующий анонимный блок показывает использование оператора `CONTINUE` в момент, когда счетчик цикла является четным числом:

```
BEGIN
FOR i IN 1..5 LOOP
dbms_output.PUT_LINE('Первое предложение, индекс равен ['||i||'].');
IF MOD(i,2)=0 THEN
CONTINUE;
END IF;
dbms_output.PUT_LINE('Второе предложение, индекс равен ['||i||'].');
END LOOP;
END;
/
```

Функция `MOD` возвращает ноль при делении любого четного числа. Поэтому вторая строка никогда не будет напечатана, поскольку оператор `CONTINUE` прерывает дальнейшие итерации. Дальнейшая информация об использовании этой команды содержится в Главе 4. Функция `MOD` рассматривается в Приложении J.

Межсеансовый кэш результатов работы функций PL/SQL

Межсеансовый кэш результатов работы функций PL/SQL является механизмом для совместного доступа сеансов через SGA к часто запрашивае-

мым функциям. До базы данных Oracle 11g, каждый вызов функции с тем же самым набором фактических параметров, или значений времени выполнения, помещался в кэш один раз за сессию. Единственное, что нужно сделать при использовании данной функциональности лишь методы доступа.

Вы можете задать любую из следующих конструкций для кэширования результатов:

- `RESULT_CACHE условие`

  или

- `RESULT_CACHE RELIES_ON(имя_таблицы)`

Фраза (clause), заданная в операторе `RELIES_ON`, накладывает ограничение на кэшированный результат. Любое изменение таблицы, на которую дана ссылка, делает некорректной функцию, а также все функции, процедуры и представления, которые основываются на этой функции.

Расход ресурсов при вызове функции в первый раз такой же, как и при вызове некэшированного результата. Кроме того, кэш имеет ограничение времени существования в SGA, если он более не запрашивается активно сеансами.

Расширение динамического SQL

Динамический SQL имеет две разновидности в базе данных Oracle 11g: собственный динамический SQL (Native Dynamic SQL), также известный как NDS, и встроенный пакет `DBMS_SQL`. Обе разновидности были усовершенствованы в последнем релизе.

Собственный динамический SQL

В Oracle 11g собственный динамический SQL поддерживает операторы больше 32 Кбайт используя тип данных `CLOB`. Вы получаете доступ к этой возможности, используя вместо оператора SQL следующий синтаксис:

- `OPEN имя_курсора FOR динамическая_строка`

Динамическая строка (dynamic string) может быть `CHAR`, `VARCHAR2` или `CLOB`. Это устраняет ограничение предыдущих версий, которое лимитировало размер динамически созданной строки.

Встроенный пакет DBMS_SQL

Несколько изменений сделали еще более полезным пакет `DBMS_SQL`. Начиная с Oracle 11g, можно использовать все типы данных, поддерживаемые NDS. Кроме того, можно использовать процедуру `PARSE` для работы со строками больше 32 Кбайт. Это стало возможным благодаря применению типа данных `CLOB`. `CLOB` заменил метод, использующий таблицы с типом данных `VARCHAR2` (обычно `VARCHAR2A` или `VARCHAR2S`). К счастью, пакет `DBMS_SQL` продолжает поддерживать старый метод, но вы должны задуматься о переходе на более современные решения.

В DBMS_SQL добавлены две новые функции: TO_REFCURSORS и TO_CURSOR_NUMBER. Они позволяют преобразовывать курсорные переменные в курсоры и обратно. Разумеется, имеются веские аргументы для применения этих функций. Вы должны открыть курсор или системную курсорную переменную (system reference cursor) перед их использованием, а после их использования, вы уже не можете получить доступ к их структурам. Код переопределяет интерактивную ссылку с курсора на системную курсорную переменную или с системной курсорной переменной на курсор.

Последний по порядку, но не по значимости, аргумент: вы можете выполнять массовые операции связывания над пользовательскими коллекционными типами. Коллекционные типы могут быть массивами скалярных величин. Ранее вы были ограничены типами, определенными спецификацией пакета DBMS_SQL.

Вызовы в SQL в смешанной нотации – по именам и по позиции

В базе данных Oracle 11g изменен порядок применения нотации по именам и нотации по позиции как в SQL, так и в PL/SQL. Сейчас они действуют одинаково в SQL и PL/SQL. Это исправило давно существующую странность базы данных Oracle.

Вызовы в PL/SQL

Ранее у вас был выбор из двух возможностей: перечислить все параметры по порядку или обратиться к некоторым параметрам по их именам. Сейчас вы можете использовать ссылку на позицию, ссылку по имени или смесь этих методов.

Следующая функция позволит вам экспериментировать с различными подходами. Эта функция принимает три необязательных параметра и возвращает сумму трех чисел.

```
CREATE OR REPLACE FUCTION add_three_numbers
(a NUMBER:=0, b NUMER:=0, c NUMBER:=0) RETURN NUMBER IS
BEGIN
RETURN a+b+c;
END;
/
```

Показано, как можно использовать вызовы функции, применяя нотацию по позиции, по имени и смешанную. В данном примере вы передаете фактические параметры, соответствующие каждому из формальных параметров, определенных в функции.

Кроме того, вы можете исключить один или несколько параметров, поскольку все параметры определены как необязательные, а это означает, что они имеют значения по умолчанию. Этот пример приведен в разделе «Нотация по выбору».

Нотация по позиции

Вы можете вызвать функцию, используя нотацию по позиции

```
BEGIN
dbms_output.PUT_LINE(add_three_numbers(3,4,5));
END;
/
```

Нотация по имени

Вы можете вызвать функцию, используя нотацию по имени

```
BEGIN
dbms_output.PUT_LINE(add_three_numbers(c=>4,b=>5,a=>3));
END;
/
```

Смешанная нотация

Вы можете вызвать функцию, используя смешанную нотацию

```
BEGIN
dbms_output.PUT_LINE(add_three_numbers(3, c=>4,b=>5));
END;
/
```

Имеется ограничение на использование смешанной нотации. Все фактические параметры, записанные в нотации по позиции, должны находиться впереди и в том порядке, в каком они определены в функции. Вы не можете задать параметр по позиции после параметра, заданного по имени.

Нотация по выбору

Как уже упоминалось, вы можете исключить один или несколько фактических параметров, если формальные параметры определены как необязательные. Все параметры в функции `ADD_THREE_NUMBERS` определены как необязательные. В следующем примере первый параметр передается по ссылке на позицию, а третий параметр – по ссылке на имя:

```
BEGIN
dbms_output.PUT_LINE(add_three_numbers(3, c=>4));
END;
/
```

Если вы решаете не передавать фактический параметр, это означает, что вы передаете неопределенное значение (`null`). Это известно, как нотация по выбору. Ранее существовала рекомендация перечислять необязательные параметры в заголовках функций и процедур последними по порядку. В настоящее время вы можете исключить один или два, а не все необязательные параметры. Это значительное усовершенствование, но будьте осторожны, применяя его.

Нотация вызовов в SQL

Ранее вы должны были перечислить все параметры в порядке их расположения, потому что не могли использовать ссылки на имена в SQL.

В Oracle 11g можно осуществлять вызов так, как в PL/SQL. Следующий пример показывает смешанную нотацию при вызове в SQL:

```
SELECT add_three_numbers(3,c=>4,b=>5) FROMdual;
```

Как и в предыдущих версиях, вы можете вызывать из SQL функции, которые имеют параметры только в режиме IN. Вы не можете вызывать из SQL функции, в которых хоть один из параметров определен как переменная в режиме IN OUT или только OUT. Это вызвано тем, что вы должны передавать ссылку на переменную, если параметр использует режим OUT. Функция возвращает ссылку на переменную в режиме OUT как фактический параметр.

Многопроцессный пул соединений

Модули Enterprise JavaBeans (EJBs) стали еще лучше при введении многопроцессного пулинга соединений (multiprocess conection pooling) в базу данных Oracle 11g. Это официальный резидентный пулинг соединений базы данных (database ReSIDent Connection Poling, DRCP). Эта возможность позволяет обслуживать еще более масштабируемый пул соединений на стороне сервера. До этой версии вам приходилось использовать разделяемые серверные процессы или многопоточный сервлет Java (multithreaded Java Servlet).

Многопроцессный пул соединений значительно сокращает объем занимаемой памяти на уровне базы данных, кроме того, он улучшает масштабируемость как на промежуточном уровне, так и на уровне базы данных. Стандартное соединение с базой данных требует 4.4 Мбайт оперативной памяти; 4 Мбайт выделяются физическому соединению и 400 Кбайт для сеанса пользователя. Следовательно, 500 выделенных параллельных соединений потребуют приблизительно 2.2 Гбайт оперативной памяти. Модель с разделяемым сервером лучше масштабируется и требует только 600 Мбайт оперативной памяти при том же количестве конкурентных пользователей, 80% этой памяти будет обслуживаться в разделяемой глобальной области Oracle (Shared Global Area, SGA). Резидентный пулинг соединений базы данных еще лучше масштабируется и потребует только 400 Мбайт оперативной памяти. Ясно, что для web-ориентированных приложений DRCP предпочтительнее, особенно при использовании постоянных соединений OCI8.

Поведение этих моделей под нагрузкой определяет их соответствующую масштабируемость. На Рис. 1.2 показано использование памяти в этих трех моделях при изменении числа конкурентных пользователей с 500 до 2000.

Встроенный пакет DBMS_CONNECTION_POOL позволяет запускать, останавливать и конфигурировать параметры пулинга соединений – лимит размера и времени. Вы можете запустить пул соединений как пользователь SYS, используя следующую команду:

```
SQL>EXECUTE DBMS_CONNECTION_POOL.START_POOL();
```

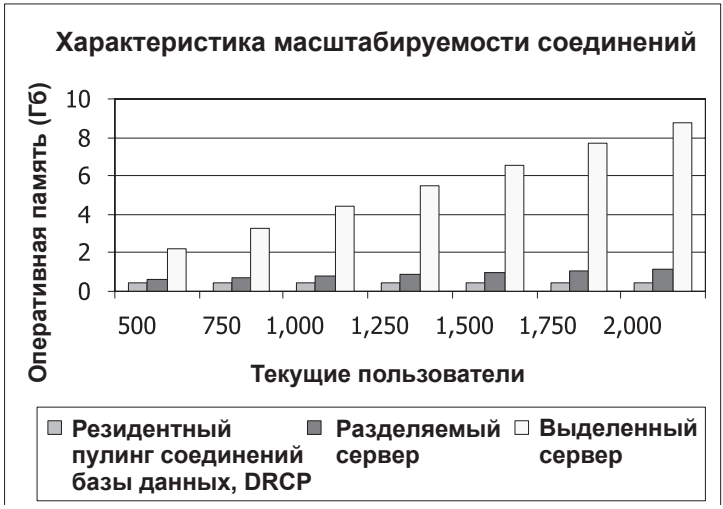


Рис. 1.2 *Масштабируемость соединений*

Вам должен быть доступен файл tnsnames.ora для поддержки соединений к разделяемому пулу. Следующий пример показывает, как сделать доступным дескриптор соединения разделяемого пула (shared pool connection descriptor), при условии, что вы подставите правильное имя узла (hostname), домен и номер порта наблюдателя Oracle (Oracle listener port number):

```
ORCLCP -
(DESCRIPTION -
(ADDRESS - (PROTOCOL - TCP)
(HOST - имя_узла.домен)
(PORT - номер_порта)
)
(CONNECT_DATA - (SERVER - POOLED)
(SERVICE_NAME - orcl)
)
)
```

Ключ `SERVER` в дескрипторе соединения пере направляет соединение в службу пулинга соединений. Использовать пул соединений вы можете только в том случае, когда у вас имеется поддерживаемая база данных Oracle 11g или вы клиент Oracle 11g, хотя не исключено, что эта возможность впоследствии получит поддержку. Следующая ошибка генерируется при попытке установить соединение путем использования более старой версии клиента или библиотеки сервера:

```
ОШИБКА:
ORA-56606: DRCP: Версия клиента не поддерживает эту возможность
```

Это сообщение сервер подает в случае, когда он не может создать подходящий сокет; оно означает, что он отверг удаленный запрос к пулу соединений.

Таблица 1.2 дает список словарных представлений (dictionary view) для аудирования пула соединений. Вы можете использовать их для мониторинга соединений или характеристик производительности.

Для остановки пула соединений вы как пользователь SYS должны использовать следующую команду:

```
SQL>EXECUTE DBMS_CONNECTION_POOL.STOP_POOL();
```

Таблица 1.2 *Словарные представления пулинга соединений*

Представление	Описание
DBA_CPOOL_INFO	Для каждого пула соединений это представление показывает статус, максимальные и минимальные соединения и время простоя
V$CPOOL_STAT	Это представление показывает число запросов сеанса, сколько раз сеанс, соответствующий запросу, найден в пуле и общее время ожидания в расчете на одну сессию
V$CPOOL_CC_STATS	Статистика соединения уровня класса для каждого экземпляра пула соединений

Оказалось, что исходный релиз будет поддерживать только один пул соединений. Имя этого пула соединений – SYS_DEFAULT_CONNECTION_POOL. Кроме того, имеются три других процедуры в пакете DBMS_CONNECTION_POOL для управления пулом соединений: процедуры ALTER_PARAM(), CONFIGURE_POOL() и RESTORE_DEFAULTS(). С помощью процедуры ALTER_PARAM() вы можете изменить один параметр пула соединений. Если вы хотите изменить более одного параметра, нужно использовать процедуру CONFIGURE_POOL(). Процедура RESTORE_DEFAULTS() переопределяет все параметры пула соединений в их значения по умолчанию.

Новая возможность базы данных Oracle 11g определенно улучшила масштабируемость web-приложений. Она необходима, чтобы сделать возможным использование постоянного соединения (persistent connection), введенного в библиотеки OCI8 в базе данных Oracle 10g release 2.

Иерархический профайлер PL/SQL

Иерархический профайлер (Hierarchical Profiler), поставляемый с базой данных Oracle 11g, позволяет видеть, как работают все компоненты приложения. Он отличается от неиерархических (плоских) профайлеров, которые просто записывают время, которое потрачено каждым модулем. Иерархический профайлер следует за циклом выполнения, начиная с главной программы до подпрограммы самого низкого уровня.

Иерархический профайлер PL/SQL (PL/SQL Hierarchical Profiler) делает следующее:

- создает отчет по динамическому профилю выполнения вашей программы PL/SQL, упорядоченный по вызовам подпрограмм;
- разделяет время исполнения SQL и PL/SQL и помещает его в разные отчеты;

- не требует специальной подготовки в коде или во время компиляции, такой как оператор PRAGMA, необходимой для рекомендуемого встраивания;
- сохраняет результаты в наборе таблиц базы данных, которые вы можете использовать для разработки инструментов создания отчетов или использовать инструмент командной строки plshprof для генерации простых отчетов HTML.

Итоговый отчет по подпрограммам содержит информацию о количестве вызовов подпрограмм, времени их исполнения или принадлежащих им подпрограмм более низкого уровня и детализированную информацию между модулями. В Приложении G рассматривается, как читать данные и использовать иерархический профайлер PL/SQL.

Собственный компилятор PL/SQL генерирует собственный код

Собственная компиляция PL/SQL изменилась в базе данных Oracle 11g. В отличие от предыдущих версий, в которых PL/SQL сначала транслировался в код C, а затем компилировался, сейчас вы можете компилировать напрямую. Скорость выполнения итогового кода в некоторых случаях удваивалась или даже возрастала еще больше.

Oracle рекомендует запускать все программы PL/SQL или в режиме NATIVE, или в режиме INTERPRETED. Для базы данных по молчанию устанавливается режим INTERPRETED, а модули PL/SQL хранятся как текст или упакованный (wrapped) текст. Вы можете просматривать хранимые программы путем запроса к словарным таблицам данных ALL_SOURCE, DBA_SOURCE или USER_SOURCE. Код в режиме NATIVE компилируется в промежуточную форму до преобразования в машинно-зависимый код. Копия этого кода также сохраняется в словаре данных (data dictionary), если библиотека размещается во внешней директории. Вы можете задать соответствие физической и виртуальной директории, определенной параметром базы данных plsql_NATIVE_LIBRARY_DIR.

Код, скомпилированный в собственную форму, имеет преимущества если время исполнения PL/SQL значительное. Это может случиться при использовании кода, включающего интенсивные вычисления, но обычно задержки в исполнении вызваны обработкой операторов SQL. Вы должны использовать новый иерархический профайлер PL/SQL, чтобы выяснить, стоит ли тратить усилия на конвертирование.

PL/Scope

PL/Scope – это инструмент, управляемый компилятором, который позволяет проверять идентификаторы и их поведение в базе данных Oracle 11g. По умолчанию он отключен. Вы можете рассмотреть включение его на уровне сеанса, но не базы данных. Вы можете включить его, используя следующую команду:

```
ALTER SESSION SET PLSCOPE_SETTINGS='IDENTIFIER:ALL';
```

Утилита PL/Scope не занимается сбором статистики до тех пор, пока вы не включите ее. Эта статистика позволяет понять, как ваша программа использует идентификаторы. Она должны быть включена в ваш арсенал средств оптимизации. В Приложении H дано краткое введение в эту новую возможность.

Расширение регулярных выражений

Oracle 11g release 1 расширяет возможности функций REGEXP_LIKE и REGEXP_INSTR и вводит функцию REGEXP_COUNT. В Приложении E обсуждаются, рассматриваются и демонстрируются регулярные выражения на примере функций регулярных выражения базы данных Oracle 11g.

Тип данных SIMPLE_INTEGER

В базе данных Oracle 11g введен тип SIMPLE_INTEGER. Это производный от BINARY_INTEGER тип и он имеет тот же диапазон значений. В отличие от BINARY_INTEGER, SIMPLE_INTEGER исключает неопределенные значение (null), а переполнение отсекается. Кроме того, отсечение переполнения не вызывает генерацию ошибки. Вы должны использовать тип SIMPLE_INTEGER, если хотите произвести компиляцию вашей программы в собственный код, поскольку это приводит к существенному повышению производительности компилированного кода.

В этом разделе рассмотрены новые возможности базы данных Oracle 11g даны ссылки на другие части книги для получения дальнейшей информации. Очевидно, что эти новые возможности делают обновление версии весьма привлекательной идеей.

Прямое обращение к последовательности в операторах

Oracle11g позволяет обращаться к последовательности (sequence), используя .nextval и .currval внутри команды SQL, а это означает, что вы можете делать это так:

```
SELECT следующее_значение_некоторой_последовательности.nextval
INTO некоторая_локальная_переменная
FROMdual;
```

В этой книге используются и новый, и старый стили. Новый стиль проще и удобнее в применении.

Итоги

В этой главе рассмотрены история, преимущества, основы кодирования и последние возможности, добавленные в язык программирования PL/SQL. Кроме того, объяснена важность PL/SQL и то, как он может повысить отдачу от ваших инвестиций в базу данных Oracle 11g. Теперь вы должны понимать, что комбинация из SQL и PL/SQL может упростить разработку ваших проектов внешних приложений, осуществляемую на языках Java и PHP.